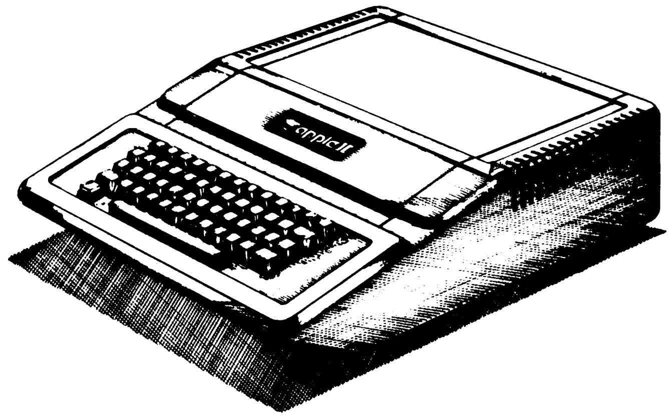
Apple II Family Technical Documents



# Apple Assembly Line Article Archive

Written by Bob Sander-Cederlof
from October 1980 through May 1988

This archive contains issues
for October 1980 to June 1986

Source:
http://salfter.dyndns.org/aal/
salfter@salfter.dyndns.org
15 September 2000

# Apple Assembly Line Archive

A while back, I downloaded all of the issues of Apple Assembly Line that had been archived in GEnie's A2Pro file area. At this point, GEnie is either dead or dying (last time I used it was a few years ago). Delphi's A2Pro might eventually get them, but it hasn't happened yet.

Until that time, I've put them all here. The only change from the way they were presented on GEnie is that I renamed the files so that a directory listing of them could easily be sorted chronologically...instead of "AAL.JAN.85.BXY," for instance, that file is now "AAL.8501.BXY." The info about each issue given in this HTML document is, if I remember right, the description that GEnie had used for the file.

So, without further ado, here's the archive. The whole lot is only about 2.5MB, so you can either just click away at the links or use something like Go!Zilla (no, Go!Zilla isn't an Apple II program) to "leech" all of 'em in one swell foop. :-)

The entire collection is also available as a single ZIP archive. It's mainly of benefit to non-Apple II users who might want to browse the collection. (There are unzip programs for the II, but ShrinkIt files are better if you're working with this stuff on a II.)

Also, I received email on 2 Nov 99 from Bob Sander-Cederlof, the author of most of these files. It turns out that publication ceased sometime in 1988, which means I'm missing a few files. If you have 'em and can send 'em to me, I'd be interested...send me mail.

AAL.ZIP
The entire collection as a single (~2MB) file.

AAL.8010.BXY
This issue contains articles on alternate ways to add and subtract one from a number, a general message printing subroutine, some S-C Macro Assembler patches and a hardware error in the JMP (addr) instruction in all 6502 chips (one of the first publications of this bug!).

AAL.8011.BXY
This issue contains articles on bugs and new commands for the S-C Macro Assembler, a new USR command for that assembler, instructions for turning S-C files into text source files, a variable cross-reference generator for Applesoft programs and a simulated numeric keypad for the Apple II+, all in 6502 assembly!

AAL.8012.BXY
This issue contains articles on intelligent disassemblers, a pretty LIST for Integer BASIC, new commands and directives for the S-C Macro Assembler and ways to handle 16-bit comparisons on an 8-bit machine.

AAL.8101.BXY
This issue contains articles on how to move memory, a computed GOSUB for Applesoft and putting a new COPY and EDIT into the S-C Macro Assembler.

AAL.8102.BXY
This issue contains articles on making all kinds of noises with the Apple II speaker (tones, bells, machine guns, swoops, lasers, inch-worms, touch-tones

and morse code)! It also has stuffing object code in protected places, multiplying on the 6502 and string swapping in Applesoft.

AAL.8103.BXY
This issue contains articles on a pretty 'dump' command, 'unused' opcodes and what they do on a 6502, a complet 6502 opcode chart, moving commands to the language card, a _commented_ listing of the DOS 3.2.1 RWTS and an '&' command for the S-C Assembler II.

AAL.8104.BXY
This issue contains articles on text file I/O in assembly language, AppleSoft internal entry points, fast string input for Applesoft, hiding things in DOS, and the format code for both DOS 3.2.1 and DOS 3.3! PLUS a substring search for Applesoft and some S-C Assembler II patches.

AAL.8105.BXY
This issue contains articles on a hi-res SCRN function for Applesoft, conquering paddle jitter, a shift-key modification, the 6502 programming model and a commented listing of DOS 3.2.1 from $B800 through $BCFF.

AAL.8106.BXY
This issue contains articles on two fancy tone generators, more multiplication on the 6502, specialized multiplication, a commented listing of DOS 3.3 from $B800 through $BCFF and a review of 'Beneath Apple DOS' from when it was _new_.

AAL.8107.BXY
This issue contains articles on lower case in a II+, printing the screen, restoring clobbered page 3 pointers, corrections to the variable X-ref program in V1N2 (AAL.8011.BXY) and a step-trace utility!

AAL.8108.BXY
This issue contains articles on finding Applesoft line numbers, binary keyboard input, two ways to compare a byte, selective catalogs in FID, random number generation in Integer BASIC, corrections to V1N2 (AAL.8011.BXY) and a commented listing of the DOS 3.3 boot ROM!

AAL.8109.BXY
This issue contains articles on a field input routine for Applesoft, CHRGET and CHRGOT, exiting the S-C Assembler II, a new .AS directive for that assembler and a commented listing of DOS 3.3 RWTS (also used in ProDOS)!

AAL.8110.BXY
This issue contains articles on sifting primes faster and faster, a 6809 cross assembler, extending the Apple II monitor, some errata and a disassembly of DOS 3.3 from $B052-$B0B5 and $B35F-$B7FF.

AAL.8111.BXY
This issue contains articles on using AppleSoft from assembly language, a formatted print subroutine, a poor man's disassembler and a beginning lesson on loops.

AAL.8112.BXY
This issue contains articles on a 6809 card with FLEX, AppleSoft hi-res subroutines, hex constants in AppleSoft, an AppleSoft line editing aid, improved AppleSoft fast string input, adding ASCII dump to the original Apple II monitor and an AppleSoft GOTO from assembly language.

AAL.8201.BXY

This issue contains articles on a hi-res SCRN function with color, a 6502 relocator, a note of a problem in DOS 3.3, some handy EXEC files, a one-chip microcomputer, a couple of reviews and some S-C Assembler goodies.

AAL.8202.BXY
This issue contains articles on DOS error trapping from machine language, improving the EPSON controller card, even faster primes, a printer FIFO buffer, patches for Apple Writer to unhook PLE, a great free adventure and dividing by ten.

AAL.8203.BXY
This issue contains articles on reading 2 paddles at once, EPROM blasters, reviews, more about the EPSON interface, tricky code that always skips, using the AE Time II card, some corrections and a note from the publisher.

AAL.8204.BXY
This issue contains articles on adding auto-save to the S-C assembler, a review of an Applesoft editor, an easy shift-key modifier, using macros and nested macros and recursive macros, controlling software configuration and making a funny noise.

AAL.8205.BXY
This issue contains articles on a secret RWTS caller inside DOS 3.3, benchmarking block MOVEs, another recursive macro, reading a whole track with RWTS, reading the game buttons unambiguously and a macro branch library.

AAL.8206.BXY
This issue contains articles on implementing 'new' opcodes using BRK, a new hi-res function for Applesoft, a bubble sort, macro hints, a yes/no subroutine, a bell routine, a shift-key modification, searching for zero-page references, an automatic CATALOG for the S-C Macro Assembler and a memory examiner.

AAL.8207.BXY
This issue contains articles on run-anywhere subroutines, a giant macro for messages, sorting out zero-page references, simple hi-res animation, a text file display command for DOS and some reviews.

AAL.8208.BXY
This issue contains articles on search and perform subroutines, DOS free space patches, a quick way to write DOS on a disk, corrections to the July relocatable JSR command, efficient handling of very large assembly source files, a blinking underscore cursor and lots more goodies!

AAL.8209.BXY
This issue contains articles on new S-C products, a directory of assembler directives, relocatable ampersand-vector code, eliminating paddle interaction, some fast screen tricks, a bibliography, a note about the 6800 cross assembler and the underline cursor and some reviews and patches.

AAL.8210.BXY
This issue contains articles on a DOS 3.3 catalog arranger, why you need macros, converting toolkit source to S-C, S-C assembler goodies and info on how people could have written for AAL, plus a correction to the fast screen scrolling by Bob.

AAL.8211.BXY
This issue contains articles on sound patterns, digitized speech on an Apple

II, more fast primes, moving a symbol table, EXEC without END in Applesoft, an Applesoft program locator and REPEAT/UNTIL for Applesoft.

AAL.8212.BXY
This issue contains articles on making relocatable JMPs and JSRs, adding bit-control to the monitor, assembly listings on text files, commented Applesoft source, 65C02 preview, garbage collection in arrays, splitting strings to display length, several quickies and more S-C assembler goodies.

AAL.8301.BXY
This issue contains articles on a Super Scroller, branch opcode names, more on catalog arranger, adding decimal values from ASCII strings, programming the language card, seed thoughts on extensions, more quickies, ideas and reviews.

AAL.8302.BXY
This issue contains articles on really useful ASCII string adding, an endless alarm, Apple IIe notes (introduced just before this issue), an Applesoft INPUT tuner, star-tling stunts and quickies, S-C goodies and reviews.

AAL.8303.BXY
This issue contains articles on PTRGET and GETARYPT, a macro-building macro, Epson MX-80 screen dumps, a division tutorial, a note on prime benchmarks, garbage-collection indicator for Applesoft, more on the IIe and reviews.

AAL.8304.BXY
This issue contains articles on patching DOS 3.3 for fast LOAD and BLOAD, an 'ORG' macro, date processing modules, a new version of DOS 3.3, a general purpose patch installer, more reviews and a few notes.

AAL.8305.BXY
This issue contains articles on displaying character generator EPROMs, a reference of chips in the Apple II+, a PAUSE directive for S-C, some new cards, a program to find address references, generating parity and garbled error messages under DOS.

AAL.8306.BXY
This issue contains articles on a spiral screen clear, a burglary (for real), binary to decimal conversion, why not to replace INIT in DOS 3.3, reformatting a lot of text, working with track balls and an ampersand monitor caller.

AAL.8307.BXY
This issue contains articles on a 6502 mini-assembler in Applesoft, speeding up text file I/O, the 65C02, a revised monitor patch for ASCII display, an 80-column SHOW command, an explanation of the DOS 3.3 APPEND bug, S-C goodies and the resolution of the burglary.

AAL.8308.BXY
This issue contains articles on using auxiliary memory on the IIe, the 65C02, speeding up spirals, tinkering with variable cross references, reversing, getting and storing nibbles, some small patches and patch unification, and some 68000 boards for the Apple II.

AAL.8309.BXY
This issue contains articles on jump vectoring, generating machine code with Applesoft, Amper-monitor, more DOS 3.3 revisions, calculating base addresses, saving source files for Apple's mini-assembler, generic screen

dumps, a CATALOG interrupt and an 80-column ASCII Monitor dump.

AAL.8310.BXY
This issue contains articles on more tinkering with variable cross-references, faster booting for ScreenWriter II, large assembly listings to text files, lower case titles, a macro-calculated spiral screen clear, counting lines and more goodies.

AAL.8311.BXY
This issue contains articles with a commented listing of ProDOS 8's disk nibblization routines, a look at Aztec C, killing an EXEC file, shapemaker enhancements, ProDOS clock drivers and more on lower case titles.

AAL.8312.BXY
This issue contains articles with more disassemblies of ProDOS 8, more assembly listings into text files, more on Aztec C, generalized GOTO and GOSUB, finding trouble in a RAM card, the TimeMaster II from AE and converting S-C files to text files.

AAL.8401.BXY
This issue contains articles on a code profiler, more on a Don Lancaster assembly language book, DOS patches to avoid interrupt problems, more on the 65C02, some reviews, online with Steve Wozniak and a 68000 'color pattern'.

AAL.8402.BXY
This issue contains articles on listing buried messages, peeking at the catalog, fast scrolling on IIe 80-column screens, a look at the Macintosh, wrap-around addressing, delays, IIe soft switches, a text area erase routine, a macro to generate a quotient/remainder table for Hi-Res and even more good stuff!

AAL.8403.BXY
This issue contains articles on fast garbage collection, changing VERIFY to DISPLAY, faster table lookups via redundancy, disk drive pressure pads, ProDOS on a Franklin, the color pattern in 6502 code and a philosophical article wondering if ProDOS will succeed.

AAL.8404.BXY
This issue contains articles on a CRC subroutine, more clocks, an evening with Woz, quick DOS updating (no more MASTER CREATE), burning and erasing EPROMs, and macro source code available.

AAL.8405.BXY
This issue contains articles on random numbers for Applesoft, the Apple IIc, news from Roger Wagner, the enhanced Apple II ROM, the 65C02 in older Apple II machines, decimal floating point arithmetic, making a difference map and a solution to an old puzzle.

AAL.8406.BXY
This issue contains articles on 18-digit arithmetic (part 2), DOS studies, revisiting $48, more random number generators, booting ProDOS with a modified ROM, finding the bad bit using CRCs, and lots more too intricate to list here!

AAL.8407.BXY
This issue contains articles on 18-digit arithmetic (part 3), building label tables for DISASM, quick memory testing, a 68000 sieve benchmark, an updated 6502 prime sifter, sorting and swapping, 'gotchas' on the Apple IIc, orphans and widows, and speed vs. space.

AAL.8408.BXY
This issue contains articles on 18-digit arithmetic (part 4), enabling and disabling IRQ from Applesoft, line number cross references, slow chips, and a modification to DOS 3.3 for big BSAVEs.

AAL.8409.BXY
This issue contains articles on 18-digit arithmetic (part 5), faster ampersand routines to zero arrays, turning an index into a mask, putting messages on the screen, a bibliography on hi-res graphics and some great 'new' books.

AAL.8410.BXY
This issue contains amplifications on past articles on 18-digit arithmetic (plus part 6 of the series), more on 'index to mask', a review and sample program for the 65802, an index to volume 4 and reviews of two early Macintosh 68000 assemblers, of all things.

AAL.8411.BXY
This issue contains part 7 of 18-digit arithmetic (and square roots!), megabytes for the IIe, the 65816, an improved 80-column monitor dump, generating cross-reference files with DISASM, macro information by example, turning bit-masks into indexes and converting two-digit decimal strings to binary.

AAL.8412.BXY
This issue contains part 8 of 18-digit arithmetic, more details on 65C02's in older Apple II computers, corrections on V5N2's MVN/MVP, a strange way to divide by 7, sly hex conversion, remembering early computer prices, tables for faster hi-res, Blankenship's BASIC and a solution to overlapping DOS 3.3 patches.

AAL.8501.BXY
This issue contains part 9 of 18-digit arithmetic (the printing routine!), a symbol table source maker and a short single-byte hex-to-decimal printer. The first two routines are so informative they take up almost all of the 32-page paper issue!

AAL.8502.BXY
This issue contains part 10 of 18-digit arithemetic, questions and answers on the S-C 2.0 assembler, making DOS-less disks, corrections, reviews, more S-C assembler stuff and building hi-res pre-shift tables.

AAL.8503.BXY
This issue contains info on shortening the DOS file buffer builder, more on 65C02s in older Apple IIs, improved DOS 3.3 number parsing and lower-case DOS 3.3 commands, the Oki 6203 multiply/divide chip, a real 65816 diassembler (with source!) and finding memory size from the ProDOS 8 global page.

AAL.8504.BXY
This issue contains a volume catalog for Corvus and Sider hard disks, shrinking code inside BASIC.System, fast text windows for Applesoft, discussion of some 'new' products, reviews and S-C macro assembler stuff.

AAL.8505.BXY
This issue contains a new catalog for DOS 3.3, an 80-column window utility for the IIe and IIc, adding a DATE command to BASIC.System and lots of S-C Macro Assembler 2.0 modifications, plus some reviews and modifying the

Rak-Ware DISASM program, for all of us who still use it.

AAL.8506.BXY
This issue contains the Boyer-Morris string search algorithm, a short integer square-root subroutine, a note on the TXS instruction on the 65802, interrupt trace, improving the single-byte converter, two ROM sets in one IIe, a Call utility for Applesoft and some final DP18 subroutines.

AAL.8507.BXY
This issue contains info on how to read DOS 3.3 disks under ProDOS, how to recursively list files (including contents of subdirectories) on a ProDOS filesystem, and how to BSAVE to a new non-binary file under BASIC.SYSTEM 1.1. A review of the MCT SpeedDemon accelerator is also included.

AAL.8508.BXY
This issue contains how to make a 576K printer buffer on your IIc with a Z-RAM card, a discussion of how many bytes each opcode takes, some generic conversion routines and a wildcard file name search.

AAL.8509.BXY
This issue contains a prime benchmark for the 65802, putting DOS and ProDOS on the same disk, software sources for 65802 and 65816, problems putting 65802 chips in Apple II+ computers and a short binary-to-decimal conversion routine in 65802 (good for 65816 as well).

AAL.8510.BXY
This issue contains articles on a ProDOS driver that records what calls are made to it, a DOS 3.3 RWTS patch to do the same recording, a puzzle in a program that erases itself and more, more on putting 65C02 chips in older Apple II machines, a multiple-column disassembler, reviews, news and more.

AAL.8511.BXY
This issue contains articles on a 15K language card-based RAM disk for DOS 3.3, a patch to ProDOS QUIT to allow the right-arrow key, three solutiosn to the previous month's puzzle, a commented disassembly of the ProDOS QUIT call, and two ways to merge fields into one byte.

AAL.8512.BXY
This issue contains articles on bugs in last month's RAM disk driver, tracing the ProDOS MLI, a review of the OKS Kache Card, more puzzle solutions, pseudo-variables in machine language and computing the day of the week.

AAL.8601.BXY
This issue contains articles on converting lo-res pictures to hi-res, a question on returning from BRUN, text file transfer under DOS 3.3, fast 6502 and 65802 multiplication routines, a RAMWorks compatible auxmove routine, a correction to the dual DOS 3.3/ProDOS disk creator and trivia from Bill Mensch on the origin of the number '6502'.

AAL.8602.BXY
This issue contains articles on a wildcard-capable CATALOG for DOS 3.3, the Mitsubishi 50740 series microprocessors (MPW IIgs assembler actually recognizes these guys), a faster CRC method, corrections to faster garbage collection and a DOS 3.3 patch to prevent directly-entered commands from working.

AAL.8603.BXY
This issue contains articles on running ProDOS on non-Apple ROMs, even

faster 16X16 multiplication for the 65802 (or 65816), making a smarter 65816 disassembler, the fastest 6502 multiplication yet, PAL programming hardware, reviews, and a routine to determine which 65XXX series processor you're using!

AAL.8604.BXY
This issue contains articles on tool for restoring lost catalogs, using primitive text windows, dividing BCD values by four, booting into 80 columns, a faster boot for DOS 3.3 with more disk space and a screen hole gaffe in the second Apple IIc ROM release.

AAL.8605.BXY
This issue contains articles on modifying DOS 3.3 to use 3.5' disks, recovering lost programs in the S-C assembler environment and even more better division by seven.

AAL.8606.BXY
This issue contains articles on the 65816 stack relative addressing mode, fast 16X16 multiply and divide for the 65802, the real story about DOS and BRUN, toggling between two values, using SmartPort, generalized MLI error handling and a practical CRC use.

```
================================================================================
DOCUMENT !READ.ME.txt
================================================================================
```

```
       _____ ______
      /_____|/______\
     /__/|__|  ___|__|
    /__/_|__| /_____/
   /________|/__/      __ __ __
  /__/   |__|__/______ /_//_// /
 /__/    |__|________//  / \/_/
```

APPLE II PROGRAMMERS AND DEVELOPERS
ROUNDTABLE (A2Pro)

Part of GEnie, the General Electric
Network for Information Exchange

APPLE ASSEMBLY LINE (ANOTHER RELEASE OF THE LOST CLASSICS PROJECT OF THE APPLE II ROUNDTABLES ON GENIE): IMPORTANT INFORMATION

Welcome to A2Pro's release of Apple Assembly Line, the outstanding assembly-language programming newsletter written and published by Bob Sander-Cederlof from October 1980 through May 1988. These programming magazines are now available to all members of A2Pro on GEnie for only the cost of a download, including all source code disks and all articles!

If you wish to become a part of the Lost Classics project, visit the Lost Classics headquarters in the A2 RoundTable (p. 645) on the GEnie Information Service and check out the Lost Classics Bulletin Board Category (#7). This is a continuing effort, and we wish to embrace the entire Apple II community. Your assistance is greatly appreciated, and by helping Lost Classics you help all Apple II users everywhere!



Should you have any questions about the distribution restrictions, you may contact the A2Pro RoundTable (A2PRO.HELP) on GEnie for more detailed information.



NOTES ON FILES AND ORGANIZATION

Until July 1985, all Apple Assembly Lines source code and articles were created and delivered exclusively on DOS 3.3 disks. To help alleviate difficulty in retrieving the information, we have used the DOS 3.3 FST in GS/OS to transfer all files to ProDOS disks. We've also renamed the files accordingly so you can easily unpack, read and enjoy them.

Starting in July 1985, Apple Assembly Lines was delivered (to those subscribers who also purchased the disks) on "hybrid" DOS 3.3/ProDOS disks. These disks contain both ProDOS and DOS 3.3 catalogs. The ProDOS side usually included ProDOS versions of the source code and programs, and would occasionally include ProDOS-specific code or discussion.

Each issue in A2Pro's release of Apple Assembly Lines contains up to three folders:

```
  ARTICLES:  Text files with the articles as printed in AAL that month.
             Articles were written using Apple Writer and still have some
             Apple Writer formatting commands in the files.
  DOS3.3:    Source and object code files from the DOS 3.3 parts of disks,
             copied to ProDOS disks and archived
  PRODOS:    Source and object code files from the ProDOS parts of disks,
             when available.
```

Some of the information may be duplicated, but we prefer to bring it to you as it was mailed to subscribers.

ABOUT THE SOURCE CODE

Nearly all source code supplied is for the S-C Macro Assembler (also written by Bob Sander-Cederlof). The S-C Macro Assembler used a BASIC-like file format to store source code, including line numbers and simple compression of repeating characters. It "stole" the Integer BASIC file type (in both DOS 3.3 and ProDOS) to store its source files, making them not very useful to those without the S-C Macro Assembler.

To help the code look as it did in the magazine, we've converted all the files to ASCII text files, including their original line numbers, so you can follow the descriptions of the code in the articles. The conversion was done through a custom command for the Davex eight-bit command shell. The command ("sclist") is available separately in A2Pro's library.

We chose not to increase the archive sizes by including the original files as well as the text file versions. If you have need for any unmodified files from an original Apple Assembly Line disk, please let us know in the A2Pro bulletin board and we'll do what we can to make it available.

---

A2Pro and Lost Classics are pleased to bring this long-gone programming information back to Apple II programmers around the world. If you have any suggestions or comments, please come talk to us in the A2Pro bulletin board on GEnie (menu option #1 on page 530), or send GEnie mail to A2PRO.HELP (from internet, A2PRO.HELP@genie.geis.com).

Enjoy the Apple Assembly Line!

To sign up for GEnie, follow these simple steps:

1. Set your communications software to 8N1, half duplex (local echo), at 300, 1200 or 2400 baud.
2. Dial toll-free 1-800-638-8369, or in Canada, 1-800-387-8330. Upon connection, enter HHH.

3.    At the U#= prompt, enter XTX99020,A2PRO and then press <RETURN>.
4.    Have a major credit card ready.  In the U.S., you may also use
      your checking account number.

---------------------------------------------------------------------

For more information, call 1-800-638-9636, mail feedback@genie.geis.com,
or write:

GEnie, c/o GE Information Services, P.O. Box 6403, Rockville, MD 20850

==========================================================================
## DOCUMENT CATALOG
==========================================================================

### CATALOG

| Name | Type | Crtr | Size | Flags | Last-Mod-Date | Creation-Date |
|---|---|---|---|---|---|---|
| !READ.ME.txt | TEXT | R*ch | 97K | lvbspoImad | 11/3/99 2:41 AM | 1/5/78 12:05 PM |
| AAL-8010 | Fldr | Fldr | 776K | lvbspoIMAd | 9/18/00 5:51 PM | 9/18/00 5:49 PM |
| AAL-8011 | Fldr | Fldr | 873K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8012 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8101 | Fldr | Fldr | 970K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8102 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8103 | Fldr | Fldr | 1067K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8104 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8105 | Fldr | Fldr | 970K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8106 | Fldr | Fldr | 970K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8107 | Fldr | Fldr | 1067K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8108 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:53 PM | 9/18/00 5:49 PM |
| AAL-8109 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8110 | Fldr | Fldr | 1552K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8111 | Fldr | Fldr | 582K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8112 | Fldr | Fldr | 1940K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8201 | Fldr | Fldr | 1940K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8202 | Fldr | Fldr | 1843K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8203 | Fldr | Fldr | 1455K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8204 | Fldr | Fldr | 1358K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8205 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8206 | Fldr | Fldr | 2134K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8207 | Fldr | Fldr | 1649K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8208 | Fldr | Fldr | 2134K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8209 | Fldr | Fldr | 1843K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8210 | Fldr | Fldr | 873K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8211 | Fldr | Fldr | 2037K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8212 | Fldr | Fldr | 2037K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:49 PM |
| AAL-8301 | Fldr | Fldr | 2231K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8302 | Fldr | Fldr | 2910K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8303 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8304 | Fldr | Fldr | 1455K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8305 | Fldr | Fldr | 2037K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8307 | Fldr | Fldr | 2134K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8308 | Fldr | Fldr | 1649K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8309 | Fldr | Fldr | 2231K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8310 | Fldr | Fldr | 2910K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8311 | Fldr | Fldr | 1455K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8312 | Fldr | Fldr | 1358K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8401 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8402 | Fldr | Fldr | 2134K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8403 | Fldr | Fldr | 1843K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8404 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8405 | Fldr | Fldr | 1746K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8406 | Fldr | Fldr | 1940K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8407 | Fldr | Fldr | 1940K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8408 | Fldr | Fldr | 1067K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8409 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8410 | Fldr | Fldr | 1552K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8411 | Fldr | Fldr | 1940K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8412 | Fldr | Fldr | 2037K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:50 PM |
| AAL-8501 | Fldr | Fldr | 970K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:51 PM |
| AAL-8502 | Fldr | Fldr | 1261K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:51 PM |
| AAL-8503 | Fldr | Fldr | 1649K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:51 PM |
| AAL-8504 | Fldr | Fldr | 2037K | lvbspoIMAd | 9/18/00 5:54 PM | 9/18/00 5:51 PM |

```
AAL-8505            Fldr Fldr  1261K lvbspoIMAd    9/18/00  5:54 PM    9/18/00  5:51 PM
AAL-8506            Fldr Fldr  1940K lvbspoIMAd    9/18/00  5:54 PM    9/18/00  5:51 PM
AAL-8507            Fldr Fldr   873K lvbspoIMAd    9/18/00  5:54 PM    9/18/00  5:51 PM
AAL-8508            Fldr Fldr  1067K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8509            Fldr Fldr  1358K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8510            Fldr Fldr  1843K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8511            Fldr Fldr  1746K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8512            Fldr Fldr  1455K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8601            Fldr Fldr  2328K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8602            Fldr Fldr  1164K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8603            Fldr Fldr  2231K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8604            Fldr Fldr  1358K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8605            Fldr Fldr   970K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM
AAL-8606            Fldr Fldr  2716K lvbspoIMAd    9/18/00  5:55 PM    9/18/00  5:51 PM

:AAL-8010:
Articles            Fldr Fldr   582K lvbspoIMAd    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3              Fldr Fldr   194K lvbspoIMAd    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8010:Articles:
Add.Sub.One.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Gen.Msg.Printer.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
HW.Err.6502.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LC.for.SCAsm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Products.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8010:DOS3.3:
LowerCase.Adapt.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Msg.Printer.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8011:
Articles            Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3              Fldr Fldr   291K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8011:Articles:
BagsDisks4Sale.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sim.KeyPad.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Src.On.TxtFiles.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Use.For.USR.Cmd.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Variable.XRef.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8011:DOS3.3:
S.NumericKeyPad.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.TEXT.LIST.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Var.XRef.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8012:
Articles            Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3              Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8012:Articles:
BlockMoveCopy.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Compare.16Bits.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IBas.Prty.List.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Listed.Xprsns.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PrinterOnError.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Smart.Disasms.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8012:DOS3.3:
B.COPY.LINES.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MkCopyLinesFile.txt TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.COPY.LINES.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
S.IB.Ptry.Lstr.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PATCH.DA.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Setup.CopyLines.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8101:
Articles              Fldr Fldr   485K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8101:Articles:
Computed.Gosub.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Copy.for.SCAsm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Edit.Cmd.SCASM.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
How.Move.Mem.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8101:DOS3.3:
S.AmperGosub.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ASoft.BLTU.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.EDIT.COMMAND.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GENERAL.MOVE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.AmperGosub.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8102:
Articles              Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8102:Articles:
AppleNoiseSound.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AS.Str.Swapper.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.Misc.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
GRAM.Buy.Printr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
GRAM.Ftr.Laumer.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
GRAM.Hello.AS.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Multiply.6502.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8102:DOS3.3:
Demo.Str.Swap.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.APPLE.BELL.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.INCH.WORM.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LASER.BLAST.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LASER.SWOOP.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MACHINE.GUN.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MORSE.CODE.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MULTIPLY.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SIMPLE.TONE.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.STRING.SWAP.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.TOUCH.TONES.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8103:
Articles              Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8103:Articles:
A.Beaut.Dump.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Amper.Cmd.Int.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS321.RWTS.Lst.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Opcode.Chart.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Unused.Opcodes.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8103:DOS3.3:
AsmDisk4.0.Mod.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS321.BD00BE9F.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.AmperIntf.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BernardMemD.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Welman.Modifier.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8104:
Articles              Fldr Fldr  582K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM
DOS3.3                Fldr Fldr  679K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM

:AAL-8104:Articles:
AS.Substr.srch.txt    TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
DOS.Format.List.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Hiding.Undr.DOS.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Part.1.txt            TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Text.File.IO.txt      TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8104:DOS3.3:
Demo.Txt.Fl.Rd.txt    TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
DOS321BEAO.BFFF.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
DOS33.BEAF.BFFF.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
FastStr.Input.txt     TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Substr.search.txt     TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Test.Str.Input.txt    TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Test.Subst.Srch.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8105:
Articles              Fldr Fldr  582K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM
DOS3.3                Fldr Fldr  388K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM

:AAL-8105:Articles:
DontBeShiftless.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
DOS321.B800.Lst.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
GRAM.WPs.txt          TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Hires.Scrn.Func.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
No.Pdl.Jitter.txt     TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8105:DOS3.3:
DOS321.B800BCFF.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
HIRES.SCRN.TEST.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.HIRES.SCRN.txt      TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.PADDLE.JITTER.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8106:
Articles              Fldr Fldr  485K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM
DOS3.3                Fldr Fldr  485K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM

:AAL-8106:Articles:
DOS33.B800.List.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
FancyToneMakers.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Multiplication.txt    TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
Rvw.Beneath.DOS.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8106:DOS3.3:
DOS33.B800.BCFF.txt   TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.AMPERTONES.txt      TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.BASCALC.txt         TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.BY.TEN.txt          TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
S.MXN.MULTIPLY.txt    TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8107:
Articles              Fldr Fldr  582K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM
DOS3.3                Fldr Fldr  485K lvbspoimad   9/18/00  5:49 PM   9/18/00  5:49 PM

:AAL-8107:Articles:
Front.Page.txt        TEXT R*ch   97K lvbspoimad   11/3/99  2:41 AM    1/5/78 12:05 PM
```

```
LowerCaseApple.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Miscellaneous.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Screen.Printer.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
StepTrace.Util.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Var.XRef.Correx.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8107:DOS3.3:
S.F8EpromLC.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RESTORE.1.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RESTORE.2.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ScrnPrinter.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.STEP.TRACE.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8108:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8108:Articles:
Bin.Kbd.Input.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Compare.2Ways.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS33BootROMLst.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FID.Select.Cat.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FindASLineNums.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Miscellaneous.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rand.Nums.IntBA.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Re.AsmSrc.Text.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rvw.Apple.ML.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Whaduzzit.Do.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8108:DOS3.3:
DOS33.Boot.ROM.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hello.FW.Slot4.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.AMPERFIND.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Bin.Keyboard.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CallIB.Random.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RANDOM.TEST.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Rnd.Function.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8109:
Articles              Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8109:Articles:
CHRGET.CHRGOT.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS3.3.RWTS.Src.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Fancy.AS.Direct.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FieldInputRtn.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LeaveVers4.0.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8109:DOS3.3:
Demo.US.Direct.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CHRGET.PATCH.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CHRGET.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.D33.BD00BEAE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FldInputRtn.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.US.DIRECTIVE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Tst.Fld.Inp.Rtn.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8110:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8110:Articles:
```

```
DOS3.3Disasm.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Errata.CHRGET.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
GRAM.1lineprint.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Gram.Book.Revws.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
GRAM.Hello.AS.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sifting.Primes.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
XAsm.6809.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Xtnd.Apples.Mtr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8110:DOS3.3:
IB.Prime.Bench.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ASCII.Dump.P.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.D33.B35F.B7FF.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Mtr.Xtns.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Prm.B..Savoie.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Prm.Bnch.Fst.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Prm.Bnch.RBSC.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8111:
Articles              Fldr Fldr   388K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   194K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8111:Articles:
AS.ROMsFromAsm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Loops4Begs.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PoorMansDisasm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8111:DOS3.3:
PoorMans.Dsasm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FrmtPrint.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8112:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8112:Articles:
AS.GotoFromAsm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AS.HiRes.Subs.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AS.LineEditAID.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ASCII.Mon.Dump.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Excel.9.Review.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FstrStringInput.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hex.Const.AS.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Price.List.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8112:DOS3.3:
AS.DEMO.HI.RES.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ASoft.Inline.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Fast.Read.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GOTO.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HEX.CONSTANTS.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HI.RES.DEMO.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.INTEGER.INPUT.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Mossberg.LE.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PMD.Subr.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.FAST.READ.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.GotoFromML.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8201:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
```

```
:AAL-8201:Articles:
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
HandyExecFiles.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
HiresScrnColor.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
OneChip6500.1.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Relocator.6502.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Review.Index.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SCAsm.2.LC.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SeriousDOSPro.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
StepTraceCorrex.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8201:DOS3.3:
AS.Copy.FW.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AS.MAKE.LANGASM.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ASM.txt               TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
COPY.FIRMWARE.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
INT.txt               TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LOAD.ASM.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MAKE.LANGASM.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
READ.EXEC.FILE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HiresScrnClr.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RELOCATE.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WRITE.EXEC.FILE.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8202:
Articles              Fldr Fldr  1164K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8202:Articles:
BMA.VERSES.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS.Error.Trap.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
EvenFstrPrimes.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Great.Free.Adv.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ImprvEpsonCard.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
On.DivBy10.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Overseas.Subs.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Patch.AW.PLE.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PrinterFIFOBuf.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Problem.QD5.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8202:DOS3.3:
AW.Patch4PLE.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PutneyPrimeDrvr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DIVIDE.BY.TEN.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DOSOnErrXmpl.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.EpsonROMChng.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FIFOPrntHndlr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Putney.Primes.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8203:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   388K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8203:Articles:
Code.Alwys.Skip.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Correx.2.FIFO.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
EPROM.Blstr.Def.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.Epson.Intf.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.SCAsm.Ad.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
OtherEpsonMan.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rvw.6502.Subs.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rvw.AmperMagic.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Rvw.TimeII.Card.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SCAsm.Ready.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8203:DOS3.3:
Inst.DOS.Patch.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DATE.FILES.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DISPLAY.TIME.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PADDLES.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8204:
Articles              Fldr Fldr   776K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8204:Articles:
Add.AutoSave.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Ashby.Shift.Mod.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Pot.Tymac.Troub.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Recursive.Macro.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Review.AED.II.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sftwr.Cnfg.Ctrl.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Using.Macros.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8204:DOS3.3:
Inst.LA.Taylor.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Autosave.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FUNNY.NOISE.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LA.Ext.Taylor.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Recurs.Macro.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Schumer.Macro.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8205:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8205:Articles:
Anthr.Recur.Mac.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BlkMv.Benchmrk.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Branch.MacLIb.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Game.Buttons.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
NewAEDFeatures.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
NewOpcodes.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Printers.4Sale.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RWTS.Caller.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SCMacro.patches.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Secret.RWTS.Clr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8205:DOS3.3:
A.BlkMov.Bnch.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BlkMovBench.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BRANCH.MACROS.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GAME.BUTTON.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RecurMac.2.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.TRACK.READ.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WRTDIR.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8206:
Articles              Fldr Fldr  1164K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   970K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8206:Articles:
Auto.Catalog.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BRK.Opcodes.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BubbleSort.Demo.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
DFX.Review.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Examiner.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hint.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Bell.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Search.ZP.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shift.Key.Mod.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
XPlot4ASoft.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Yes.No.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8206:DOS3.3:
HXPLOT.DEMO.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.AUTO.CATALOG.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BubbleSrtDemo.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.EXAMINER.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HXPLOT.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Look4ZP.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MyOwnLtlBell.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.NewBrkOpcodes.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ReadKeyCase.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.YES.NO.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8207:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8207:Articles:
Animation.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Axlon.Review.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Flash.Ad.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Giant.Macro.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hierographic.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
OtherEpson.Man.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Relocatable.JSR.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Showfile.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sorted.ZeroPage.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Who.Are.We.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8207:DOS3.3:
Inst.Show.Cmd.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FILEDUMP.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GIANT.MACRO.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SHOW.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Smpl.Anim.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ZP.InOrder.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8208:
Articles              Fldr Fldr  1261K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8208:Articles:
AGAG.Review.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Auto.Man.Toggle.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Cursor.Routine.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Free.Space.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Large.Src.Files.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Macro.LC.Patch.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quick.DOS.Write.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QuickTrace.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Search.Perform.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shorts.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Videx.Patches.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
:AAL-8208:DOS3.3:
Do.Torens.Videx.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.AutoMan.Tgle.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Free.Sectors.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SearchPerform.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.UL.Cursor.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Videx.RtArrow.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Videx.Taylor.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Videx.Toren.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Toren.Dox.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8209:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8209:Articles:
Amper.Vector.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Directives.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hardcore.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Products.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Read.Paddles.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Screen.Tricks.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Underline.Fix.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
VidexPatchPatch.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
VidexRtArrow.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8209:DOS3.3:
S.CatalogArr.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PdlWOIntAct.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RelocAmperMac.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RelocAmpersnd.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Screen.Tricks.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Tookit.Conv.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Usr.Week.Fn.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.USR.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Toolkit.Conv.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8210:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8210:Articles:
Autocat.For.LC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CatalogArranger.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SC.LC.Patch.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Scroll.Correx.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SQ.Macro.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Toolkit.2.SC.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
USR.Week.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Writing.4.AAL.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8211:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr  1164K lvbspoimad    9/18/00  5:49 PM    9/18/00  5:49 PM

:AAL-8211:Articles:
Apple.Talker.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Changing.Lomem.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Exec.WO.End.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Locator.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.Speech.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Repeat.Until.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TonyFasterPrime.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8211:DOS3.3:
S.LOCATOR.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.NewAplTalker.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Repeat.Until.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.TonyFasterPrm.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SOUND.1.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SOUND.2.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SOUND.3.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SOUND.4.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SOUND.5.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Talk.A.Test.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TestRepeatUntil.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TONY.S.DRIVER.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8212:
Articles              Fldr Fldr  1455K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:49 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8212:Articles:
AS.Src.Code.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Bit.Control.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ClearStrngArray.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Enhanced.6502.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Enhancemnt.Rvw.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Es.Cape.Patch.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lancaster.Addtn.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ListOnTXTFile.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LoadRAMCard.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quickies.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RelocJMPsMeyer.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Split.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Toggle.Case.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8212:DOS3.3:
Meyers.Reloc.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BITS.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SPLIT.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.StrArrayClear.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.Split.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.StrArrClr.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8301:
Articles              Fldr Fldr  1940K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8301:Articles:
Amper.Review.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Arranger.Addtns.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Cookbook.Review.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CROSS.AD.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Filename.Editor.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hardcore.Mag.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Last.Minute.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Hardware.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QD9.COVER.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quickies.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RAM.Cards.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.C.DOCU.MENTOR.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Seed.Thought.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
String.Addition.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Super.Scroller.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
The.Book.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V3N4.6801.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Whats.Where.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8301:DOS3.3:
S.Fname.Editor.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.STRING.ADD.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SuperScroll.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8302:
Articles             Fldr Fldr  1455K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3               Fldr Fldr  1455K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8302:Articles:
Front.Page.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Gilder.Note.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIe.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MoreVidexPatchs.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Patch.TF.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Patch.TI.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PtchMacroHex.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quickie.6.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SC.WP.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Scooter.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Skinny.Page.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Stars.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
String.Adder.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Trapper.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8302:DOS3.3:
Divide.16.16.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ARRAYS.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Div.32.16.Trc.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Div.8.4.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Divide.32.16.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LinnsVidex.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MACRO.MACROS.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ScreenPrinter.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ScrnPrntrPlus.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SuperStrAddr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.TRAPPER.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.ARRAYS.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.Str.Adder.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.TRAPPER.2.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.TRAPPER.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8303:
Articles             Fldr Fldr  1746K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8303:Articles:
AAL.INDEX.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CROSS.AD.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Division.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Garbage.Indic.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIe.Stuff.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Macro.Macros.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Patch.4.68K.Asm.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PtrGet.GetAryPt.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QD10.COVER.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Screen.Printer.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Short.Item.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ShortPrimeNotes.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
T.MACRO.MACROS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Version1.1.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Version11Short.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
VisibleCPU.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8304:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8304:Articles:
Circut.Desc.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Disasm.Patches.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Fast.DOS.Patch.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mikes.Stuff.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ORG.Macro.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Patcher.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Prawm.Board.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V3N7.3.3E.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8304:DOS3.3:
Fast.Patch.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DATER.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FAST.LOAD.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ORG.MACRO.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PATCHER.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8305:
Articles              Fldr Fldr  1552K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8305:Articles:
AAL.CHART.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
APPLE.CHIPS.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Apple.Chips.Txt.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Cross.Ad.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Display.CharSet.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FADD.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mikes80ColCmts.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Cards.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ORDER.FORM.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Parity.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Pause.Direct.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDP11.XAsm.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rogram.2.Large.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SC.Capture.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8305:DOS3.3:
S.DispCharSet.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FADD.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PARITY.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PauseDirect.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SC.CAPTURE.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8307:
Articles              Fldr Fldr  1552K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8307:Articles:
```

```
Cross.Ad.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FastTextFileIO.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Feature.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mini.Assembler.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Miracle.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MonAsciiDisplay.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.DOS3.3.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
OBriens.BGE.BLT.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Opcodes.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Othello.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Short.Subjects.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Show.Poker.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V3N10.65C02.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WeishaarIIeDOS.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8307:DOS3.3:
MINI.ASSEMBLER.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FastTextRBSC.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FTSchlyter.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MAD.BOERING.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MAD.FIELD.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TxtFileSpeedup.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8308:
Articles               Fldr Fldr  1164K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                 Fldr Fldr   485K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8308:Articles:
Bit.and.Pieces.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FasterSpiral.PT.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIe.Auxmem.Bugs.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Kill.LIST.Cmd.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Macro.Patches.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.68K.Boards.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Pitz.VCR.Patch.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Reverse.Nybbles.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Wetzels.Patches.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Whisper.VolCtrl.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8308:DOS3.3:
S.NybbleGetPut.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PutneySpiral.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Wetzel11Patch.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WetzelLoader.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SJohnson.AUXMEM.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8309:
Articles               Fldr Fldr  1455K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                 Fldr Fldr   776K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8309:Articles:
Amper.Monitor.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AmperMon.Poker.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ASCII.80.Cols.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BaseAddr.Calc.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Break.Cat.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Churchs.Quickie.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Gen.Screen.Dump.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Jump.Vectoring.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
New.DOS33.Patch.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QuickTrace.Load.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RENEWAL.PLEA.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SAMPLE.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Spiral.Compiler.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8309:DOS3.3:
AmperMtr.Poker.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
JOHNSONS.MACROS.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.AMPER.MONITOR.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CatalogInt.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FastShortHBC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GenScreenDump.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Mon.ASC.DOBE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Spiral.Scr.Addr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8310:
Articles              Fldr Fldr  2425K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8310:Articles:
AAL.AUTHORS.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Adv.v1.v3.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Asm.From.400.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Avoid.Extra.Def.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Generic.Correx.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.AAAA.GGGG.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.HHHH.End.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.Page.nums.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Knouse.Mtr.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Large.Asm.Text.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LC.Titles.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Line.Counter.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Loves.Spiral.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.VCR.Tinker.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDos.Disasm.Xp.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Price.Changes.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rates.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Red.Faces.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ScreenWriter.II.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ShapeMaker.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Supress.Hex.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Where.To.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Writers.Guide.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8310:DOS3.3:
KnouseMtrPatch.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LINE.COUNTER.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LOVES.SPIRAL.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LoveSpiralFst.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.VCR.REVISED.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8311:
Articles              Fldr Fldr  1164K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8311:Articles:
Aztec.C.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Ideas....txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Killing.Exec.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lower.Case.Sq.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
PDOs.Clk.Drvr.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDos.Disasm.Ex.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Qwerty.Review.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shapemaker.Enh.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shorts.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
XAsm.6301.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8311:DOS3.3:
PDOS.F142.F1Be.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDos.F800.FFFF.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.KILL.EXEC.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8312:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   388K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8312:Articles:
Dataphile.Dgst.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LabelGOTO.Gosub.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ProDOS.Listing.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shafer.Asm.Text.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Short.Stuff.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
STB.128.Testing.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TimeMaster.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Trans.Src.Files.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8312:DOS3.3:
Conv.SC2Text.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Labelled.GOs.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Test.STB.128.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.Lbld.GOs.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8401:
Articles              Fldr Fldr  1261K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8401:Articles:
Bill.Mensch.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Interrupt.Patch.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lancaster.Books.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LocksmithReview.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Profiler.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEXT.TUTORIAL.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ThreeSuitPieces.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Understanding.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Urschels.Color.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V4N4.6502.NOTES.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Woz.Online.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8401:DOS3.3:
Ptch.DOS33.IRQ.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rods.Clr.Pat.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PROFILER.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Urschel.ClPat.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Urschel.table.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8402:
Articles              Fldr Fldr  1552K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8402:Articles:
```

```
Biblio.68000.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Creamers.Erase.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Delays.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FstScroll.IIe80.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mac.Thoughts.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Message.Search.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QR.Macros.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QuikLoader.Card.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Revisit.48.0.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Short.Subjects.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SoftswitchChart.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SWITCH.TABLES.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TimeMaster.II.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WrapAround.Addr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8402:DOS3.3:
DELAY.TIMES.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ERASE.DEMO.1.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ERASE.DEMO.2.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Erase.Creamer.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Msg.Search.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.ScrnTrIIe80.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8403:
Articles              Fldr Fldr  1164K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8403:Articles:
BragnerGPLEEtc..txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Customizing68K.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Felt.Pads.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Garbage.Collec.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lancaster.SCWP.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putney.ClrPat.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Redunancy.Table.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Shorts.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SILLY.SONGS.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
VerifyN2Display.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8403:DOS3.3:
GARBAGE.TEST.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PutneyTableMake.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QR.Table.Maker.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DISPLAY.FILE.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FastGarbage.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PutneysColor.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SATHER.3.16.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8404:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8404:Articles:
BurnErase.EPROM.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CRC.txt               TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Disasm.wExec.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Ideas....txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Intellec.Hex.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Source.Code.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Quick.DOS.Updtr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Woz.Talks.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8404:DOS3.3:
S.ApplyDOSPatch.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CRCHansKnecht.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Intellec.Hex.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8405:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8405:Articles:
Differences.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Part.1.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.IIe.ROMs.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Random.Numbers.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.IIc.65C02.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
That.Code.Did.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Wagner.News.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8405:DOS3.3:
ANOTHER.TEST.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lic.Plate.Game.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.Rnd.Tests.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DIFFERENCES.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.ADD.SUB.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RANDOM.KEYIN.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RANDOM.KNUTH.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.USRND.S.C.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.USRND.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8406:
Articles              Fldr Fldr  1358K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8406:Articles:
Andromeda.Board.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Barkovitch.Mntn.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CRC.Bad.Bit.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOSology.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Part.2.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LancastersStuff.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Making65C02Work.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.Rnd.Stuff.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Moto.Formatter.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDos.Mod.Mtr.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PRT.Command.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Revisit.48.0.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8406:DOS3.3:
S.CRCBadBidFndr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.MULTIPLY.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.Pack.Un.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.KANER.VOKEY.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MotoSType.Obj.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PRT.COMMAND.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8407:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
```

```
:AAL-8407:Articles:
DisasmNameTable.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Part.3.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIc.Notes.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Orphans.Widows.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quick.Mem.Test.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sieve.6502.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sieve.68000.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Speed.Vs.Space.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Swap.Sort.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8407:DOS3.3:
Faster.ShiftRt1.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LIST.PRIMES.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.DIVIDE.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.FIN.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.FstrMult.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SFPrimesImp.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SWAP.AND.SORT.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sieve.Eratos.1.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sieve.Eratos.2.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8408:
Articles              Fldr Fldr   776K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8408:Articles:
Big.BSAVEs.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.FOUT.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Enbl.Dsbl.IRQ.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LCR.Diagram.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LCR.txt               TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Slow.Chips.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8408:DOS3.3:
S.DP18.FOUT.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.PackUn.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LCR.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8409:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   388K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8409:Articles:
Clear.Arrays.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Dan.Pote.Ad.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Link.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Fast.Scrn.Msgs.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Graph.Biblio.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.2.Mask.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Reviews..txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8409:DOS3.3:
S.CLEAR.ARRAYS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18AmperLink.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.INDEX.MASK.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TWIRLERS.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
:AAL-8410:
Articles              Fldr Fldr  1261K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8410:Articles:
Arctec.Ad.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Correction.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Graphics.SW.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.2.Vol.4.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LCR.Correx.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mac.Assemblers.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Odd.Ways.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Out.Of.Print.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putneys.Way.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V5N1.65802.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8410:DOS3.3:
S.DP18.FUNC.1.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.GENERAL.MOVER.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PUTNEYS.WAY.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8411:
Articles              Fldr Fldr  1261K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM

:AAL-8411:Articles:
Alliance.CPUs.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Annc.2.0.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Disasm.Patches.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Func.2.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.New.SQRT.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Macro.Examples.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mask2Index.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Dump.Rtn.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
News.65816.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Quick.DecHex.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RAMWorks.MB.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8411:DOS3.3:
Opcodes.65816.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.FUNC.LOG.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Macro.Ex.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MASK.INDEX.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.New80ColMD.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.NewSQR.Rtn.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.QUICK.DEC.HEX.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8412:
Articles              Fldr Fldr  1358K lvbspoimad    9/18/00  5:50 PM    9/18/00  5:50 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8412:Articles:
BBasic.Review.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CorrectnMVNMVP.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Trig.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Funny.DivBy7.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Hex.To.Dec.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
HiresTableMaker.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIe.Auxmem.LC.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
IIPlus.65C02.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Little.Review.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Overlap.Patches.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RememberingWhen.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
XMas.CloseOuts.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8412:DOS3.3:
S.DP18.TRIG.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Funny.Divby15.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FunnyDivby3.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.FunnyDivby7.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HEX.TO.DEC.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MakeHiresAddr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Time.MVN.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8501:
Articles              Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8501:Articles:
DP18.Print.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Short.on.Mans.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ShortPrint255.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Sym.Sourceror.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
XASM.6800.2.0.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8501:DOS3.3:
S.DP18.Print.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.PRINT.000.255.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SymSourceror.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8502:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   291K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8502:Articles:
Book.review.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOSless.Disks.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Input.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Preshift.Tables.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Q.n.A.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Symbol.Pgm.Crx.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WriteGuard.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
YostsFreeOffer.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8502:DOS3.3:
S.Bld.PreShft.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DOSLESS.INIT.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.DP18.INPUT.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8503:
Articles              Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8503:Articles:
BAP.Correction.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Disasm.65816.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS.Buffer.Bldr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS.Numin.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
OKI.6203.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Sather.on.65C02.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8503:DOS3.3:
PatchDOS4LC.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.65816.DISASM.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.DOS.NUMIN.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.DOSLCPatch.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.DOSNuminRBSC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.INIT.BUFFERS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.InitBuf802.XY.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.InitBufs.802.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.InitBufs.SC.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8504:
Articles              Fldr Fldr  1358K lvbspoimad    9/18/00  5:51 PM   9/18/00  5:51 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM   9/18/00  5:51 PM

:AAL-8504:Articles:
AD.8086.XASM.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Cross.8086.8088.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Fast.Windows.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Hard.Cat.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Inside.IIc.Book.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
ListMajorLabels.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
LovesConversion.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Micro.Magic.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
ProDOS.numout.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Q.n.A.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
QuikLoader.Euge.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Review.Sider.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8504:DOS3.3:
Asm2.0FastBLOAD.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.Hard.Cat.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.List.Mjr.Lbl.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.PD.NUMOUT.SC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.ProDOS.NUMOUT.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.WINDOWS.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
WINDOW.DEMO.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8505:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:51 PM   9/18/00  5:51 PM
DOS3.3                Fldr Fldr   388K lvbspoimad    9/18/00  5:51 PM   9/18/00  5:51 PM

:AAL-8505:Articles:
Auto.Manual.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Disasm.TechNote.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Front.page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Littles.ProDOS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
New.Catalog.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Probs32BitValue.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
ProDOS.Date.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
Windows80Column.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM

:AAL-8505:DOS3.3:
S.AUTO.MAN.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.DATE.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.NEW.CATALOG.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
S.WINDOWS.80.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM    1/5/78 12:05 PM
```

```
:AAL-8506:
Articles             Fldr Fldr  1164K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3               Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8506:Articles:
Ads.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Alliance.Note.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
AppleVisions.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BernardsHexSrch.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.Leftovers.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Firmware.27128.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Johnsons.Call.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Note.65802.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putney.IRQTrace.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SQRT16.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8506:DOS3.3:
DIGITS.3.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DP18.MOVE.SUBS.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CALL.UTIL.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.HEX.SEARCH.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.IRQ.TRAPPER.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LovesConvers.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SQRT16.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.SQRT16.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8507:
Articles             Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS               Fldr Fldr   194K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8507:Articles:
BSave2NewFile.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
New.Cat.Revisit.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ProDOS.DOS.Load.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Recursive.Cat.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SpeedDemon.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8507:ProDOS:
S.DOS.LOAD.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RECURCAT.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8508:
Articles             Fldr Fldr   582K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3               Fldr Fldr   194K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS               Fldr Fldr   291K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8508:Articles:
Conversions.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Davids.IIc.Buff.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
How.Many.Bytes.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WildcardMatcher.txt  TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8508:DOS3.3:
S.Byte.Table.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WILDCARD.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8508:ProDOS:
BUF.320K.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BUF.576K.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
BUF.64K.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8509:
Articles              Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8509:Articles:
Convert.65802.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS.PDos.Init.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PrimeSieve65802.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Problems.65802.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RainbowProgInfo.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Software.65802.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8509:DOS3.3:
PrintPrimeTable.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.65802.Convers.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BINDEC.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Init.Dos.PDos.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SF802PrmPlus.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.SFast802Prm.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8510:
Articles              Fldr Fldr  1552K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   194K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr    97K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8510:Articles:
Another65C02Fix.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Apple.Manuals.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ErvEdgeExecFile.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ErvEdgeWildcat.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ErvEdgeWildcatx.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Gilder.Review.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Index.2.Vol.5.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
JohnLoveArticle.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mcinerney.Sieve.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PolyCol.Disasm.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Puzzle.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
QD20.CoverSheet.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Snooper.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Snoopers.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8510:DOS3.3:
S.POLYCOL.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RWTS.SNOOPER.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8510:ProDOS:
PRODOS.SNOOPER.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8511:
Articles              Fldr Fldr  1067K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   582K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr    97K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8511:Articles:
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Kablit.txt            TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Merging.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Object.Vector.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
PDos.Quit.Code.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ProDOS.Quit.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Puzzle.Solves.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RAMDisk.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
SathersComments.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Words.On.MacAsm.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8511:DOS3.3:
DJohnsonsFiller.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
LittleRamDisk.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MergeFieldByte.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RAMFill.Adam.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RAMFILL.RBSC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WROMWRITE.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8511:ProDOS:
S.PRODOS.QUIT.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8512:
Articles              Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8512:Articles:
Day.Of.Week.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Kashmarek.Trace.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
More.Pzl.Solves.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PQRS.txt              TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PseudoVariables.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RAMDisk.Bug.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8512:DOS3.3:
S.DAY.OF.WEEK.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RAMFIll.BLove.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.RAMFILLPutney.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.READ.TIME.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.READTIMEPLUS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.DayWeek.1.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Test.DayWeek.2.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8601:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr   582K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8601:Articles:
Browns.Mover.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Correx.DblInit.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lawries.Notes.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Lores2Hires.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Monthly.Disks.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Multiplying.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
My.Ad.txt             TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Parker.Trivia.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Potts.TxtCopy.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8601:DOS3.3:
BrownMoveProg.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
POTTS.A               Fldr Fldr    97K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
PottsTextCopier.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Lores2Hires.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.M1616.802.EF.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Mult.16.16.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
S.MULTIPLY.8X8.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TextTransferObj.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8601:DOS3.3:POTTS.A:
S.TRANSFER.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8601:ProDOS:
BROWNS.MOVE.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
POTTSTEXTCOPIER.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.LORESTOHIRES.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MUL16X1665802.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MULTIPLY16X16.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MULTIPLY8X8.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8602:
Articles              Fldr Fldr   582K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr    97K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8602:Articles:
ErvEdge.Wildcat.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Faster.CRCs.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Garbage.Correx.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mitsubishi.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RichardDOSPatch.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8602:DOS3.3:
Gendron.DOS.Mod.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.CRC.GENERATOR.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WILDCAT.EXEC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WILDCAT.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
WILDCAT.EXEC.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8602:ProDOS:
S.CRC.GENERATOR.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8603:
Articles              Fldr Fldr   970K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr   776K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8603:Articles:
Boughner.Mult.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Disasm65816Plus.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PAL.Programmer.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
PDos.Franklines.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putney.Mul8x8.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Transwarp.Rvw.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
V6N6.IIX.Rumors.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Weishaars.Book.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Which.Processor.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8603:DOS3.3:
Boughner.Mult.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Creat.SqTbl.Src.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putney.Fst.8x8.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Putney.Fstr.8x8.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Which.CPU.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8603:ProDOS:
BOUGHNERS.MULT.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CHECKSUMMER.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CREATE.SQUARE.T.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
PUTNEYS.8X8.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ROBISONS.8X8.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.816.DSM.NEW.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.WHICH.PROC.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
TEST.CKSUMMER.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8604:
Articles              Fldr Fldr   679K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr   194K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8604:Articles:
BCD.Magic.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Boot.80.txt           TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
IIc.ROM.Bug.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Msg.Into.Window.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
NewDOSInit.Boot.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rest.Clob.Cata.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8604:DOS3.3:
BCD.MAGIC.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DOS33.B700.B7FF.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.BigCatDisp.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Find.TS.Lists.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Msg.Into.Wind.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8604:ProDOS:
BCD.MAGIC.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.MSG.INTO.WNDW.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8605:
Articles              Fldr Fldr   388K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   485K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr    97K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8605:Articles:
Bartletts.Searc.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Division.By7.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
UniDisk.RWTS.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8605:DOS3.3:
BETTER.DIV.7.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
FIND.START.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
RWTS.3.5.txt          TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Format.UDsk.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.UNIDISK.RWTS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8605:ProDOS:
BETTER.DIV.7.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8606:
Articles              Fldr Fldr   873K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
DOS3.3                Fldr Fldr   873K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM
ProDOS                Fldr Fldr   970K lvbspoimad    9/18/00  5:51 PM    9/18/00  5:51 PM

:AAL-8606:Articles:
Butterill.Ops.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Call.Sequences.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
CorrexAbtBruns.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Front.Page.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MLI.Error.Hndlr.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Protocol.Conv.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Rindsbergs.CRC.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

```
Stack.Relative.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Toggling.Values.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8606:DOS3.3:
Bell.Demo.Src.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Butterill.Demo.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Butterill.Div.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Butterill.Mult.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Div16.Demo.Src.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
Mult16.Demo.Src.txt   TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ROM.CRC.Calc.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Test6502Call.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.Test816Call.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM

:AAL-8606:ProDOS:
BUTTERILL.DEMO.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BUTTERILLS.DIV.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
BUTTERILLS.MUL.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
DIV16.DEMO.txt        TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MLI.ERROR.PLUS.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MLI.ERROR.txt         TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
MULT16.DEMO.txt       TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
ROM.CRC.CALC.txt      TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.02.CALL.SEQ.txt     TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
S.816.CALL.SEQ.txt    TEXT R*ch    97K lvbspoimad    11/3/99  2:41 AM     1/5/78 12:05 PM
```

================================================================
DOCUMENT :AAL-8010:Articles:Add.Sub.One.txt
================================================================

## How to Add and Subtract One

I suppose there are as many ways to do it as there are programmers. Some are short and fast, some long and slow, some neat, some sloppy.

Adding one to a number is called "incrementing", and subtracting one is called "decrementing". The 6502 has two instructions for these two functions: INC and DEC. (For the moment I will overlook the four instructions for doing the same to the X and Y registers: INX, INY, DEX, and DEY.) It is easy to see how to use them on single-byte values; with a little more trouble we can also use them for values of two or more bytes.

Single-Byte Values:

Here are five different ways to increment a single byte:

```
Methods 1 and 2:  Add 1
     CLC                   SEC
     LDA VALUE             LDA VALUE
     ADC #1                ADC #0
     STA VALUE             STA VALUE

Method 3 and 4:  Subtract (-1)
     SEC                   CLC
     LDA VALUE             LDA VALUE
     SBC #$FF              SBC #$FE
     STA VALUE             STA VALUE

Method 5:  Use the INC instruction
     INC VALUE
```

Here are five similar ways to decrement a value:

```
Method 1 and 2:  Subtract 1
     SEC                   CLC
     LDA VALUE             LDA VALUE
     SBC #1                SBC #0
     STA VALUE             STA VALUE

Method 3 and 4:  Add (-1)
     CLC                   SEC
     LDA VALUE             LDA VALUE
     ADC #$FF              ADC #$FE
     STA VALUE             STA VALUE

Method 5:  Use the DEC instruction
     DEC VALUE
```

There are times when any of the above may be justified, depending on the state of the A-register and the Carry Status bit.

Multi-Byte Values:

Incrementing a two-byte value is a very common practice in 6502 programs.  Here are two methods:

Method 1:  Add 1
```
     CLC
     LDA VALL   LOW BYTE
     ADC #1
     STA VALL
     LDA VALH   HIGH BYTE
     ADC #0
     STA VALH
```

Method 2:  Use the INC instruction
```
     INC VALL   INCREMENT LOW BYTE
     BNE .1     IF NOT ZERO, THEN NO CARRY
     INC VALH   INCREMENT HIGH BYTE
.1   .....
```

Of course, there are many variations on these methods.  It is easy to see how to extend these two methods to more than two bytes.  Here is a three-byte version of Method 2:
```
     INC VALL   INCREMENT LOW BYTE
     BNE .1     UNLESS ZERO, NO CARRY
     INC VALM   INCREMENT MIDDLE BYTE
     BNE .1     UNLESS ZERO, NO FURTHER CARRY
     INC VALH   INCREMENT HIGH BYTE
.1   ....
```

Believe it or not, there is one disadvantage to using Method 2, in some circumstances.  Sometimes code is required to have a constant running time; then, Method 1 is the one to use.  But most of the time, Method 2 is the best.

How about subtracting one?  Here are two ways to do it to a two-byte value:

Method 1:  Subtract 1
```
     SEC
     LDA VALL
     SBC #1
     STA VALL
     LDA VALH
     SBC #0
     STA VALH
```

Method 2:  Use the DEC instruction
```
     LDA VALL   SEE IF NEED TO BORROW
     BNE .1     NO
```

```
     DEC VALH   YES
.1   DEC VALL
```

Which one do you like better? It is still a matter of taste, unless the amount of memory used or time consumed is very important. There are also different side effects, such as the final state of the carry status. INC and DEC do not change the carry status, while of course ADC and SBC do. You may wish to preserve carry through the process, making the INC/DEC code preferable. Or, you may wish to know the resulting carry status after incrementing or decrementing for some reasong; then you should use the ADC/SBC code.

Back to subtracting one...how about doing it to a three-byte value? We just add three more lines:

```
     LDA VALL   SEE IF NEED TO BORROW
     BNE .2     NO
     LDA VALM   SEE IF NEED TO BORROW AGAIN
     BNE .1     NO
     DEC VALH   BORROW FROM HIGH BYTE
.1   DEC VALM   BORROW FROM MIDDLE BYTE
.2   DEC VALL
```

Easier than you though, right? You would not believe the many strange ways I have seen this operation coded in commercial software (even some released by Apple themselves!). Yet it seems to me that this method is the same way we would do it with pencil and paper in decimal arithmetic. Think how you would do this:

```
               123040
                   -1
               ------
               xxxxxx
```

If you think of each digit as though it were a byte...isn't the algorithm the same?

Now it is time for all of us to go back over the programs we wrote during the past three years for the Apple, and replace a lot of old code!

```
==============================================================================
DOCUMENT :AAL-8010:Articles:Front.Page.txt
==============================================================================
```

Volume 1 -- Issue 1           October, 1980

Welcome to the premier issue of the Apple Assembly Line!

This new monthly newsletter is dedicated to the many Apple owners using assembly language, or who would like to learn how.  Articles will include commented disassemblies of Apple ROM routines, DOS, and other commercial software; how to augment and modify existing products; beginner's lessons in assembly language; handy subroutines every programmer needs in his tool kit; and many more.

In this issue you will find a tutorial on efficient ways to increment and decrement multiple-byte values, a very powerful subroutine for formatting messages on the screen, and patch code for the S-C ASSEMBLER II Version 4.0 to "adapt" it to the Paymar Lower-Case Adapter.  There is also an article describing a recently reported error found in ALL 6502 chips, and a brief announcement of some new products from S-C SOFTWARE.

Since there will be a lot of source code printed in this and forthcoming issues of the Apple Assembly Line, I plan to offer quarterly diskettes containing all published source code (in the format of the S-C ASSEMBLER II Version 4.0) at a nominal price.  How does $15 per quarter sournd?  Of course, you can always type it in.... The articles should be considered copy-righted, but feel free to use the code in any way you can.  It is printed here for your enlightenment, entertainment, and for your USE.  I hope you find it all helpful.

I do not know all there is to know about the 6502, or the Apple, or about anything!  Nor do I have an infinite amount of time.  Therefore, I will be happy to accept articles and programs from you.  I may print them exactly as you write them, or I may modify them first.  In any case, you will get credit, and the satisfaction of knowing you are helping many others in their conquest of the computer.

If you know others who should be receiving this newsletter, spread the word!  If you are not subscribing yet, then send your $12 today!  If you have any comments about the content, format, or whatever, write now!  Or, you can call me during reasonable at (214) 324-2050.

Sincerely,

<<signature>>

Bob Sander-Cederlof

==============================================================================
DOCUMENT :AAL-8010:Articles:Gen.Msg.Printer.txt
==============================================================================

General Message Printing Subroutine

Formatting a series of nice messages or screens-full of messages is hard enough to do in Applesoft...but in assembly language it can really be a difficult job. And it seems to take so much memory to do the equivalent of VTAB, HTAB, HOME, and PRINT. I was recently motivated to do something about this for a large, verbose program. I designed a general subroutine for printing text, which can print all 128 chracters of ASCII, plus do some fancy footwork on the way.

Embedded control codes in the text to be printed perform such handy functions as HTAB, VTAB, HOME, NORMAL, INVERSE, Clear to End of LIne, Clear to End of Page, Two-Second Delay, and Repeat. All characters to be printed directly are entered with the high-order bit set to one; bytes with the high order bit zero are control codes. Comments in lines 1250-1350 of the listing show what the codes are.

To simplify the calling sequence, a table of message addresses is built along with the messages themselves. To print a specific message, merely load the message index number into the A-register (LDA #0 for the first message, LDA #1 for the second, etc.), and JSR MESSAGE.PRINTER. Some sample messages are given in the listing, starting at line 2240.

There are a lot of unused control codes, which you can use to augment the subroutine. I am planning to add a code to switch to a HI-RES TEXT driver, for writing text on either of the two Hi-Res screens. You can probably think of a lot of useful ones yourself. The point is that this type of subroutine can simplify programming of an interactive program, and save memory too.

```
==============================================================================
DOCUMENT :AAL-8010:Articles:HW.Err.6502.txt
==============================================================================
```

Hardware Error in ALL 6502 Chips!

INTERFACE, the newsletter of Rockwell International (P. O. Box 3669, RC 55, Anaheim, CA 92803), Issue No. 2, is the source for the following information. It should be noted by all Apple owners working in assembly language, because it could cause an almost unfindable bug!

There is an error in the JUMP INDIRECT instruction of ALL 6500 family CPU chips, no matter where they were made. This means the error is present in ALL APPLES. This fatal error occurs only when the low byte of the indirect pointer location happens to be $FF, as in JMP ($08FF). Normally, the processor should fetch the low-order address byte from location $08FF, increment the program counter to $0900, and then fecth the high-order address byte from $0900. Instead, the high-order byte of the program counter never gets incremented! The high-order address byte gets loaded from $0800 instead of $0900! For this reason, your program should NEVER include an instruction of the type JMP ($xxFF).

Try this example to satisfy yourself that you understand the problem: insert the following data from the monitor.

```
       *800:09
       *810:6C FF 08   (this is JMP ($08FF)
       *8FF:50 0A      (pointer
       *A50:00         (BRK instruction we SHOULD reach)
       *950:00         (BRK instruction we DO reach!)
```

Execute the instruction at $0810 by typing 810G. If the JMP indirect worked correctly, it would branch to location $0A50 and execute the BRK instruction there. However, since the JMP indirect instruction has this serious flaw, it will actually branch to the BRK instruction at $0950!

Since it is very difficult to predict the final address of all pointers in a large assembly language program, unless they are all grouped in a block at the beginning of the program, I suggest that you take special measures to protect yourself against this hardware problem. (One measure, of course, was suggested in that sentence.) My favorite method is to avoid using the JMP indirect instruction. It takes too long to set it up in most cases anyway. I prefer to push the branch address (less one) onto the stack, and RTS to effect the branch. This allows me to create the effect of an indexed JMP. For example, suppose a command character is being decoded. I process it into a value in the A-register between 0 and N-1 (for N commands), and do the following:

```
       ASL             Double to create index
       TAX                     for address table
       LDA JUMP.TABLE+1,X      High order byte
```

```
       PHA                     of branch address
       LDA JUMP.TABLE,X        Low order byte
       PHA                     of branch address
       RTS
```

The jump table looks like this:

```
       JUMP.TABLE
               .DA COMMANDA-1  The "-1" is
               .DA COMMANDB-1  on each line
               .DA COMMANDC-1  because the RTS
               .DA COMMANDD-1  adds one before
               et cetera       branching.
```

This trick was described by Steve Wozniak in an article in BYTE magazine back in 1977 or 1978. It is also used by him in the Apple monitor code, and in SWEET-16. In both of these cases, he has arranged all the command processors to be in the same page, so that the high order byte of the address can be loaded into the A-register with a load-A-immediate, and the jump table can be only one-byte-per-command. See your Apple ROMs at locations $FFBE-FFCB (jump table at $FFE3-FFF9) and in SWEET-16 at $F69E, F6A0, F684-F6B8 (jump table at $F6E3-F702).

You can extend this idea of an indexed JMP instruction into a simulated indexed JSR instruction. All you have to do is first push onto the stack the return address (less one), and then the branch address (less one). I use this trick in the Message.Printer program described elsewhere in this issue.

```
==============================================================================
DOCUMENT :AAL-8010:Articles:LC.for.SCAsm.txt
==============================================================================
```

Using the Paymar Lower-Case Adapter            Bob Matzinger
with S-C Assembler II Version 4.0               817-275-2910

Since purchasing the Paymar adapter, I have spent a lot of time adapting software to effectively use it!  The program geven here will adapt the version 4.0 of Bob Sander-Cederlof's assembler to allow lower-case comments.

The two patches at lines 1340 and 1390 have to be entered, and the body of the patch loaded at $300.  Once installed, typing a control-A will toggle the shift-lock; control-S will perform a single-character upper-case shift; control-K, -L, and -O give access to the characters normally missing from the Appple keyboard.

Only comments can be entered in lower-case.  Further modification to the assembler would be required to allow commands, labels, and opcodes to be entered in lowr- or mixed-case.

==============================================================================
DOCUMENT :AAL-8010:Articles:New.Products.txt
==============================================================================

New Products from S-C SOFTWARE

As many of you know, because you have already bought it, version 4.0 of the S-C Assembler II is now on the market. With this new version, the price has gone up from $35 to $55. An upgrade kit for owners of previous versions is only $22.50

Now another new version is available, for those of you without disks! Tape Version 4.0 requires only 16K RAM and a cassette drive. The price is $45 for the complete package, or $22.50 for an upgrade kit from the previous tape version. All of the new features of Disk Version 3.2 and 4.0 are included, except those which require a disk drive. For the time being, the manual consists of a copy of the disk version 4.0 manuals, with a single sheet describing the differences in the tape version. Purchasers of tape version 4.0 will be able to upgrade to the disk version when they get a disk drive, for only $12.50.

And still another version of the assembler! This one is a cross assembler for the Motorola 6800, 6801, and 6802 microprocessors. It has all the features of the S-C Assembler II Disk Version 4.0, but the source language accepted is that of the 6800 family rather than the 6502. The price for this package is only $300, which is less than a month of time-sharing services for an equivalent capability would cost! An Apple, a ROM blower from Mountain Hardware, and the S-C Assembler II-6800 are all you need for a full-blown development system.

```
===================================================================
DOCUMENT :AAL-8010:DOS3.3:LowerCase.Adapt.txt
===================================================================

 1000  *---------------------------------
 1010  * Lower case conversion for
 1020  * S-C ASSEMBLER II Version 4.0
 1030  * Copyright 1980 by S-C SOFTWARE
 1040  * Complete with 126 ASCII characters
 1050  *---------------------------------
 1060  * The CTRL-A and CTRL-S keys are used similar to
 1070  * shift and lock keys on a standard typewriter.
 1080  *
 1090  * CTRL-A is the shift-lock key.
 1100  * Each time CTRL-A is pressed the case
 1110  * will toggle to the opposite mode.
 1120  *
 1130  * CTRL-S makes the following character
 1140  * enter in upper-case.
 1150  *---------------------------------
 1160  * REMEMBER!
 1170  * All commands and mnemonic entries
 1180  * must be in UPPER case!
 1190  * Use lower case only for comments!
 1200  *---------------------------------
 1210  CTRLA  .EQ $81      SHIFT LOCK
 1220  CTRLK  .EQ $8B      [ or {
 1230  CTRLL  .EQ $8C      \ or |
 1240  CTRLO  .EQ $8F      _ or rubout
 1250  CTRLS  .EQ $93      SHIFT
 1260  *---------------------------------
 1270  * Remember:
 1280  * shift M yields ] or }
 1290  * shift N yields ^ or ~
 1300  * shift P yields @ or `
 1310  RDKEY  .EQ $FD0C
 1320  *---------------------------------
 1330         .OR $1380
 1340         .TF LC.PATCH1
 1350         JSR LC
 1360  *---------------------------------
 1370         .OR $139A
 1380         .TF LC.PATCH2
 1390         AND #$FF
 1400  *---------------------------------
 1410         .OR $300
 1420  * CAUTION: Do not assemble your programs into
 1430  * $0300 up.  You will destroy this routine!!!
 1440  LC     JSR RDKEY
 1450         CMP #CTRLA
 1460         BEQ LOCK
 1470         CMP #CTRLS
 1480         BNE CHECK
```

```
1490  SHIFT  LDA #0
1500         STA LCKFLG
1510  SHIFT1 LDA #0
1520         STA CASE
1530         BEQ LC        ...ALWAYS
1540  LOCK   LDA LCKFLG
1550         EOR #1
1560         STA LCKFLG
1570         BNE SHIFT1
1580         LDA #$20
1590         STA CASE
1600         BNE LC        ...ALWAYS
1610  CHECK  CMP #CTRLK
1620         BEQ SPEC
1630         CMP #CTRLL
1640         BEQ SPEC
1650         CMP #CTRLO
1660         BNE CONV
1670  SPEC   ORA #$50
1680  CONV   CMP #$C0
1690         BCC RETURN
1700         ORA CASE
1710  RETURN PHA
1720         LDA LCKFLG
1730         BNE OUT
1740         LDA #$20
1750         STA CASE
1760  OUT    PLA
1770         RTS
1780  LCKFLG .DA #0
1790  CASE   .DA #$20
1800  *--------------------------------
1810  * Written by Bob Matzinger
1820  * September 6, 1980
1830  *--------------------------------
```

```
=======================================================================
DOCUMENT :AAL-8010:DOS3.3:S.Msg.Printer.txt
=======================================================================

 1000  *--------------------------------
 1010  MON.CH     .EQ $24
 1020  MON.CV     .EQ $25
 1030  MON.VTAB   .EQ $FC22
 1040  MON.CLREOP .EQ $FC42
 1050  MON.HOME   .EQ $FC58
 1060  MON.CLREOL .EQ $FC9C
 1070  MON.WAIT   .EQ $FCA8
 1080  MON.COUT   .EQ $FDED
 1090  MON.NORMAL .EQ $FE84
 1100  MON.INVERSE .EQ $FE80
 1110  *--------------------------------
 1120  MSG.PNTR   .EQ $18,19
 1130  MSG.SCANNER .EQ $1A
 1140  *--------------------------------
 1150  *      MESSAGE PRINTER
 1160  *
 1170  *   CALL:
 1180  *      (A) = MESSAGE #   (0-N)
 1190  *      JSR MESSAGE.PRINTER
 1200  *
 1210  *   ACTION:
 1220  *      1.  FINDS SPECIFIED MESSAGE
 1230  *      2.  PRINTS ON THE SCREEN
 1240  *      3.  INTERPRETS CHARACTERS AS FOLLOWS:
 1250  *          $00     END OF MESSAGE
 1260  *          $01-28  HTAB 1-40
 1270  *          $40-57  VTAB 1-24
 1280  *          $60     CLEAR SCREEN, HOME CURSOR
 1290  *          $61XXYY REPEAT CHARACTER YY, XX TIMES
 1300  *          $62     DELAY ABOUT TWO SECONDS
 1310  *          $63     NORMAL MODE
 1320  *          $64     INVERSE MODE
 1330  *          $65     CLEAR TO END OF LINE
 1340  *          $66     CLEAR TO END OF SCREEN
 1350  *          $80-FF  PRINT AS IS
 1360  *
 1370  *--------------------------------
 1380  MESSAGE.PRINTER
 1390         ASL          DOUBLE MSG NUMBER TO GET INDEX
 1400         TAY
 1410         LDA MESSAGE.ADDRESS.TABLE,Y
 1420         STA MSG.PNTR
 1430         LDA MESSAGE.ADDRESS.TABLE+1,Y
 1440         STA MSG.PNTR+1
 1450         LDA #0
 1460         STA MSG.SCANNER
 1470  .1     JSR GET.NEXT.CHAR.FROM.MESSAGE
 1480         BNE .3
```

```
1490        RTS          $00:  EOM
1500  .3    BPL .5       SPECIAL ACTION
1510        JSR MON.COUT PRINT THE CHARACTER
1520  .4    JMP .1
1530  *---------------------------------
1540  .5    CMP #$40     CHECK FOR VTAB
1550        BCS .6       YES
1560        CMP #$29     IN RANGE FOR HTAB?
1570        BCS .4       NO, IGNORE
1580        STA MON.CH
1590        DEC MON.CH
1600        BCC .4       ...ALWAYS
1610  *---------------------------------
1620  .6    CMP #$58     IN RANGE FOR VTAB?
1630        BCS .7       NO
1640        AND #$1F     MASK VALUE
1650        STA MON.CV   YES
1660        JSR MON.VTAB
1670        JMP .4
1680  *---------------------------------
1690  .7    EOR #$60     CHECK FOR TOKENS
1700        CMP #7       $60 THROUGH $66
1710        BCS .4       NOT TOKEN, SO IGNORE
1720        ASL          MAKE DUBLE INDEX
1730        TAX
1740        LDA /.4-1    PUT RETURN ON STACK
1750        PHA          TO SIMULATE A JSR ADDR,X
1760        LDA #.4-1
1770        PHA
1780        LDA MSGTKNTBL+1,X
1790        PHA
1800        LDA MSGTKNTBL,X
1810        PHA
1820        RTS
1830  *---------------------------------
1840  MSGTKNTBL
1850        .DA MON.HOME-1
1860        .DA MSG.REPEAT-1
1870        .DA LONG.DELAY-1
1880        .DA MON.NORMAL-1
1890        .DA MON.INVERSE-1
1900        .DA MON.CLREOL-1
1910        .DA MON.CLREOP-1
1920  *---------------------------------
1930  MSG.REPEAT
1940        JSR GET.NEXT.CHAR.FROM.MESSAGE
1950        TAX          NUMBER OF MULTIPLES
1960        JSR GET.NEXT.CHAR.FROM.MESSAGE
1970  .1    JSR MON.COUT
1980        DEX
1990        BNE .1
2000        RTS
2010  *---------------------------------
2020  LONG.DELAY
```

```
2030        LDY #12
2040 .1     JSR MON.WAIT DELAY 167309 CYCLES
2050        DEY
2060        BNE .1
2070        RTS
2080 *---------------------------------
2090 GET.NEXT.CHAR.FROM.MESSAGE
2100        LDY MSG.SCANNER
2110        LDA (MSG.PNTR),Y
2120        INC MSG.SCANNER
2130        BNE .1
2140        INC MSG.PNTR+1
2150 .1     CMP #0
2160        RTS
2170 *---------------------------------
2180 MESSAGE.ADDRESS.TABLE
2190        .DA MSG0
2200        .DA MSG1
2210        .DA MSG2
2220        .DA MSG3
2230 *---------------------------------
2240 MSG0   .HS 60       HOME SCREEN
2250 * CELL 1 -- VOCABULARY CHECK
2260        .HS 64       INVERSE MODE
2270        .HS 6129AD   4A DASHES
2280        .HS 28ADAD   2 DASHES
2290        .HS 28ADAD
2300        .HS 28ADAD   2 DASHES
2310        .HS 28ADAD   2 DASHES
2320        .HS 28ADAD   2 DASHES
2330        .HS 28ADAD   2 DASHES
2340        .HS 286129AD 41 DASHES
2350        .HS 63       NORMAL MODE
2360        .HS 4205     VTAB 3, HTAB 5
2370        .AS -/DEMONSTRATION OF MESSAGE PRINTER/
2380        .HS 440F     VTAB 5, HTAB 15
2390        .AS -/S-C SOFTWARE/
2400        .HS 450E     VTAB 6, HTAB 14
2410        .AS -/P. O. BOX 5537/
2420        .HS 460B     VTAB 7, HTAB 11
2430        .AS -/RICHARDSON, TX 75080/
2440        .HS 4A       VTAB 11
2450        .HS 00
2460 *---------------------------------
2470 MSG1   .HS 490166   VTAB 10, HTAB 1, CLR EOP
2480        .AS -/SELECT ONE:  /
2490        .HS 00
2500 *---------------------------------
2510 MSG2   .HS 570165   VTAB 24, HTAB 1, CLR EOL
2520        .HS 64       INVERSE MODE
2530        .AS -/ <SPACE> FOR MENU, <RETURN> FOR MORE /
2540        .HS 6300     NORMAL MODE, EOM
2550 *---------------------------------
2560 MSG3   .HS 87878D
```

```
2570        .AS -/***SYNTAX ERROR/
2580        .HS 8D00
```

```
=====================================================================
DOCUMENT :AAL-8011:Articles:BagsDisks4Sale.txt
=====================================================================
```

Bags, Boxes, et cetera

Since I sell software in stores, I buy a lot of zip-lock bags, cardboard mailing boxes, diskettes, and so on.  I thought that maybe you need some of these, and haven't been able to find a source at good prices in small quantities.  I will sell you some of mine, at the follwoing prices:

```
     6"x9" zip-lock bags     $8.50/100
     9"x12" zip-lock bags    $12/100
     Verbatim diskettes
         without hubrings  $30 for box of ten, $265 for 100
         with hubrings     $32 for box of ten, $285 for 100
```

Anything else you need?  Let me know, maybe I have it or can get it for you or tell you where you can get it at a good price.

=============================================================================
DOCUMENT :AAL-8011:Articles:Front.Page.txt
=============================================================================

Volume 1 -- Issue 2 November, 1980

Our second issue is 33% larger than the first! And not only so, but also there is useful information on the back page! I found a source for 6x9 white envelopes, so your address can be external to the newsletter, and so your copy will arrive in better condition. In less than a month since the newsletter was first announced, we already have over 45 paid subscribers. They are sprinkled all over the map, including one in Japan!

In This Issue...



A Bug in S-C Assembler II Disk Version 4.0

One real bug has turned up, and a few of you have had the bad luck to discover it the hard way. The assembler is free-format, in that opcodes and directives may start in any column after the blank which terminates the label field. However, the ".IN" directive will malfunction unless there are at least six spaces. If you tab over before typing ".IN" there will be no problem. However, if you type your line like "1230  .IN FILE1", with only two spaces between the line number and the period, you are in for a long wait. The processor goes into a loop printing D's. If you have the MONC mode on, you will see "LOADDDDDDDDD....." with D's forever appearing on your screen. Remember to TAB OVER, and it will not malfunction.

One fancied bug has been reported, and I would like to explain it. A user pointed out that you cannont shorten the SAVE command to three letters if you wish to save the source program on a disk file. Why? Because "SAVE" or "SAV" with no file name is not a DOS command. It is an assembler command to save the source program on cassette tape! On the other hand, SAVE with a filename is not an assembler command. It is a DOS command, and the assembler never sees it. The same goes for "LOAD", "LOA", and LOAD with a filename.

==============================================================================
DOCUMENT :AAL-8011:Articles:Sim.KeyPad.txt
==============================================================================

A Simulated Numeric Key-Pad

This little program will turn part of your Apple's keyboard into a simulated numeric key-pad. A lot cheaper than buying a real one! It is set up to run in page 3, and assumes you are using DOS. If not, just change line 1120 to an RTS.

If you BRUN it or CALL it at 768, the input vector is patched to input all characters through the NKP program. Typing a control-S will toggle the numeric key-pad translator on and off. When the translator is off, all keyboard action is normal, except that another control-S will turn it back on again. When the translator is on, all keys which are not part of the simulated key-pad will input normally.

The keys translated by the simulator are listed in line 1390. The slash key duplicates RETURN, because it is easier to hit when yu are entering a lot of numbers. For the same reason, the L-key duplicates "-", in case you are in a hurry to enter negative numbers too. The space bar is used for "0". I set it up to use "NM," for "123", "HJK" for "456", and "YUI" for "789". You shuld be able to easily change these translations to any other combination, by changing lines 1390-1420.

The heart of the translator is the search loop in lines 1240-1280. If the input character is not found in CHRTBL, the search loop drops out and the character is not changed. If the character is found, line 1310 picks up the alias for the key, and returns. That's all there is to it!

==============================================================================
DOCUMENT :AAL-8011:Articles:Src.On.TxtFiles.txt
==============================================================================

Assembly Source on Text Files

Version 4.0 of the S-C Assembler II allows you to EXEC a source program, if it is on a DOS text file. This is handy if you have created it with a different editor, or perhaps with a compiler. But what if you want to go the other way? What if you want to SAVE a source program on a text file, so that it can be used in another editor, or by another assembler?

There is no built-in command to allow it, so I have now written a separate program to do it. The program loads at $0800 thru $093C, and does not borrow any code from the assembler. It does use some routines in the Monitor ROMs, and the DOS I/O rehook routine. If you BRUN the program, it will assume the pointers at $CA,CB and $4C,4D are bracketing a valid assembly source program, and try to list it on a text file.

The main body of the program is in lines 1190 thru 1630. Lines 1200 and 1210 serve to un-hook the S-C Assembler II from the output. They will also turn off your printer, if you had it on. Lines 1220 and 1230 tell DOS that it should recognize commands printed after a control-D. Lines 1240 and 1250 change the prompt symblol to a blank, so that the monitor input subroutine will not print a colon or some other character as the prompt when reading the file name.

Lines 1290-1360 request you to enter a file name, read it into the monitor buffer starting at $0200, and move it to a safe place at $0280. It has to be moved, because when we print DOS commands later the area starting at $0200 will be written on by DOS.

Once the file name you have typed is safely stored at $0280 and following, lines 1410 thru 1490 will set up the file for writing. This is done in five steps. First, close all files. Second, issue an OPEN-DELETE-OPEN sequence, with the file name (of course); this will make sure that we are writing on a fresh empty file. Then the WRITE command is sent, and we are ready to roll.

Line 1530 calls a subroutine which lists your source program. Since the file is OPEN and in WRITE mode, the listing goes into your text file. If you have MON O mode set, you will also see the listing on your screen. Note that it is not really necessary for me to use a subroutine at this point. ASM.LIST is only called once, and it is not very long. But I did it anyway, to keep the main body short enough to fit on a page (of paper), easy to understand, modular, structured, etc.

After the listing is completed, line 1570 will close the text file. Lines 1610 and 1620 turn off the DOS run flag, so that DOS will not look for control-D commands. And finally, line 1630 re-enters the S-C Assembler II through its soft entry point.

Lines 1670 thru 1780 are text strings, printed by the subroutine named PRINT.QUOTE. Each string is written with the sign bit of every byte zero except for the last byte. The sign bit of the last byte is 1, telling PRINT.QUOTE that it is finished. For example, the first message is the word "CLOSE" and a carriage return. The carriage return is entered in hex with the sign bit 1 as in $8D. The second message is the word "OPEN", and the letter "N" is preceded by a minus sign in the .AS directive to indicate that the sign bit should be 1.

The PRINT.QUOTE subroutine is at lines 2140 thru 2200. It expects the Y-register to contain the offset of the desired message from the beginning of all the messages at QTS. It calls on PRINT.CHAR to actually send each character.

PRINT.CHAR, at lines 2020 thru 2100, calls on the monitor print character routine at $FDED. This branches through DOS, and DOS writes the character on the text file. PRINT.CHAR saves and restores the Y-register and A-register contents. It also sets the sign bit on each character before printing it. Upon exit, the status will reflect the value of the character printed.

Lines 1820-1980 issue a DOS command. The Y-register points at one of the message strings in QTS. Control-D is printed, followed by the command key word, a space, and file name you previously typed. Since DOS does not allow slot and drive specifications on the WRITE command, and since it is sufficient to specify them only once, the subroutine chops them off after printing them once. The logic for this is in lines 1910-1940: after printing a comma, it is replaced with a carriage return. The next time the name is printed, the carriage return will be the end.

The subroutine which really controls the listing is in lines 2330-2450. The first four instructions set up a zero-page pointer SRCP to point at the beginning of your source program. Lines 2380-2420 compare the pointer with HIMEM to see if the listing is completed. If you really had no source program, we would already be finished at this point. If there is another line (or more), the subroutine named ASM.LIST.LINE is called to list the next lne. The process is repeated until the last line has been printed onto your text file.

At this point it might be helpful to explain how source lines are stored in memory. Each line begins with a single byte which contains the byte-count of the line. Next are a byte-pair containing the line number of the line, in the usual backwards 6502 format. The text of the line follows, and a final byte containing $00 ends the line. No carriage return is stored. Blanks are treated specially. A single blank is stored as $81. Two blanks in a row are replaced by one byte of value $82. Any string of blanks up to 63 blanks is thus replaced by a single token of value $80 plus the blank count. Longer strings of blanks will take more than one token.

For example, the source line

```
       1000 ABC    LDA SAM
```

```
is stored as:  0F  (total of 15 bytes in line image)
               E8 03   (line number 1000)
               41 42 43 84   ("ABC" and 4 blanks)
               4C 44 41 81   ("LDA" and 1 blank)
               53 41 4D      ("SAM")
               00      (end of line indicator)
```

The subroutine ASM.LIST.LINE at lines 2490-2610 prints one source line. A subroutine named GNB ("get next byte") is called to skip over the length byte, and to pick up the line number. PRINT.LINNUM is called to convert the line number to decimal and print it, with leading zeroes if necessary, as a four digit number. The loop at lines 2570-2600 is seeded with a blank (because the blank between the line number and the label field is not actually stored in the source program), and the text of the line is printed. The loop prints a character, and then calls NEXT.TOKEN to get the next one. When the token returned equals $00, the line is finished.

GNB, lines 2630-2690, clears the queued blank count, picks up the character pointed at by SRCP, and increments SRCP.

NEXT.TOKEN, lines 2710-2820, tests the blank count. If it is non-zero, the count is decremented and a blank ($20) character is returned. If the count was zero, the next character is picked up from the line. If this character is not a blank count token, it is returned and the pointer in SRCP is incremented. If the character is a blank count token, it is saved, the SRCP pointer is incremented past the token, and then the count is decremented and a blank returned.

The PRINT.LINNUM routine, lines 2860-3170, is a revision of a routine used in the Integer BASIC ROMs. I think it is commented well enough for you to follow. The general idea is to divide by 1000 and print the quotient; divide the remainder by 100 and print the quotient; then by 10; and finally print the remainder.

Since several of you have asked me to provide the capability to list programs onto text files, you should be pleased with this program. If you do not need it, then maybe it has shed some light on the internal structure of part of the assembler, or served as a tutorial in programming.

=============================================================================
DOCUMENT :AAL-8011:Articles:Use.For.USR.Cmd.txt
=============================================================================

A Use for the USR Command

The S-C Assembler II Version 4.0 has one user-programmable command, called "USR". (The Quick Reference Card spells it erroneously "USEr".) One good use for it is to re-print the current symbol table.

After an assembly, if the listing was not printed, it is often desirable to be able to see what the spelling or value of a symbol or group of symbols is. If the VAL command is not enough for you, then the following steps will set up the USR command to re-list the symbol table on the screen. And, if your printer is selected, it will also print there.

Get into the assembler, by using BRUN ASMDISK 4.0 from either Applesoft or Integer BASIC. Type "$1E4EL" after the prompt. The first two lines listed should be "LDY #$02" and "STY $E1". If they are not, you have a different version. (It may still be version 4.0, but slightly different.) The "LDY#$02" line is the first instruction of the symbol table printing subroutine.

Patch the USR vector by typing "$1007:4E 1E", and then BSAVE the result like this:

```
       :BSAVE ASMDISK 4.0 (WITH USR),A$1000,L$14FB
```

This new version, whenever you type "USR", will print out the current symbol table. It will look exactly the same as the symbol table pritned out at the end of an assembly.

==============================================================================
DOCUMENT :AAL-8011:Articles:Variable.XRef.txt
==============================================================================

Variable Cross Reference for Applesoft Programs

Besides illustrating a lot of programming techniques, the VCR program is a very useful tool when you are writing large Applesoft programs. As listed here, it requires a 48K Apple, and assumes that HIMEM is set to at least $8AA7. You BRUN it, and it sets up the &-vector. When you are ready to print a cross reference, you merely type "&" and a carriage return, and out it comes. It is VERY fast: about 15 times faster than the VCR program included in Apple's DOS Tool Kit. It also takes less memory than Apple's version, both for the program itself and for the tables it constructs during execution.

The main body of the program is in lines 1400 thru 1460. After calling INITIALIZATION, the subroutine PROCESS.LINE is called until there are no more lines. Then PRINT.REPORT is called, and finally INITIALIZATION is called again to restore Applesoft's tables to their original form.

INITIALIZATION sets up PNTR to point to the beginning of the program, and EOT to point to the end of the table area. It also clears out a set of 26 two-byte pointers in HSHTBL (hash table). PROCESS.ONE scans a single line looking for variables by calling SCAN.FOR.VARIABLES, until the end of the program is reached. PRINT.REPORT merely prints a nice orderly report from the data which has been stored in the table by SCAN.FOR.VARIABLES.

The symbol table routines used in VCR are very similar to the ones used inside S-C Assembler II Version 4.0. There are 26 pointers starting at HSHTBL ($280), each one representing one letter of the alphabet. The first letter of a variable name selects one of these pointers. The pointer points at the first entry in a chain of variable names. When a new variable name is found, it is inserted in the appropriate chain at the place where it will be in alphabetical order. A sub-chain is kept for each variable name of all the line numbers from which it is referenced. The line number chain is maintained in numerical order. Thus there is no sorting necessary when it comes time to print the report.

Since no routines from the Applesoft ROMs are used, VCR will work with no changes with the RAM version of Aplesoft. Since it loads below $9000, it will not conflict with Neil Konzen's PLE (Program Line Editor). Since it is just straight-forward code, with no address tables or embedded data, you can easily relocate it to a different running address; only the 3-byte instructions with the third byte equal to $88, $89, or $8A need to be changed. Or, you can type it in, and use a different origin (line 1040).

If you like to modify programs, this one needs one improvement. (Only one?) I forgot to take note of the FN token, so any FN definitions or uses will look like references to an array variable.  Another kind of modification, called "major" perhaps, will turn the VCR into LNCR (Line Number Cross Reference).

```
==============================================================================
DOCUMENT :AAL-8011:DOS3.3:S.NumericKeyPad.txt
==============================================================================

 1000  *---------------------------------
 1010  *     NUMERIC KEY PAD FOR APPLE
 1020  *---------------------------------
 1030         .OR $300
 1040         .TF B.NKP
 1050  *---------------------------------
 1060         LDA #1
 1070         STA TOGGLE
 1080         LDA #NKP
 1090         STA $38
 1100         LDA /NKP
 1110         STA $39
 1120         JMP $3EA
 1130  *---------------------------------
 1140  TOGGLE .BS 1
 1150  SAVEY  .BS 1
 1160  *---------------------------------
 1170  NKP
 1180         JSR $FD1B
 1190         CMP #$93     CONTROL-S
 1200         BEQ .4
 1210         BIT TOGGLE
 1220         BMI .2       NOT IN NUMERIC MODE
 1230         STY SAVEY
 1240         LDY #TBLSIZ-1
 1250  .1     CMP CHRTBL,Y
 1260         BEQ .3       FOUND IN TABLE
 1270         DEY
 1280         BPL .1
 1290         LDY SAVEY
 1300  .2     RTS
 1310  .3     LDA ALIAS,Y
 1320         LDY SAVEY
 1330         RTS
 1340  .4     LDA TOGGLE
 1350         EOR #$80
 1360         STA TOGGLE
 1370         JMP $FD0C
 1380  *---------------------------------
 1390  CHRTBL .AS -"/L NM,HJKYUI"
 1400  TBLSIZ .EQ *-CHRTBL
 1410  ALIAS  .HS 8D
 1420         .AS -"-0123456789"
 1430  *---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8011:DOS3.3:S.TEXT.LIST.txt
==============================================================================

 1000        .LIST OFF
 1010 *---------------------------------
 1020 *      WRITE ASSEMBLY SOURCE ON A TEXT FILE
 1030 *---------------------------------
 1040        .OR $800
 1050 MON.PROMPT .EQ $33
 1060 PP     .EQ $CA,CB
 1070 HIMEM  .EQ $4C,4D
 1080 DOS.RUNFLAG .EQ $D9
 1090 MON.BUFFER .EQ $200
 1100 DOS.BUFFER .EQ $280
 1110 MON.GETLN  .EQ $FD6A
 1120 MON.CROUT  .EQ $FD8E
 1130 MON.COUT   .EQ $FDED
 1140 MON.SETVID .EQ $FE93
 1150 DOS.REHOOK .EQ $3EA
 1160 BLANK.COUNT .EQ $00
 1170 SRCP   .EQ $01,02
 1180 LINNUM .EQ $03,04
 1190 *---------------------------------
 1200 TEXT.LIST
 1210        JSR MON.SETVID
 1220        JSR DOS.REHOOK
 1230        LDA #$FF
 1240        STA DOS.RUNFLAG
 1250        LDA #' +$80  SET PROMPT CHAR = BLANK
 1260        STA MON.PROMPT
 1270 *---------------------------------
 1280 *      GET FILE NAME
 1290 *---------------------------------
 1300        LDY #QFILNAM-QTS
 1310        JSR PRINT.QUOTE
 1320        JSR MON.GETLN
 1330        LDY #$7F     MOVE FILE NAME TO SEPARATE BUFFER
 1340 .1     LDA MON.BUFFER,Y
 1350        STA DOS.BUFFER,Y
 1360        DEY
 1370        BPL .1
 1380 *---------------------------------
 1390 *      SET UP THE TEXT FILE
 1400 *      (CLOSE, OPEN, DELETE, OPEN, WRITE)
 1410 *---------------------------------
 1420        JSR CLOSE.FILE
 1430        LDY #QOPEN-QTS
 1440        JSR ISSUE.DOS.COMMAND
 1450        LDY #QDELETE-QTS
 1460        JSR ISSUE.DOS.COMMAND
 1470        LDY #QOPEN-QTS
 1480        JSR ISSUE.DOS.COMMAND
```

```
1490        LDY #QWRITE-QTS
1500        JSR ISSUE.DOS.COMMAND
1510 *---------------------------------
1520 *      LIST THE SOURCE PROGRAM
1530 *---------------------------------
1540        JSR ASM.LIST
1550 *---------------------------------
1560 *      CLOSE THE FILE
1570 *---------------------------------
1580        JSR CLOSE.FILE
1590 *---------------------------------
1600 *      RETURN TO CALLER
1610 *---------------------------------
1620        LDA #0
1630        STA DOS.RUNFLAG
1640        JMP $1003
1650 *---------------------------------
1660 *      MESSAGE TEXT
1670 *---------------------------------
1680 QTS    .EQ *
1690 QCLOSE .AS /CLOSE/
1700        .HS 8D
1710 QOPEN  .AS /OPE/
1720        .AS -/N/
1730 QDELETE .AS /DELET/
1740        .AS -/E/
1750 QWRITE .AS /WRIT/
1760        .AS -/E/
1770 QFILNAM .HS 0D
1780        .AS /TEXT FILE NAME:/
1790        .AS -/ /
1800 *---------------------------------
1810 *      ISSUE DOS COMMAND
1820 *---------------------------------
1830 ISSUE.DOS.COMMAND
1840        LDA #$84     CONTROL-D
1850        JSR PRINT.CHAR
1860        JSR PRINT.QUOTE
1870        LDY #0
1880        LDA #'       PRINT A SPACE
1890 .5     JSR PRINT.CHAR
1900        CMP #$8D
1910        BEQ .7
1920        CMP #$AC     COMMA?
1930        BNE .6
1940        LDA #$8D
1950        STA DOS.BUFFER-1,Y
1960 .6     LDA DOS.BUFFER,Y
1970        INY
1980        BNE .5       ...ALWAYS
1990 .7     RTS
2000 *---------------------------------
2010 *      PRINT CHARACTER
2020 *---------------------------------
```

```
2030  PRINT.CHAR
2040         PHA
2050         STY PC.SAVEY
2060         ORA #$80
2070         JSR MON.COUT
2080         LDY PC.SAVEY
2090         PLA
2100         RTS
2110  PC.SAVEY .BS 1
2120  *---------------------------------
2130  *      PRINT A QUOTATION
2140  *---------------------------------
2150  PRINT.QUOTE.NEXT
2160         INY
2170  PRINT.QUOTE
2180         LDA QTS,Y
2190         JSR PRINT.CHAR
2200         BPL PRINT.QUOTE.NEXT
2210         RTS
2220  *---------------------------------
2230  *      CLOSE ALL FILES
2240  *---------------------------------
2250  CLOSE.FILE
2260         JSR MON.CROUT
2270         LDA #$84
2280         JSR PRINT.CHAR CONTROL-D
2290         LDY #QCLOSE-QTS
2300         JMP PRINT.QUOTE
2310  *---------------------------------
2320  *      LIST SOURCE PROGRAM
2330  *---------------------------------
2340  ASM.LIST
2350         LDA PP
2360         STA SRCP
2370         LDA PP+1
2380         STA SRCP+1
2390  .1     LDA SRCP
2400         CMP HIMEM
2410         LDA SRCP+1
2420         SBC HIMEM+1
2430         BCS .2       FINISHED
2440         JSR ASM.LIST.LINE
2450         JMP .1
2460  .2     RTS
2470  *---------------------------------
2480  *      LIST ONE SOURCE LINE
2490  *---------------------------------
2500  ASM.LIST.LINE
2510         JSR GNB      SKIP OVER BYTE COUNT
2520         JSR GNB      GET LINE NUMBER
2530         STA LINNUM
2540         JSR GNB
2550         STA LINNUM+1
2560         JSR PRINT.LINNUM
```

```
2570        LDA #'       BLANK
2580 .1     JSR PRINT.CHAR
2590        JSR NEXT.TOKEN
2600        CMP #0
2610        BNE .1
2620        JMP MON.CROUT
2630 *---------------------------------
2640 GNB    LDY #0
2650        STY BLANK.COUNT
2660        LDA (SRCP),Y
2670 GNBI   INC SRCP
2680        BNE .1
2690        INC SRCP+1
2700 .1     RTS
2710 *---------------------------------
2720 NEXT.TOKEN
2730        LDY #0
2740        LDA BLANK.COUNT
2750        BNE .1
2760        LDA (SRCP),Y
2770        BPL GNBI
2780        AND #$7F
2790        STA BLANK.COUNT
2800        JSR GNBI
2810 .1     DEC BLANK.COUNT
2820        LDA #'       BLANK
2830        RTS
2840 *---------------------------------
2850 *      PRINT LINE NUMBER
2860 *---------------------------------
2870 PRINT.LINNUM
2880        LDX #3       PRINT 4 DIGITS
2890 .3     LDA #'0      SET DIGIT TO ASCII ZERO
2900 .1     PHA          PUSH DIGIT ON STACK
2910        SEC          SUBTRACT CURRENT DIVISOR
2920        LDA LINNUM
2930        SBC PLNTBL,X
2940        PHA          SAVE BYTE ON STACK
2950        LDA LINNUM+1
2960        SBC PLNTBH,X
2970        BCC .2       LESS THAN DIVISOR
2980        STA LINNUM+1
2990        PLA          GET LOW BYTE OFF STACK
3000        STA LINNUM
3010        PLA          GET DIGIT FROM STACK
3020        ADC #0       INCREMENT DIGIT
3030        BNE .1       ...ALWAYS
3040 .2     PLA          DISCARD BYTE FROM STACK
3050        PLA          GET DIGIT FROM STACK
3060        JSR PRINT.CHAR
3070        DEX          NEXT DIGIT
3080        BPL .3
3090        RTS          RETURN
3100 *---------------------------------
```

```
3110  PLNTBL .DA #1
3120         .DA #10
3130         .DA #100
3140         .DA #1000
3150  PLNTBH .DA /1
3160         .DA /10
3170         .DA /100
3180         .DA /1000
```

=================================================================
DOCUMENT :AAL-8011:DOS3.3:S.Var.XRef.txt
=================================================================

```
 1000  *----------------------------------
 1010  *      VARIABLE CROSS REFERENCE
 1020  *       FOR APPLESOFT PROGRAMS
 1030  *----------------------------------
 1040  ZZ.BEG .EQ $8800
 1050         .OR ZZ.BEG
 1060         .TF B.VCR
 1070  *----------------------------------
 1080         LDA #$4C     AMPERSAND VECTOR
 1090         STA $3F5
 1100         LDA #VCR
 1110         STA $3F6
 1120         LDA /VCR
 1130         STA $3F7
 1140         RTS
 1150  *----------------------------------
 1160  PNTR   .EQ $18,19   POINTER INTO PROGRAM
 1170  DATA   .EQ $1A THRU $1D
 1180  LZFLAG .EQ $1A      LEADING ZERO FLAG
 1190  NEXTLN .EQ $1A,1B   ADDRESS OF NEXT LINE
 1200  LINNUM .EQ $1C,1D   CURRENT LINE NUMBER
 1210  STPNTR .EQ $1E,1F   POINTER INTO VARIABLE TABLE
 1220  TPTR   .EQ $9B,9C   TEMP POINTER
 1230  SYMBOL .EQ $9D THRU $A4  8 BYTES
 1240  VARNAM .EQ SYMBOL+1
 1250  HSHTBL .EQ $280
 1260  ENTRY.SIZE .EQ $A5,A6
 1270  *----------------------------------
 1280  PRGBOT .EQ $67,68   BEGINNING OF PROGRAM
 1290  LOMEM  .EQ $69,6A   BEGINNING OF VARIABLE SPACE
 1300  EOT    .EQ $6B,6C   END OF VARIABLE TABLE
 1310  *----------------------------------
 1320  TKN.REM    .EQ 178
 1330  TKN.DATA   .EQ 131
 1340  *----------------------------------
 1350  MON.CH     .EQ $24
 1360  MON.PRBL2  .EQ $F94A
 1370  MON.COUT   .EQ $FDED
 1380  MON.CROUT  .EQ $FD8E
 1390  *----------------------------------
 1400  VCR
 1410         JSR INITIALIZATION
 1420  .1     JSR PROCESS.LINE
 1430         BNE .1       UNTIL END OF PROGRAM
 1440         JSR PRINT.REPORT
 1450         JSR INITIALIZATION  ERASE VARIABLE TABLE
 1452         LDA #0       CLEAR $A4 SO APPLESOFT WILL
 1454         STA $A4      WORK CORRECTLY
 1460         RTS
```

```
1470  *--------------------------------
1480  INITIALIZATION
1490         LDA LOMEM
1500         STA EOT
1510         LDA LOMEM+1
1520         STA EOT+1
1530         LDX #52      # OF BYTES FOR HASH POINTERS
1540         LDA #0
1550  .1     STA HSHTBL-1,X
1560         DEX
1570         BNE .1
1580         LDA PRGBOT
1590         STA PNTR
1600         LDA PRGBOT+1
1610         STA PNTR+1
1620         RTS
1630  *--------------------------------
1640  PROCESS.LINE
1650         LDY #3       CAPTURE POINTER AND LINE #
1660  .1     LDA (PNTR),Y
1670         STA DATA,Y
1680         DEY
1690         BPL .1
1692         LDA DATA+1   CHECK IF END
1694         BEQ .3       YES
1700         CLC          SKIP OVER DATA
1710         LDA PNTR
1720         ADC #4
1730         STA PNTR
1740         BCC .2
1750         INC PNTR+1
1760  .2     JSR SCAN.FOR.VARIABLES
1770         LDA DATA
1780         STA PNTR
1790         LDA DATA+1
1800         STA PNTR+1
1810  *      BNE .3
1820  .3     RTS
1830  *--------------------------------
1840  SCAN.FOR.VARIABLES
1850  .1     JSR GET.NEXT.VARIABLE
1860         BEQ .3       END OF LINE
1870         JSR PACK.VARIABLE.NAME
1880         JSR SEARCH.VARIABLE.TABLE
1890         BCC .2       FOUND SAME VARIABLE
1900         LDA #0
1910         STA SYMBOL+4 START OF LINE NUMBER CHAIN
1920         STA SYMBOL+5
1930         LDA LINNUM+1 MSB FIRST
1940         STA SYMBOL+6
1950         LDA LINNUM
1960         STA SYMBOL+7
1970         LDA #8       ADD 8 BYTE ENTRY
1980         JSR ADD.NEW.ENTRY
```

```
1990          JMP .1
2000  .2      JSR SEARCH.LINE.CHAIN
2010          BCC .1        FOUND SAME LINE NUMBER
2020          LDA #4        ADD 4 BYTE ENTRY
2030          JSR ADD.NEW.ENTRY
2040          JMP .1
2050  .3      RTS
2060  *--------------------------------
2070  GET.NEXT.VARIABLE
2080  .1      JSR NEXT.CHAR.NOT.QUOTE
2090          BEQ .2        END OF LINE
2100          CMP #TKN.DATA
2110          BEQ .3
2120          CMP #TKN.REM
2130          BEQ .2        SKIP TO NEXT LINE
2140          JSR LETTER    LETTER?
2150          BCC .1        NO, KEEP LOOKING
2160  .2      RTS
2170  *   DATA, SO SKIP TO NEXT STATEMENT
2180  .3      JSR NEXT.CHAR.NOT.QUOTE
2190          BEQ .2        EOL, RETURN
2200          CMP #':       COLON?
2210          BNE .3        NOT END YET
2220          BEQ .1        ...ALWAYS
2230  *--------------------------------
2240  NEXT.CHAR.NOT.QUOTE
2250  .1      JSR NEXT.CHAR
2260          BEQ .2        EOL, RETURN
2270          CMP #'"       QUOTE?
2280          BEQ .3        YES, SCAN OVER QUOTATION
2290  .2      RTS           RETURN
2300  .3      JSR NEXT.CHAR
2310          BEQ .2        EOL, RETURN
2320          CMP #'"       TERMINAL QUOTE?
2330          BNE .3        NOT YET
2340          BEQ .1        ...ALWAYS
2350  *--------------------------------
2360  *       NEXT CHARACTER FROM LINE
2370  *         CALL:  JSR NEXT.CHAR
2380  *       RETURN:  (A)=CHAR FROM LINE
2390  *                IF CHAR .NE. EOL,
2400  *                    INCREMENT PNTR AND
2410  *                    STATUS Z=0
2420  *                IF CHAR .EQ. EOL,
2430  *                    STATUS Z=1
2440  *--------------------------------
2450  NEXT.CHAR
2460          LDY #0
2470          LDA (PNTR),Y
2480          BEQ .1        EOL
2490          INC PNTR      BUMP POINTER
2500          BNE .1
2510          INC PNTR+1
2520  .1      RTS
```

```
2530  *---------------------------------
2540  PACK.VARIABLE.NAME
2550         STA VARNAM   FIRST CHAR OF NAME
2560         LDA #'       BLANKS FOR OTHER TWO CHARS
2570         STA VARNAM+1
2580         STA VARNAM+2
2590         JSR NEXT.CHAR
2600         BEQ .5       END OF LINE
2610         JSR LTRDIG
2620         BCC .2       NOT LETTER OR DIGIT
2630         STA VARNAM+1
2640  .1     JSR NEXT.CHAR IGNORE EXCESS NAME
2650         BEQ .5       END OF LINE
2660         JSR LTRDIG
2670         BCS .1       LETTER OR DIGIT
2680  .2     CMP #'$      DOLLAR SIGN?
2690         BEQ .3       YES
2700         CMP #'%      PER CENT?
2710         BNE .4       NO
2720  .3     STA VARNAM+2
2730         JSR NEXT.CHAR
2740         BEQ .5       END OF LINE
2750  .4     CMP #'(      LEFT PAREN?
2752         BEQ .6       YES
2754         CMP #'"      QUOTE?
2760         BNE .5       NO
2762         LDA PNTR     YES, BACK UP POINTER
2763         BNE .7
2764         DEC PNTR+1
2765  .7     DEC PNTR
2766         RTS
2770  .6     LDA VARNAM+2 SET HIGH BIT
2780         ORA #$80     TO FLAG ARRAY
2790         STA VARNAM+2 REFERENCE
2800  .5     RTS
2810  *---------------------------------
2820  SEARCH.VARIABLE.TABLE
2830         SEC          CONVERT 1ST CHAR TO
2840         LDA VARNAM   HASH TABLE INDEX
2850         SBC #'A
2860         ASL
2870         ADC #HSHTBL
2880         STA STPNTR
2890         LDA /HSHTBL
2900         ADC #0
2910         STA STPNTR+1
2920  *---   FALL INTO CHAIN SEARCH ROUTINE
2930  *---------------------------------
2940  CHAIN.SEARCH
2950  .1     LDY #0       POINT AT POINTER IN ENTRY
2960         LDA (STPNTR),Y
2970         STA TPTR
2980         INY
2990         LDA (STPNTR),Y
```

```
3000        BEQ .4       END OF CHAIN, NOT IN TABLE
3010        STA TPTR+1
3020        LDX #2       2 MORE CHARS IN SYMBOL
3030        LDY #2       POINT AT NAME IN ENTRY
3040  .2    LDA (TPTR),Y COMPARE NAMES
3050        CMP SYMBOL,Y
3060        BCC .3       NOT THIS ONE, BUT KEEP LOOKING
3070        BNE .4       NOT IN THIS CHAIN
3080        DEX
3090        BEQ .5       NAME IS THE SAME
3100        INY          NEXT BYTE PAIR
3110        BNE .2       ...ALWAYS
3120  *---------------------------------
3130  .3    JSR .5       UPDATE POINTER, CLEAR CARRY
3140        BCC .1       ...ALWAYS
3150  *---------------------------------
3160  .4    SEC          DID NOT FIND
3170        RTS
3180  *---------------------------------
3190  .5    LDA TPTR
3200        STA STPNTR
3210        LDA TPTR+1
3220        STA STPNTR+1
3230        CLC
3240        RTS
3250  *---------------------------------
3260  ADD.NEW.ENTRY
3270        STA ENTRY.SIZE
3280        CLC          SEE IF ROOM
3290        LDX #1
3300        LDY #0
3310        STY ENTRY.SIZE+1
3320  .1    LDA (STPNTR),Y  GET CURRENT POINTER
3330        STA SYMBOL,Y
3340        LDA EOT,Y
3350        STA (STPNTR),Y
3360        STA TPTR,Y
3370        ADC ENTRY.SIZE,Y
3380        STA EOT,Y
3390        INY
3400        DEX
3410        BPL .1
3420  *---  SEE IF GOING TO BE ENOUGH ROOM
3430        LDA EOT
3440        CMP #ZZ.BEG
3450        LDA EOT+1
3460        SBC /ZZ.BEG
3470        BCS .3       MEM FULL ERR
3480  *---  MOVE ENTRY INTO VARIABLE TABLE
3490        LDY ENTRY.SIZE
3500        DEY
3510  .2    LDA SYMBOL,Y
3520        STA (TPTR),Y
3530        DEY
```

```
3540        BPL .2
3550        LDA TPTR
3560        STA STPNTR
3570        LDA TPTR+1
3580        STA STPNTR+1
3590        RTS
3600 .3     JMP MEM.FULL.ERR
3610 MEM.FULL.ERR
3620        BRK
3630 *---------------------------------
3640 SEARCH.LINE.CHAIN
3650        CLC          ADJUST POINTER TO START
3660        LDA STPNTR   OF LINE # CHAIN
3670        ADC #4
3680        STA SYMBOL
3690        LDA STPNTR+1
3700        ADC #0
3710        STA SYMBOL+1
3720        LDA #SYMBOL
3730        STA STPNTR
3740        LDA /SYMBOL
3750        STA STPNTR+1
3760        LDA LINNUM   PUT LINE NUMBER INTO SYMBOL
3770        STA SYMBOL+3
3780        LDA LINNUM+1
3790        STA SYMBOL+2
3800        JMP CHAIN.SEARCH
3810 *---------------------------------
3820 PRINT.REPORT
3830        LDA #'A      START WITH A'S
3840 .1     STA VARNAM
3850        SEC
3860        SBC #'A      CONVERT TO HSHTBL INDEX
3870        ASL
3880        TAY
3890        LDA HSHTBL+1,Y
3900        BEQ .2       NO ENTRY FOR THIS LETTER
3910        STA PNTR+1
3920        LDA HSHTBL,Y
3930        STA PNTR
3940        JSR PRINT.LETTER.CHAIN
3950 .2     INC VARNAM   NEXT LETTER
3960        LDA VARNAM
3970        CMP #'Z+1
3980        BCC .1       STILL MORE LETTERS
3990        RTS          FINISHED
4000 *---------------------------------
4010 LTRDIG
4020        CMP #'0      DIGIT?
4030        BCC LD1      NO
4040        CMP #'9+1
4050        BCC LD2      YES
4060 LETTER
4070        CMP #'A      LETTER?
```

```
4080        BCC LD1      NO
4090        CMP #'Z+1
4100        BCC LD2      YES
4110        CLC          NO
4120 LD1    RTS
4130 LD2    SEC
4140        RTS
4150 *---------------------------------
4160 PRINT.LETTER.CHAIN
4170 .1     LDA VARNAM   FIRST LETTER
4180        JSR PRINT.CHAR
4190        LDY #1
4200 .2     INY
4210        LDA (PNTR),Y REST OF NAME
4220        AND #$7F
4230        CMP #'       BLANK?
4240        BEQ .3
4250        JSR PRINT.CHAR
4260 .3     CPY #3
4270        BCC .2
4280        LDA (PNTR),Y CHECK IF ARRAY
4290        BPL .4
4300        LDA #'(
4310        JSR PRINT.CHAR
4320 .4     CLC          POINT AT LINE # CHAIN
4330        LDA PNTR
4340        ADC #4
4350        STA TPTR
4360        LDA PNTR+1
4370        ADC #0
4380        STA TPTR+1
4390        JSR PRINT.LINNUM.CHAIN
4400        JSR MON.CROUT
4410        LDY #1
4420        LDA (PNTR),Y POINTER TO NEXT VARIABLE
4430        BEQ .5       NO MORE
4440        PHA
4450        DEY
4460        LDA (PNTR),Y
4470        STA PNTR
4480        PLA
4490        STA PNTR+1
4500        BNE .1       ...ALWAYS
4510 .5     RTS
4520 *---------------------------------
4530 PRINT.LINNUM.CHAIN
4540 .1     JSR TAB.NEXT.COLUMN
4550        LDY #2       POINT AT LINE #
4560        LDA (TPTR),Y
4570        STA LINNUM+1
4580        INY
4590        LDA (TPTR),Y
4600        STA LINNUM
4610        JSR PRINT.LINE.NUMBER
```

```
4620        LDY #1       SET UP NEXT POINTER
4630        LDA (TPTR),Y
4640        BEQ .2
4650        PHA
4660        DEY
4670        LDA (TPTR),Y
4680        STA TPTR
4690        PLA
4700        STA TPTR+1
4710        BNE .1       ...ALWAYS
4720  .2    RTS
4730  *---------------------------------
4740  TAB.NEW.LINE
4750        JSR MON.CROUT
4760  TAB.NEXT.COLUMN
4770  .1    LDA #7       FIRST TAB STOP
4780  .2    CMP MON.CH   CURSOR POSITION
4790        BCS .3       PERFORM TAB
4800        ADC #6       NEXT TAB STOP
4810        CMP #33      END OF LINE?
4820        BCC .2
4830        BCS TAB.NEW.LINE  ...ALWAYS
4840  .3    BEQ .4       ALREADY THERE
4850        SBC MON.CH   CALCULATE # OF BLANKS
4860        TAX
4870        JSR MON.PRBL2
4880  .4    RTS
4890  *---------------------------------
4900  PRINT.LINE.NUMBER
4910        LDX #4       PRINT 5 DIGITS
4920        STX LZFLAG   TURN ON LEADING ZERO FLAG
4930  .1    LDA #'0      DIGIT=0
4940  .2    PHA
4950        SEC
4960        LDA LINNUM
4970        SBC PLNTBL,X
4980        PHA
4990        LDA LINNUM+1
5000        SBC PLNTBH,X
5010        BCC .3       LESS THAN DIVISOR
5020        STA LINNUM+1
5030        PLA
5040        STA LINNUM
5050        PLA
5060        ADC #0       INCREMENT DIGIT
5070        BNE .2       ...ALWAYS
5080  .3    PLA
5090        PLA
5100        CMP #'0
5110        BEQ .5       ZERO, MIGHT BE LEADING
5120        SEC          TURN OFF LZFLAG
5130        ROR LZFLAG
5140  .4    JSR PRINT.CHAR
5150        DEX
```

```
5160        BPL .1
5170        RTS
5180  .5     BIT LZFLAG   LEADING ZERO FLAG
5190        BMI .4       NO
5200        LDA #'       BLANK
5210        BNE .4       ...ALWAYS
5220  PLNTBL .DA #1
5230        .DA #10
5240        .DA #100
5250        .DA #1000
5260        .DA #10000
5270  PLNTBH .DA /1
5280        .DA /10
5290        .DA /100
5300        .DA /1000
5310        .DA /10000
5320  *---------------------------------
5330  PRINT.CHAR
5340        ORA #$80
5350        JSR MON.COUT
5360        RTS
5370  *---------------------------------
5380  ZZ.END .EQ *
5390  ZZ.SIZ .EQ ZZ.END-ZZ.BEG
```

==============================================================================
DOCUMENT :AAL-8012:Articles:BlockMoveCopy.txt
==============================================================================

Block MOVE and COPY for Version 4.0

How many times have you wished there was an easy way to move a bunch of lines of your source program to some other place? I know it happens to me, and I frequently wish the assembler had this capability. Now, at last, it is possible. I no longer have to use DELETE, SAVE, HIDE, MERGE, LOAD in a very complicated sequence just to move that 20 line subroutine from the middle to the end of my source program!

The program as written assumes you have set up the USR command vector to jump to $800. You do this by stuffing a 0 into $1007 and an 8 into $1008 (type $1007:00 08 as a command). Then if you type, for example, "USR 1100,1190,1800", a copy of lines 1100 through 1190 will be inserted before line 1800. A word of caution: the lines in their new location will still have the old line numbers, until you RENUMBER. You can LIST, SAVE, and LOAD while the lines are out of sequence like this, but beware of doing any further editing! First, use the USR command to make the new copy of the lines; second, RENUMBER the program; third, DELETE the lines form their old location. Voila! You have moved them.

I just know someone (maybe everyone) is going to think that I should have made this program do its own renumbering. The reason I am confident of this is that I feel the same way. But the program as it stands is useful, and I will refine it later. My plan is to add one more parameter which specifies the increment for the line numbers in their new location. Then let the third parameter be the line number for the first line of the block being copied. The program will check whether making the copy will clobber any existing lines, and error out if so. If not, the copy will be made with its new line numbers. Then a question will be asked of the form" DO YOU WISH TO DELETE THE OLD LINES? (Y/N)". But for now, I will live with the more tedious but still very useful version you see here.

I would suggest that you put the object code of this program on a binary file, and then create an EXEC text file that contains the patch line to set up the USR command and a BLOAD command for the COPY program. The quarterly AAL diskette contains just such a file.

Now let me describe how the COPY program works. Notice that lines 1000-1060 are a summary of the operating syntax. Line 1070, together with lines 2390 and 2400, make the last three symbols in the symbol table listing tell me the start, end, and length of the object code. These are very useful for writing the object code out to a binary file. (Of course, I could use the .TF directive and write it automatically.)

Lines 1090-1220 define the page-zero locations the program uses. SS, SE, SL, and NEWPP are peculiar to this program; the rest of them are used by the monitor and the assembler. PP points to the beginning of the first source line in memory, and LOMEM is the lowest PP can go. A0, A1, A2, and A4 are used to pass addresses to the Apple Monitor Memory Move subroutine.

Linew 1240-1280 define some addresses of routines inside the S-C ASSEMBLER II Version 4.0. SYNX is the Syntax Error routine. You will get a syntax error message if you type in less than three parameters with the USR command, if the first two parameters are backwards or the same, if the block specified to be copied is empty, or if the target location is inside the block to be copied. MFER is the routine to print MEM FULL ERR, and you will get this error message if there is not room to make a copy; that is, the space between PP and LOMEM is less than the size of the block you want to copy.

SCND is the assembler routine to scan an input line from the current position and look for a decimal number. If it finds a decimal number, it will convert the number to binary and store it in A2L and A2H. As explained on page 10 of the Upgrade manual for Version 4.0, the first two parameters will have already been stored in A0 and A1.

SERTXT is the assembler routine to find a line in your source program, given the line number. It is called with the X-register containing the address of the first byte in page-zero of the byte-pair containing the line number you are looking for. When SERTXT is finished, $E4,E5 points at the first byte of the line found, and $E6,E7 points at the first byte of the next line. (Of course, if your line number could not be found, both pointers will point at the next larger line.)

MON.MOVE is a program inside the Apple Monitor ROM. It will copy a block of memory whose first byte address is in A1, last byte address in A2, to a new place in memory starting at the byte address in A4. This is the routine used when you use the monitor "M" command. It works fine as long as the target is not inside the source block.

Now to the COPY program itself. Briefly, the three parameters are checked for presence and consistency, and pointers are set up defining the area to be copied. A new value of PP is computed based on the length of this block, and I check to see if there is room in memory. Next I search for the target location, and check to make sure it is not inside the source block. (We don't wat any infinite loops!) If the target is higher in memory than the source block I adjust the source block pointers by subtracting the block length from them. Then I move all source lines below the insertion point down in memory far enough to make a hole in the text into which the source block can be copied. Finally, I copy in the source block, and return.

Some final comments... The COPY program is very fast, so play with it a little on a scratch program to convince yourself it is working. If you don't want to type in the source, you can just enter the hex codes from the monitor, and BSAVE it. Or, your can order the Quarterly AAL diskette, which will have the source, object, and a textfile to EXEC for BLOADing and patching the USR vector. Or, if you are very patient, you can wait till next August for Version 5.0 of the S-C ASSEMBLER II!

================================================================
DOCUMENT :AAL-8012:Articles:Compare.16Bits.txt
================================================================

Handling 16-bit Comparisons

It can be confusing enough in the 6502 to compare two single-byte values. Trying to remember that BCC means "branch if less than" (assuming that the values were considered to be unsigned values from 0-255), and that BCS means "branch if greater than or equal to" is enough to saturate my memory banks. I finally made a note on a card and tacked it up over my computer. Of course, if the values are considered to be signed values, in the range of -128 through +127, the problem is compounded, to say the least.

But what about comparing two values of two-bytes each? Like comparing two address pointers, for instance? A last resort would be to subtract one from the other, in two-byte arithmetic, and then compare the difference to zero. At least that would be understandable! But let's try to do it a little better than that. There is an example of this kind of comparison in lines 1310 through 1350 of the PRETTY.LIST program elsewhere in this issue of the Apple Assembly Line. Here is the segment:

```
    1310 .1    LDA SCRP
    1320       CMP HIMEM
    1330       LDA SRCP+1
    1340       SBC HIMEM+1
    1350       BCS .2
```

The object is to determine whether the value in PP,PP+1 is still less than the value in HIMEM,HIMEM+1 or not. The low-order byte of each value is stored in the first byte of each byte-pair, and the high-order byte is stored in the second byte. If all we needed to compare was the low-order bytes, we could do it with lines 1310 and 1320 above. Carry would be cleared by the CMP instruction if (SCRP) was less than (HIMEM). (I have just started using "(" and ")" to mean "the value stored in".)

Now let's use that carry bit and continue the comparison by actually subtracting the two high-order bytes. If the result of the subtraction leaves carry clear, we know that (SCRP) is indeed less tha (HIMEM), all 16 bits of it.

If you need to extend this to more than two bytes per value, you may. Just insert a pair of LDA-SBC instructions for each extra byte of precision, before the BCS instruction.

For another example of this kind of comparison, you might look up the NXTA1 routine in the Apple Monitor listing, at $FCBA. This routine is used by the Monitor MOVE command, and several other routines.

```
==============================================================================
DOCUMENT :AAL-8012:Articles:Front.Page.txt
==============================================================================
```

As I write thes, there are 85 paid subscribers! I sent out about 140 flyers in the last two weeks, so maybe the number will double again next month!  Pass the word to your friends and local Apple clubs ... and let me know how you like the content, style, et cetera.

In this issue...



Quarterly Disk #1

If you find there just isn't enough time to type in all the source programs in the Apple Assembly Line, I will be happy to save you the trouble.  Every three months I will put together a "Disk of the Quarter" which contains all the source in the format of the S-C ASSEMBLER II Version 4.0.  The price is only $15, and I will pay the postage.

The first such disk is ready now, covering October, November, and December of 1980.  The disks and the programs are for subscribers only.  Save your fingers, get yours now!

Help for Beginners

I will write some beginner's material from time to time for this newsletter, but I cannot cover every base at once.  Meanwhile, many of the magazines and club newsletters are beginning to publish articles for beginners who want to learn assembly language.  One of the best and most accessible is Creative Computing.  Chuck Carpenter's "Apple-Cart", a monthly feature, in the November, 1980 issue, was great!  He actually began the subject of machine language in the October issue, but in the November one he covered indexing, indirect addressing, and interrupts.  By the way, Chuck is also a subscriber to the Apple Assembly Line.

There have also been some good beginner articles in recent copies of Nibble and Softalk.  Nibble has been printing a lot of assembly language programs, which are good to study.

```
==============================================================================
DOCUMENT :AAL-8012:Articles:IBas.Prty.List.txt
==============================================================================
```

Integer BASIC Pretty Lister

About 2 1/2 years ago, Mike Laumer, of Carrollton, Texas, wrote a program to make pretty listing of Integer BASIC programs. He gave me a copy to look at, and then we both forgot about it. A few days ago I found it again, dusted it off, typed it in, and tried it out. After a little debugging, here is the result.

Which is neater?

```
     100 FOR I=1 TO 40: A(I)=I: A(I+41)=I*I: NEXT I

 or? 100 FOR I=1 TO 40
         : A(I)=I
         : A(I+41)=I*I
         : NEXT I
```

Mike and I happen to like the latter format, especially for printing in newsletters. It is a lot easier to read. And why print it if no one is going to read it?

If you are in Integer BASIC, and you have a program in memory ready to list, here are the steps to get a "pretty listing".

```
     1.  BLOAD B.PRETTY.LISTER
     2.  POKE 0,40    (or whatever number of characters
     3.  CALL 2048          per line you wish it to use)
```

If you want it to print on your printer, be sure to turn it on in the way you usually do before the CALL 2048. For example, if you have a standard Apple interface in slot 1, type "PR#1" just before the CALL 2048.

If you check it out, you will find a lot of similarity between the code in this program and what is stored in the Integer BASIC ROMs around locations $E00C through $E0F9. The routines are not in the same order, and there are a few significant changes to make the listing "pretty" and to control the line length. As I was typing in Mike's program, I took the liberty of "modularizing" it a little more, so that I could understand it. the PRINT.DECIMAL routine in lines 2500-2810 is almost identical to the one at $E51B in the BASIC ROMs. The changes are for the purpose of counting the number digits actually printed; this allows a closer control over line length.

Since one of the promised features of the Apple Assembly Line was commented disassemblies of some of the Apple's ROM code, I will try to explain how PRETTY.LIST works in some detail, module by module. You can then apply my explanation to the code which resides in ROM at $E00C-$E0F9.

PRETTY.LIST: This module is the overall control for the listing process. Since PP points to the beginning of the BASIC source program, lines 1270-1300 transfer this pointer into SRCP. Then SRCP is compared with HIMEM, to see if we are finished listing. The check is made before even listing one line, because it is possible that there is no source program to list! If the value in SRCP is greater than or equal to the value in HIMEM, then the listing is finished, and PRETTY.LIST returns to BASIC by JMP to DOS.REENTRY ($3D0). If the listing is not finished, I call PRINT.ONE.LINE to format and print out one line of the source program. "One line" may be several statements separated by colons. Then I jump back to the test to see if we are through yet, and so on and on and on.

PRINT.ONE.LINE: A source line in Integer BASIC is encoded in token form, and this routine has to convert it back to the original form to list it. First, let's look at how a coded line is laid out.

```
      #     line
     bytes number     body of source line      01
```

The first byte of a line is the line length; we will ignore it in this program, because we do not need it. The last byte of each line is the hex value $01, which is the token for end-of-line. That is all we need to signal the end of a line, and the start of another one. The second and third bytes of each line are the line number, in binary, with the low byte first. The body of the line is made up of a combination of tokens and ASCII characters.

For the most part, tokens have a hex value less than $80, while the ASCII characters have a hex value greater than $80. One important exception is the token for a decimal constant. These are flagged by a pseudo-token consisting of the first digit of the constant in ASCII (hex $B0 through $B9); after the token, two bytes follow which contain the binary form of the constant with the low byte first. For example, the decimal constant 1234 would be stored in three bytes as: $B1 D2 04.

The task of PRINT.ONE.LINE is to scan through the coded form of a line, printing each ASCII character, and converting each token to its printing form. In addition, the routine must count line position as it goes, so that a new line can be started when one fills up. Furthermore, we want it to start a new line whenever the ":" indicates a new statement has begun within a line. We have to look out for REM statements and quoted strings, because the ":" might appear in them without signalling a new statement.

Lines 1400-1460 start the ball rolling. The line position is set to zero, and the fill flag for the PRINT.DECIMAL routine is set to produce a right-justified-blank-filled number. Then GET.NEXT.BYTE is called to advance the SCRP past the byte count in the first byte of the line. GET.NEXT.BYTE returns the value of the byte in A, and with Y=0. This time we ignore the value in A, and use the fact that Y=0 to clear A.

Lines 1470-1510 pick up the two bytes of the line number and call PRINT.DECIMAL to print it out. These same lines will be used later to print out any constants which are in the line. These lines are entered this time with A=0 and with IB.FILL set for the RJBF mode (right-justified-blank-filled). Later for constants they will be entered with IB.FILL set for printing with no leading blanks, and with A <> 0. The value in A is used to set IB.FLAG, which determines whether a trailing blank will be printed. One will be printed after the line number, but not after a constant inside a line. (For a character that uses so little ink, blanks can sure eat up a lot of code!)

At line 1520 the main body of the PRINT.ONE.LINE routine begins. CHECK.EOL.GET.NEXT.BYTE decides whether we are getting too close to the end of the line. This prevents splitting token-words in half, with a few characters dangling off the end of one line, and the rest starting a new one. (At least, on the screen it would look like that; on a printer it might just print out into a margin.) The routine will start a new line before returning if the end is too near. When it finally does return, the next byte will be in A, and Y will be zero. If the next byte is a token (less than $80), control branches to line 1720. If the first bit of the byte is 1, and the second bit is 0, the code at lines 1550-1580 assumes the pseudo-token for a constant has appeared. If the second bit is also 1, the byte is an ASCII character. Before printing the character, lines 1590-1630 may print a blank. This would be a trailing blank after printing a token or a line number. The character is then printed at lines 1640-1650, and another end-of-line check is made. This time "too near the end" is defined as within 3 spaces. The next byte must either be a token or yet another ASCII character, so a determination is made in lines 1660-1700.

Tokens are harder to handle, because we have to test for several special cases, and if not a special case the token table must be searched to find the token's name. Lines 1720-1740 test for the end-of-line token; if this is it, a carriage return is printed and PRINT.ONE.LINE returns back to its caller.

If the token is the new-statement-token, used for ":", a new line is started. Then the fun begins: we have to search the token table. This table is the most recondite portion of the whole Apple computer! I have only scratched its surface. The table is located between $EC00 and $EDFF, but it is not in that order. It goes like this: first $ED00, then $EDFF-$ED01 (yes, backwards!), then $EC00, then $ECFF-$EC01. The names for all the tokens are stored in the table, along with various bits of information about precedence and syntax. If you print out the table, you will not see any names... Steve Wozniak subtracted $20 from each byte before putting it into the table. Well, there is a lot more to it than that, but I am getting lost, side-tracked.

After finding the token's name string inside the token table, we have to print it out. This is done in lines 1840-1940. The name is terminated either by the last character having a value greater than $BF, or by the next character in the table having a value less than $80. The routine at $E00C decides whether or not to print a trailing blank, I think.

After printing the token's name, lines 1960-2010 test for REM or a quoted string. Either of these would be followed by a bunch of ASCII characters terminated by a token, so control branches to line 1660 to print them out. If neither, we go back to line 1520, to get the next token, or whatever.

Somehow I skipped over line 1830. I believe the JSR $EFF8 determines whether or not to print a space in front of the token name.

FIND.TOKEN: Lines 2040-2110 set up a pointer to the half of the token table which contains the name string for the token we want. Tokens $00 through $50 are in the first half, and $51 through $7F are in the second half.

Lines 2120-2250 scan through the table, counting token names as they are passed. When the nth one is found, where n is the token value, the routine returns. It returns with A=0, and Y = offset in the half of the token table we have been scanning.

CHECK.EOL.GET.NEXT.BYTE: Enter this routine with A containing the number of bytes short of the end of the line you want to test for, as a negative number. If too near the end, CR.7.BLANKS will be called to start a new line. In any case the routine exits by transferring to GET.NEXT.BYTE to get the next byte from the source line.

CR.7.BLANKS: Prints a carriage return adn 7 blanks to start a new line.

CHAR.OUT: Simply counts characters and then calls on the Apple monitor to print out a character. We need to count columns for CHECK.EOL.GET.NEXT.BYTE.

PRINT.DECIMAL: Lifted out of Integer BAIC from $E51B, and modified to eliminate the ability to store the converted number in the input buffer, and to add the ability to count output characters.

Additions to this program: You might like to add some more featrures to this program. For example, it would be nice to have it request the line length and printer slot number itself, and turn the printer on and off. Also, it would be helpful to add indentation for FOR...NEXT loops and IF...THEN statements. The same program could be merged with a cross reference program to build and print a variable and line number cross reference.

If you decide to try any of these, or any other enhancements, why not write them up and send them to me for publication?

```
===========================================================================
DOCUMENT :AAL-8012:Articles:Listed.Xprsns.txt
===========================================================================
```

Allow List of Expressions with .DA Directive

Some customers have said they wished the .DA directive in the S-C ASSEMBLER II allowed more than one expression per line. For example, ".DA 1000,100,10,1" would then produce 8 bytes of code just as though there were four separate .DA lines. (Once and a while I wish it worked this way too!)

The following little patch will transform your .DA in just that way. Because of the .OR and .TF directives, assembling these 42 lines will produce two binary files that are ready to BLOAD. When you BLOAD them, the copy of the assembler in memory will be patched. You can then BSAVE the assembler (use a different name!), and you have the new capability.

If you do not have Version 4.0 of the assembler, then this patch will not work. If you have one of the very earliest copies of Version 4.0, it may have some different addresses. Check it out before you type in the code: at $20D4 you should find three JMP instructions, as indicated in the comments here in lines 1210 through 1230. If you find those JMPs, go right ahead and make the patches. Of course, if you have already added some code at $24B0, then you will have to put this patch somewhere else.

If you do not find those JMP instructions at $20D4, but you do find them at $20B1, then you need to change a few addresses in the patch code. Change the following lines as indicated:

```
     1170 PSDA   .EQ $2092
     1190        .OR $20B1
```

==============================================================================
DOCUMENT :AAL-8012:Articles:PrinterOnError.txt
==============================================================================

Keeping Printer On After Error Message

One customer wanted this, and maybe you would too. He needed the printer to stay enabled even if an editor or assembler error message was generated. S-C ASSEMBLER II Version 4.0 shuts off any printer after any error occurs, so he couldn't get his printer to stay on long enough to get a listing.

Here is a patch that will leave a printer "hooked in".

```
    :$1756:F0 24   (address of patch area)
    :$24F0:A9 FF 85 D9 20 80 1F 4C 26 10
```

After making the patch, you can BSAVE using A$1000,L$14FB.

The patch is put at $24F0; if you have already put some other patch there, be sure to put this one somewhere else! Be sure you TEST it before you clobber or delete the original! Be sure you really WANT it before you even bother to type it in!

==============================================================================
DOCUMENT :AAL-8012:Articles:Smart.Disasms.txt
==============================================================================

Intelligent Disassemblers

Not one, but two!  In this issue of AAL you find two ads for intelligent disassemblers.  Dr. Robert F. Zant, of Decision Systems, and Bob Kovacs, of RAK-WARE, have each written one.  After all these years, two of them pop up in the same week!

Dr. Zant's reads a binary file and writes a text file which can be EXECed into either the S-C ASSEMBLER II Version 4.0 or the Apple assembler from the DOS Tool Kit.  He writes an intermediate text file during pass one of the disassembly, and then reads it back in, formats it for the desired assembler, and writes it back out.  His disassembler is a combination of machine language code and Applesoft code; you have to have Applesoft in ROM and at least 32K RAM.  He includes a couple of handy utility programs on the diskette.

Bob Kovac's disassembler works from a binary program already in memory.  Both passes are performed in memory, and then the text file is written.  Since everything is done in memory, it is very fast.  The resulting text file is EXECed into the S-C ASSEMBLER II Version 4.0.

Both disassemblers create labels for all branch addresses inside the block being disassembled.  Bob Kovac's version also makes labels for all external branch addresses, putting .EQ lines at the beginning to define them.  The RAK-WARE version also make symbols for all page-zero references.  They also are set up with .EQ lines at the beginning of the text file.

Both disassemblers output a control-I at the beginning of each line rather than a line number.  This causes the assembler to generate its own line number when the file is EXECed, and allows you to set your own increment and starting line number just before typing the EXEC command.  Set the increment by using the INC command; and set the starting line number by typing the number you want less the increment, followed by a space and return.

I forgot to mention, Bob Kovac's disassembler works with eihter Integer BASIC or Applesoft.  He has driver programs written in both languages on the diskette.

They both are excellent tools, which have long been needed.  They both cost the same, $25.  What can I say?  Buy them both!  Do it before the end of 1980, and get a tax deduction before Reagan and our new Congress lower the incode tax rate!

Advertising in AAL

For the first time, there are some ads in your newsletter.  I think you will find them almost as useful as the non-ad material, because so many of you have asked me for compatible two-pass disassemblers to go along with the S-C ASSEMBLER. If you have written some programs that your want to sell, which you think other readers of the Apple Assembly Line would be interested in, you can advertise here, too. The cost is quite low ... $20 for a full page, $10 for 1/2 page.

================================================================================
DOCUMENT :AAL-8012:DOS3.3:B.COPY.LINES.txt
================================================================================

( DTC removed -- lots of garbage characters )

```
=============================================================================
DOCUMENT :AAL-8012:DOS3.3:MkCopyLinesFile.txt
=============================================================================
```

```
∑MAKE TEXT FILE1∑TO SET UP "COPY.LINES"=-D$»‰(4)\(∑D$"OPEN SETUP COPY
LINES"|2∑D$"WRITE SETUP COPY LINES"£<∑"$1007:00 08"∑F∑"BLOAD
B.COPY.LINES"∑P∑D$"CLOSE"
```

```
===================================================================
DOCUMENT :AAL-8012:DOS3.3:S.COPY.LINES.txt
===================================================================

 1000  *---------------------------------
 1010  *      COPY L1,L2,L3
 1020  *          L1 = FIRST LINE OF RANGE TO COPY
 1030  *          L2 = LAST LINE OF RANGE TO COPY
 1040  *          L3 = LINE NUMBER BEFORE WHICH TO INSERT
 1050  *                   THE COPIED LINES
 1060  *---------------------------------
 1070  ZZ.BGN .EQ *
 1080  *---------------------------------
 1090  SS     .EQ $00,01   START OF SOURCE BLOCK
 1100  SE     .EQ $02,03   END OF SOURCE BLOCK
 1110  SL     .EQ $04,05   LENGTH OF SOURCE BLOCK
 1120  NEWPP  .EQ $06,07   NEW PROGRAM POINTER
 1130  A0L    .EQ $3A
 1140  A0H    .EQ $3B
 1150  A1L    .EQ $3C
 1160  A1H    .EQ $3D
 1170  A2L    .EQ $3E
 1180  A2H    .EQ $3F
 1190  A4L    .EQ $42
 1200  A4H    .EQ $43
 1210  LOMEM  .EQ $4A,4B
 1220  PP     .EQ $CA,CB
 1230  *---------------------------------
 1240  SYNX   .EQ $105E
 1250  MFER   .EQ $1128
 1260  SCND   .EQ $112D
 1270  SERTXT .EQ $14F6
 1280  MON.MOVE   .EQ $FE2C
 1290  *---------------------------------
 1300         JMP COPY
 1310  *---------------------------------
 1320  ERR1   JMP SYNX
 1330  ERR2   .EQ ERR1
 1340  ERR3   JMP MFER
 1350  ERR4   .EQ ERR1
 1360  *---------------------------------
 1370  COPY
 1380         JSR SCND     GET THIRD PARAMETER
 1390         CPX #6       BE SURE WE GOT THREE
 1400         BCC ERR1     NOT ENOUGH PARAMETERS
 1410         LDX #A0L     FIND BEGINNING OF SOURCE
 1420         JSR SERTXT
 1430         LDA $E4      SAVE POINTER
 1440         STA SS
 1450         LDA $E5
 1460         STA SS+1
 1470         LDX #A1L     FIND END OF SOURCE BLOCK
 1480         JSR SERTXT
```

```
1490        SEC          SAVE POINTER AND COMPUTE LENGTH
1500        LDA $E6
1510        STA SE
1520        SBC SS
1530        STA SL       SOURCE LENGTH
1540        LDA $E7
1550        STA SE+1
1560        SBC SS+1
1570        STA SL+1
1580        BCC ERR2     RANGE BACKWARD
1590        BNE .4
1600        LDA SL
1610        BEQ ERR2     NOTHING TO MOVE
1620 *---------------------------------
1630 .4     LDA PP       COMPUTE NEW PP POINTER
1640        SBC SL
1650        STA NEWPP
1660        LDA PP+1
1670        SBC SL+1
1680        STA NEWPP+1
1690 *---------------------------------
1700        LDA NEWPP    SEE IF ROOM FOR THIS
1710        CMP LOMEM
1720        LDA NEWPP+1
1730        SBC LOMEM+1
1740        BCC ERR3     MEM FULL ERR
1750 *---------------------------------
1760        LDX #A2L     FIND TARGET LOCATION
1770        JSR SERTXT
1780        LDA SS       BE SURE NOT INSIDE SOURCE BLOCK
1790        CMP $E4
1800        LDA SS+1
1810        SBC $E5
1820        BCS .1       BELOW SOURCE BLOCK
1830        LDA $E4
1840        CMP SE
1850        LDA $E5
1860        SBC SE+1
1870        BCC ERR4     INSIDE SOURCE BLOCK
1880 *   TARGET IS ABOVE SOURCE BLOCK, SO WE HAVE TO
1890 *   ADJUST SOURCE BLOCK POINTERS.
1900        SEC
1910        LDA SS
1920        SBC SL
1930        STA SS
1940        LDA SS+1
1950        SBC SL+1
1960        STA SS+1
1970        SEC
1980        LDA SE
1990        SBC SL
2000        STA SE
2010        LDA SE+1
2020        SBC SL+1
```

```
2030        STA SE+1
2040  *--------------------------------
2050  .1    LDA PP       SET UP MOVE TO MAKE HOLE
2060        STA A1L
2070        LDA PP+1
2080        STA A1H
2090        LDA NEWPP
2100        STA PP
2110        STA A4L
2120        LDA NEWPP+1
2130        STA PP+1
2140        STA A4H
2150        LDA $E5
2160        STA A2H
2170        LDA $E4
2180        STA A2L
2190        BNE .2
2200        DEC A2H
2210  .2    DEC A2L
2220        LDY #0
2230        JSR MON.MOVE
2240  *--------------------------------
2250        LDA SS       MOVE IN SOURCE BLOCK
2260        STA A1L
2270        LDA SS+1
2280        STA A1H
2290        LDA SE+1
2300        STA A2H
2310        LDA SE
2320        STA A2L
2330        BNE .3
2340        DEC A2H
2350  .3    DEC A2L
2360        JSR MON.MOVE
2370        RTS
2380  *--------------------------------
2390  ZZ.END .EQ *-1
2400  ZZ.SIZ .EQ ZZ.END-ZZ.BGN+1
```

```
==============================================================================
DOCUMENT :AAL-8012:DOS3.3:S.IB.Ptry.Lstr.txt
==============================================================================

 1000         .TF B.PRETTY.LISTER
 1010         .LIST OFF
 1020  *---------------------------------
 1030  *      INTEGER BASIC PRETTY-LIST
 1040  *---------------------------------
 1050  LINE.LENGTH         .EQ $00
 1060  LINE.POSITION       .EQ $01
 1070  MON.CH              .EQ $24
 1080  PP     .EQ $CA,CB
 1090  HIMEM  .EQ $4C,4D
 1100  SRCP   .EQ $E2,E3
 1110  TKNP   .EQ $CE,CF
 1120  IB.FLAG    .EQ $EA
 1130  IB.FILL    .EQ $FA
 1140  *---------------------------------
 1150  DOS.REENTRY         .EQ $3D0
 1160  GET.NEXT.BYTE       .EQ $E02A
 1170  TOKEN.TABLE         .EQ $ED00
 1180  MON.COUT            .EQ $FDED
 1190  MON.CROUT           .EQ $FD8E
 1200  *---------------------------------
 1210  TOKEN.EOL           .EQ $01
 1220  TOKEN.COLON         .EQ $03
 1230  TOKEN.REM           .EQ $5D
 1240  TOKEN.QUOTE         .EQ $28
 1250  *---------------------------------
 1260  PRETTY.LIST
 1270         LDA PP
 1280         STA SRCP
 1290         LDA PP+1
 1300         STA SRCP+1
 1310  .1     LDA SRCP     SEE IF AT END
 1320         CMP HIMEM
 1330         LDA SRCP+1
 1340         SBC HIMEM+1
 1350         BCS .2       FINISHED
 1360         JSR PRINT.ONE.LINE
 1370         JMP .1
 1380  .2     JMP DOS.REENTRY
 1390  *---------------------------------
 1400  PRINT.ONE.LINE
 1410         LDA #0
 1420         STA LINE.POSITION
 1430         LDA #$A0     SET UP PRINT.DECIMAL FOR RJBF
 1440         STA IB.FILL
 1450         JSR GET.NEXT.BYTE SKIP OVER BYTE COUNT
 1460         TYA          (A)=0
 1470  .1     STA IB.FLAG
 1480         JSR GET.NEXT.BYTE GET LINE NUMBER
```

```
1490        TAX          LOW BYTE
1500        JSR GET.NEXT.BYTE HIGH BYTE
1510        JSR PRINT.DECIMAL  PRINT THE LINE NUMBER RJBF
1520  .2    LDA #-7      WITHIN 7 OF END OF LINE
1530        JSR CHECK.EOL.GET.NEXT.BYTE
1540        STY IB.FILL  CLEAR RJBF
1550        TAX          TEST BYTE AND SAVE IN X-REG
1560        BPL .6       TOKEN
1570        ASL
1580        BPL .1       CONSTANT, GO PRINT IT
1590        LDA IB.FLAG
1600        BNE .3       DO NOT NEED A BLANK
1610        LDA #$A0
1620        STA IB.FLAG
1630        JSR CHAR.OUT
1640  .3    TXA          RETRIEVE BYTE
1650  .4    JSR CHAR.OUT AND PRINT IT
1660  .5    LDA #-3      WITHIN 3 OF EOL
1670        JSR CHECK.EOL.GET.NEXT.BYTE
1680        TAX          TEST BYTE, SAVE IN X-REG
1690        BMI .4       NORMAL CHAR
1700        STA IB.FLAG
1710  *---------------------------------
1720  .6    CMP #TOKEN.EOL
1730        BNE .7       NOT END OF LINE
1740        JMP MON.CROUT   END OF LINE
1750  .7    CMP #TOKEN.COLON
1760        BNE .8
1770        JSR CR.7.BLANKS
1780        LDA #TOKEN.COLON
1790  .8    PHA          SAVE TOKEN
1800        JSR FIND.TOKEN
1810        BIT IB.FLAG
1820        BMI .9
1830        JSR $EFF8
1840  .9    LDA (TKNP),Y  GET CHAR OF TOKEN NAME
1850        BPL .10
1860        TAX          SAVE CHAR IN X
1870        AND #$3F
1880        STA IB.FLAG
1890        CLC
1900        ADC #$A0
1910        JSR CHAR.OUT
1920        DEY
1930        CPX #$C0
1940        BCC .9
1950  .10   JSR $E00C
1960        PLA          GET ORIGINAL UNMOLESTED TOKEN
1970        CMP #TOKEN.REM
1980        BEQ .5       REM
1990        CMP #TOKEN.QUOTE
2000        BEQ .5       QUOTATION
2010        BNE .2       NEITHER
2020  *---------------------------------
```

```
2030  FIND.TOKEN
2040         LDX #TOKEN.TABLE
2050         STX TKNP
2060         LDX /TOKEN.TABLE
2070         CMP #$51     SEE IF NEED OTHER HALF TOKEN.TABLE
2080         BCC .1       NO
2090         DEX          YES
2100         SBC #$50
2110  .1     STX TKNP+1
2120  .2     PHA          SAVE MODIFIED TOKEN ON STACK
2130         LDA (TKNP),Y  Y GOES 0,FF,FE,...
2140  .3     TAX
2150         DEY
2160         LDA (TKNP),Y LOOK FOR NEGATIVE BYTE
2170         BPL .3
2180         CPX #$C0     IF BYTE BEFORE NEGATIVE BYTE IS
2190         BCS .4       BTWN $C0 AND $FF, THEN
2200         CPX #$00     KEEP LOOKING
2210         BMI .3
2220  .4     TAX
2230         PLA
2240         SBC #1       DECREMENT TOKEN
2250         BNE .2       NOT THERE YET
2260         RTS
2270  *---------------------------------
2280  CHECK.EOL.GET.NEXT.BYTE
2290         CLC
2300         ADC LINE.LENGTH
2310         CMP LINE.POSITION
2320         BCS .1
2330         JSR CR.7.BLANKS
2340  .1     JMP GET.NEXT.BYTE
2350  *---------------------------------
2360  CR.7.BLANKS
2370         LDA #$8D
2380         LDY #7
2390         STY LINE.POSITION
2400  .1     JSR CHAR.OUT
2410         LDA #$A0
2420         DEY
2430         BNE .1
2440         RTS
2450  *---------------------------------
2460  CHAR.OUT
2470         INC LINE.POSITION
2480         JMP MON.COUT
2490  *---------------------------------
2500  PRINT.DECIMAL
2510         STA $F3
2520         STX $F2
2530         LDX #4
2540         STA $C9      LEADING ZERO FLAG
2550  .7     LDA #$B0
2560         STA $F9
```

```
2570  .1     LDA $F2
2580         CMP $E563,X
2590         LDA $F3
2600         SBC $E568,X
2610         BCC .2
2620         STA $F3
2630         LDA $F2
2640         SBC $E563,X
2650         STA $F2
2660         INC $F9
2670         BNE .1       ...ALWAYS
2680  .2     LDA $F9
2690         CPX #0       SEE IF LAST DIGIT
2700         BEQ .4       YES
2710         CMP #$B0     NO, SEE IF LEADING ZERO
2720         BEQ .3       MAYBE
2730         STA $C9      NO
2740  .3     BIT $C9      STILL PLUS IF LEADING ZERO
2750         BMI .4       NOT LEADING ZERO
2760         LDA IB.FILL  SEE IF BLANK FILL
2770         BEQ .5       NO
2780  .4     JSR CHAR.OUT PRINT CHAR
2790  .5     DEX
2800         BPL .7
2810         RTS
```

```
==============================================================================
DOCUMENT :AAL-8012:DOS3.3:S.PATCH.DA.txt
==============================================================================

 1000  *--------------------------------
 1010  *      PATCH FOR .DA WITH COMMA LIST
 1020  *--------------------------------
 1030  *
 1040  *      TO INSTALL THIS PATCH:
 1050  *
 1060  *      1.  BRUN ASMDISK 4.0
 1070  *      2.  BLOAD PATCH.DA.1
 1080  *      3.  BLOAD PATCH.DA.2
 1090  *      4.  BSAVE ASMDISK 4.1,A$1000,L$14FB
 1100  *
 1110  *--------------------------------
 1120  EXP.VALUE .EQ $DB
 1130  *--------------------------------
 1140  GNC    .EQ $128B
 1150  EMIT   .EQ $19FA
 1160  CMNT   .EQ $188E
 1170  PSDA   .EQ $20B5
 1180  *--------------------------------
 1190         .OR $20D4         REPLACES:
 1200         .TF PATCH.DA.1
 1210         JMP BOTH.BYTES    (JMP $19B2)
 1220         JMP LOW.BYTE      (JMP $194D)
 1230         JMP HIGH.BYTE     (JMP $19D7)
 1240  *--------------------------------
 1250         .OR $24B0    PATCH AREA
 1260         .TF PATCH.DA.2
 1270  BOTH.BYTES
 1280         LDA EXP.VALUE
 1290         JSR EMIT
 1300  HIGH.BYTE
 1310         LDA EXP.VALUE+1
 1320  ALL    JSR EMIT
 1330         JSR GNC
 1340         CMP #',      COMMA?
 1350         BEQ MORE
 1360         JMP CMNT     FINISHED
 1370  MORE   JMP PSDA
 1380  LOW.BYTE
 1390         LDA EXP.VALUE
 1400         CLC
 1410         BCC ALL      ...ALWAYS
```

==============================================================================
DOCUMENT :AAL-8012:DOS3.3:Setup.CopyLines.txt
==============================================================================

```
$1007:00 08
BLOAD B.COPY.LINES
```

==============================================================================
DOCUMENT :AAL-8101:Articles:Computed.Gosub.txt
==============================================================================

A Computed GOSUB for Applesoft

How many times I have wished for one!  I guess I am spoiled from FORTRAN and Apple Integer BASIC.  The Computed GOTO is also left out, but I saw that one written up in a recent newsletter.  The author said he didn't know how to do the Computed GOSUB, so here it is!

<<<<listing>>>>

Lines 1160 and 1170 check the token after the "&" to see if it is "GOSUB"; if not, you will get a big SYNTAX ERROR.  Lines 1180 and 1190 check the stack to see if there is room for another GOSUB entry; if not, you get an OUT OF MEMORY error.  Lines 1200-1290 push the data on the stack that will be needed to RETURN.  Lines 1300 and 1310 compute the value of whatever expression follows the &GOSUB, and turn it into an integer that looks just like a line number.  Finally, lines 1320 and 1330 simulate a normal GOTO.  That's all there is to it!

Here is a sample Appplesoft program using the new &GOSUB statement:

```
 10 POKE 1013,76: POKE 1014,0: POKE 1015,3
 20 INPUT X
 30 &GOSUB x*100
 40 GOTO 20

100 PRINT 100:RETURN
200 PRINT 200:RETURN
300 PRINT 300:RETURN
400 PRINT 400:RETURN
```

==============================================================================
DOCUMENT :AAL-8101:Articles:Copy.for.SCAsm.txt
==============================================================================

Putting COPY in S-C Assembler II

I just looked at the first AAL Disk of the Quarter. The first item of business was to incorporate the changes into my copy of the assembler.

The lower-case mod and the .DA mod went just as described in AAL. However, when it came to the COPY stuff, I found that I wasn't really happy to load it at $800 and hope it didn't get clobbered. Here's what I did....

I changed the origin of the COPY program to $25A0 (since I already have a special printer driver at $2500.259F). The COPY program runs from $25A0 through $266F, so I changed the symbol table origin by typing "$1011:27". This sets the bottom of the symbol table at $2700. I put a ".TF B.SC COPY MODS" line in, to write the object on a binary file.

After assembling, I BLOADed the file B.SC COPY MODS into memory. Then I could have plugged in the USR vector like Bob suggested, but I wanted a real "COPY" command. Therefore I searched around in the assembler until I found the command table. I put the letters "COP" and the program address over the top of the tape SAVE command entry, by typing "1246:43 4F 50 9F 25". I felt the loss of the tape SAVE command was worth it, to get a real COPY command.

Now the command "COPY 1000,1050,2500" will copy lines 1000 through 1150 into the pplace right before line 2500. The USR command is still intact and I'm ready for some more changes!

================================================================
DOCUMENT :AAL-8101:Articles:Edit.Cmd.SCASM.txt
================================================================

EDIT Command for S-C Assembler II................Mike Laumer

At last!  Owners of the S-C Assembler II Version 4.0 can now have the power of an EDIT command similar in function to the popular "Program Line Editor" (PLE) by Neil Knozen.  (PLE only works with INteger BASIC and Applesoft, although some wizards have figured out how to interface it with the S-C Assembler.)  The program presented here will patch itself into Version 4.0 to turn the "USR" command into an EDIT command.

Several weeks ago Bob Sander-Cederlof contacted me about some contract programming, to help out on various projects he had in mind.  So I suggested lunch, and we met to discuss some of his projects.  I was amazed at the list (as long as my arm!) of the the ideas for just one of his products, the S-C Assembler II.  (If you like version 3.2, as I did; if you are thrilled with version 4.0, as I am; then version 5.0 will ....)  So I picked out a couple that would be fairly straightforward and would let me pick up the internal structure of the assembler gradually.

After signing a non-disclosure agreement, I obtained the source files and made a listing of the assembler.  Lucky for me I have a brand new Epson MX-80 printer!  I think it is the greatest!

Thursday, I made the listing.  Friday I looked at the listing.  Friday night I began writing code for the EDIT command.  Saturday from 9AM till 1AM I wrote more code, read it through, and rewrote it.  Sunday morning I typed it into my Apple and eliminated the assembly errors (typos).  And by 11AM, with the exception of two trivial bugs, I had it working!  I nearly fell out of my chair!  A 377-line program worked on the first run!

After you type in the program, assemble it, and BRUN it, the USR command will work as an edit command.  If you type the command USR with no line number, it will do nothing.  If you type USR and one line number, it will list the line on the bottom of the screen and set yo up to edit it.  If you type USR and two line numbers, separated by a comma, all the lines in the range will be set up to edit, one at a time.

How to Use EDIT:  Twelve editing functions are available, and you may see fit to add some more.  Each function is selected by typing a control character.  If you type a normal character, it will write over the top of the characters already in the line.  The control characters and their associated functions are:

```
control-B  Move to beginning of line.
control-D  Delete character beneath cursor.
control-E  Move to end of line.
```

```
control-F  Find a character; the character searched for
           is typed after the control-F; repeatedly
           typing the same character will keep looking
           successive occurrences.
control-H  Backspace (left arrow).
control-I  Insert characters before current cursor
           position.
control-M  (RETURN)  Stop editing the line,
           and submit it to the line input routine
           in the assembler.
control-O  Same as control-I, except next character
           may be any control character.
control-Q  same as control-M, but line beyond cursor
           is truncated.
control-T  Skip to next tab stop.
control-U  (Right Arrow)  Move cursor forward.
control-X  Kill edit, does not submit line.
```

How EDIT Works: When you BRUN the file B.EDIT (after assembly has written the object code there!), the code in lines 1360-1530 is executed. This patches the USR command vector to jump to EDIT (line 1720), and makes some patches inside the assembler. The patches only work for version 4.0! Their purpose is to make the code which processes a source line into a subroutine.

Lines 1540-1620 are part of the patch code for the source line processing subroutine.

Lines 1720-2040 determine the number of line numbers typed, and search for them in the source program. Then E.LIST is called for each line to be edited.

Lines 2050-2360 list the source line on the screen and also stuff it into the line input buffer at $0200. All changes will be made in the buffer, not in the source program.

Lines 2370-2530 read a key from the keyboard and search the command table. If the key is found in the table, then DOIT is called to execute the command. If the key is not found, I assume it is a type-over character. The command table search is actually performed by a rather neat subroutine inside the assembler, called SEARCH.

Lines 2540-2690 process a type-over character, in which the key just typed replaces the character under the cursor. Then the modified line in the buffer is re-displayed on the screen.

Lines 2700-2750 position the cursor at the beginning of line 19 (on the screen), where the source line will be listed.

Lines 2760-2900 display the line from the buffer. Display always starts at line 19 on the screen. Control characters are shown in inverse video.

Lines 2910-4090 process the various commands. Each processor is written as a subroutine. The RTS returns to line 2520; at this point the Carry Status is used to flag whether or not to re-display the source line from the buffer.

Lines 4100-4260 read a character from the keyboard by calling on the monitor RDKEY subroutine. The internal line buffer index is also converted to cursor line and column position on the screen.

Lines 4270 through the end are the command table. The first line defines the entry size and key size for the SEARCH subroutine; 3 bytes per entry, with a one byte key at the fron of each entry. The remaining two bytes of each entry are the starting-address-minus-one of the command processor rotuine. A final $00 byte terminates the table.

WARNING! I have used the patch for Bob's assembler which allows a list of .DA items! Lines 4270-4420 require this patch to be installed. You can read about the patch in Apple Assembly Line for December, 1980, on page 9. If you have not installed the patch, then lines 4270-4420 need to be re-written with each .DA item on a separate source line.

Well, you better get typing on that Apple, I know this is one routine you can't wait to key in. I know I couldn't wait to create it! Or, if you CAN wait, you can get the source on the next Disk of the Quarter from Bob.

=============================================================================
DOCUMENT :AAL-8101:Articles:Front.Page.txt
=============================================================================

Volume 1 -- Issue 4                    January, 1981

There are, as of Christmas Eve, 179 of you subscribing to the Apple Assembly Line! Last month I wondered if circulation could double, from 85, but we did even better! Also, several stores have decided to carry the AAL for sale like a magazine. We are growing a lot faster than I predicted, and I like it!

In This Issue...



First "Disk of the Quarter"

Every three months I collect onto one disk all the source programs published in AAL for the quarter. QD#1 (for October, November, and December of 1980) is now available, for $15. You can save a lot of typing.

If you would like to help promote the newsletter, here is a nice offer: you sign up four new subscribers, and send me their mailing addresses and money, and I will send you a "Disk of the Quarter" FREE and POSTPAID!

Those Compatible Disassemblers

Bob Zant and Bob Kovacs both report that their new two-pass disassemblers are selling well. Well enough to warrant advertising again! Have you bought a copy yet?

TAB Locations in S-C Assembler II Version 4.0

For some reason, people are always asking me where the tab stops are kept, because they want to change them. The old version 3.2 manual gives the patch locations for the three tab stops, but they are different in version 4.0. You will find them at:

| | column | location |
|---|---|---|
| 1st tab | 14 | $140D:0B |
| 2nd tab | 18 | $1411:0F |
| 3rd tab | 27 | $1402:18 |

Note that the value stored in memory is three less than the column number.

```
==============================================================================
DOCUMENT :AAL-8101:Articles:How.Move.Mem.txt
==============================================================================
```

How to Move Memory

One of the most common problems in assembly language programming is the problem of moving data from one place in memory to another.

Moving Little Blocks: If you only need to move one or two bytes of data from one place to another in memory, it is easy. You might do it like this:

```
       LDA SOURCE
       STA DEST
       LDA SOURCE+1
       STA DEST+1
```

Or, if the A-register was busy but X and Y were not, you might write:

```
       LDX SOURCE
       LDY SOURCE+1
       STX DEST
       STY DEST+1
```

If you know ahead of time exactly how many bytes you want to move, and exactly where you want it copied from and to, you can write a very fast loop. For example, suppose I know that I want to copy 20 bytes from BUFFER1 into BUFFER2, and that there is no overlap. Then I can write:

```
       LDX #19
LOOP   LDA BUFFER1,X
       STA BUFFER2,X
       DEX
       BPL LOOP
       ...
```

The loop moves the last byte first, then the next-to-last, and so on until the first byte in BUFFER1 is moved into BUFFER2. If it is important to move them in the opposite direction (first byte first, last byte last), you can change the loop this way:

```
       LDX #0
LOOP   LDA BUFFER1,X
       STA BUFFER2,X
       INX
       CPX #20
       BCC LOOP
       ...
```

Terminating the loop can be done in various ways. The two examples above do it with a count in the X-register. Another way is to use a data sentinel.  For example, the last byte to be moved, and only the last byte, might contain the value $00, or $FF, or anything you choose.  Then after moving a byte, you can check to see if the sentinel byte was just moved.  If it was, you are finished moving. Here is an example using a sentinel of $00:

```
       LDX #-1
LOOP   INX
       LDA BUFFER1,X
       STA BUFFER2,X
       BNE LOOP
       ...
```

Pascal Language promoters often recommend the sentinel technique; however, in Assembly Language, you msut be very careful if you plan to use it.  The sentinel you choose today may become a valid data value tomorrow!

Moving Bigger Blocks:  All of the examples so far will only work if the total number of bytes to be moved is less than 256.  What if you need to move a larger block?

When I need to move a large block of data from one place to another, I frequently use the MOVE subroutine in the Apple Monitor ROM.  It starts at $FE2C, and looks like this:

```
FE2C- B1 3C    MOVE LDA (A1L),Y  MOVE (A1...A2)
FE2E- 91 42         STA (A4L),Y    TO (A4)
FE30- 20 B4 FC      JSR NSTA4
FE33- 90 F7         BCC MOVE
FE35- 60            RTS
```

The subroutine NXTA4 (at $FCB4) increments A4L,A4H ($42,43), which is the destination address.  Then it compares A1L,A1H ($3C,3D) to A2L,A2H ($3E,3F); the result of the comparison is left in the Carry Status bit: Carry is set if A1 is greater than or equal to A2.  Finally, the subroutine increments A2L,A2H ($3E,3F).

To use the MOVE subroutine, you have to set the starting address of the block to be copied into $3C,3D; the last address of the block to be copied into $3E,3F; and the starting address of the destination into $42,43.  You also need to be sure that the Y-register contains zero before you start.  Here is an example:

```
       LDY #0          CLEAR Y-REGISTER
       LDA #BUFFER1    START ADDRESS OF SOURCE
       STA $3C
       LDA /BUFFER1
       STA $3D
       LDA #BUFFER1.END  END ADDRESS OF SOURCE
       STA $3E
       LDA /BUFFER1.END
       STA $3F
       LDA #BUFFER2    START ADDRESS OF DESTINATION
```

```
       STA $42
       LDA /BUFFER2
       STA $43
       JSR $FE2C
       ...
```

Because it is there, the Monitor MOVE subroutine is handy.  But it is not a general subroutine.  If the source and destination blocks overlap, you may get funny results.  For example, if I try to move the data between $1000 and $10FF up one byte in memory, so that it runs from $1001 to $1100, the MOVE subroutine will not work.  Instead, it will copy the contents of $1000 into every location from $1001 through $1100.

The MOVE subroutine is also not very fast.  Anyway, it is not as fast as it could be.  Steve Wozniak evidently wrote with size in mind (to make it fit in ROM) rather than speed.

The Applesoft ROMs contain several subroutines for moving data around in memory.  Here is one used during execution to move the array table up to make room for a new simple variable:

<<<<listing of BLTU, $D393...D3D5>>>>

Since this code moves from the end of the block backwards, it will safely move a block up in memory.  However, it would not be save to use with an overlapping range down in memory; it will do the same thing as the Monitor MOVE subroutine.

The Applesoft subroutine is faster than the Monitor subroutine, because the least significant half of the pointer is kept in the Y-register instead of in page-zero of memory.  The INY instruction takes only two cycles, whereas an INC instruction takes five.  The three cycles saved in moving each byte add up to nearly 25 milliseconds in moving 8K bytes.  The extra overhead of setting up the pointers is more than paid for.

Additional time is saved in the termination test.  Instead of testing after moving every byte with a LDA, CMP, LDA, SBC sequence, the number of full 256-byte blocks to be moved is put in the X-register; only a DEX instruction once out of every 256 bytes is needed.  This saves over 100 millisecondes in moving an 8K block.  By putting the incrementing and testing code in line, rather than in a subroutine like NXTA4, we save the JSR and RTS time.  This amounts to another 100 milliseconds in moving an 8K block.

A General Move Subroutine:  Can we write a subroutine which will move a block of data from one place to anothere regardless of overlap and direction?  Of course!  All we have to do is test at the beginning for direction, and choose which method to use accordingly.

Here is a fast subroutine which will move any block of memory anywhere you want.  You call it by putting the starting address of the source block in A1L,A1H; the end address of the source in A2L,A2H; and the start address of the destination in A4L,A4H.  (This is the same way you set up the MOnitor MOVE subroutine.)  I wrote it to be used with the control-Y monitor command.

<<<<listing of general move subroutine>>>>

```
================================================================
DOCUMENT :AAL-8101:DOS3.3:S.AmperGosub.txt
================================================================

 1000  *---------------------------------
 1010  *      &GOSUB <EXPRESSION>
 1020  *---------------------------------
 1030  TKN.GOSUB  .EQ $B0
 1040  *---------------------------------
 1050  AS.SYNCHR  .EQ $DEC0
 1060  AS.MEMCHK  .EQ $D3D6
 1070  AS.TXTPTR  .EQ $B8,B9
 1080  AS.LINNUM  .EQ $50,51
 1090  AS.FRMNUM  .EQ $DD67
 1100  AS.GOTO1   .EQ $D941
 1110  AS.NEWSTT  .EQ $D7D2
 1120  AS.GETADR  .EQ $E752
 1130  *---------------------------------
 1140         .OR $300
 1150  VARIABLE.GOSUB
 1160         LDA #TKN.GOSUB  CHECK IF &GOSUB
 1170         JSR AS.SYNCHR
 1180         LDA #3          CHECK IF ROOM ON STACK
 1190         JSR AS.MEMCHK
 1200         LDA AS.TXTPTR+1
 1210         PHA             STACK TXTPTR
 1220         LDA AS.TXTPTR
 1230         PHA
 1240         LDA AS.LINNUM+1
 1250         PHA             STACK CURRENT LINE NO.
 1260         LDA AS.LINNUM
 1270         PHA
 1280         LDA #TKN.GOSUB  MARK STACK
 1290         PHA
 1300         JSR AS.FRMNUM   EVALUATE FORMULA
 1310         JSR AS.GETADR   CONVERT TO INTEGER
 1320         JSR AS.GOTO1    USE GOTO CODE
 1330         JMP AS.NEWSTT
```

```
==========================================================================
DOCUMENT :AAL-8101:DOS3.3:S.ASoft.BLTU.txt
==========================================================================

 1000  *---------------------------------
 1010  *      BLTU -- FROM THE APPLESOFT ROM
 1020  *      $D393 THROUGH $D3D5
 1030  *---------------------------------
 1040  *      ON ENTRY:
 1050  *          Y,A AND HIGHDS CONTAIN DESTINATION END + 1
 1060  *          LOWTR CONTAINS LOWEST ADDRESS OF SOURCE
 1070  *          HIGHTR CONTAINS HIGHEST SOURCE ADDRESS + 1
 1080  *---------------------------------
 1090  *      PAGE-ZERO VARIABLE NAMES FROM "THE APPLE ORCHARD"
 1100  *      VOL. 1, NO. 1, PAGES 12-18.
 1110  STREND .EQ $6D,6E   TOP OF ARRAY STORAGE
 1120  HIGHDS .EQ $94,95   BLTU'S DESTINATION POINTER
 1130  HIGHTR .EQ $96,97   BLTU'S SOURCE END POINTER
 1140  LOWTR  .EQ $9B,9C   BLTU'S SOURCE START POINTER
 1150  *---------------------------------
 1160  REASON .EQ $D3E3    CHECK IF ENOUGH MEMORY
 1170  *---------------------------------
 1180  BLTU   JSR REASON   BE SURE (Y,A) < FRETOP
 1190         STA STREND   NEW TOP OF ARRAY STORAGE
 1200         STY STREND+1
 1210         SEC          COMPUTE # OF BYTES TO BE MOVED
 1220         LDA HIGHTR
 1230         SBC LOWTR
 1240         STA $5E      SAVE PARTIAL PAGE AMOUNT
 1250         TAY          ALSO IN Y
 1260         LDA HIGHTR+1
 1270         SBC LOWTR+1
 1280         TAX          NUMBER OF WHOLE PAGES IN X
 1290         INX
 1300         TYA          # BYTES IN PARTIAL PAGE
 1310         BEQ .4       NO PARTIAL PAGE
 1320         LDA HIGHTR   BACK UP HIGHTR BY PARTIAL PAGE #
 1330         SEC
 1340         SBC $5E
 1350         STA HIGHTR
 1360         BCS .1
 1370         DEC HIGHTR+1
 1380         SEC
 1390  .1     LDA HIGHDS   BACK UP HIGHDS BY PARTIAL PAGE #
 1400         SBC $5E
 1410         STA HIGHDS
 1420         BCS .3
 1430         DEC HIGHDS+1
 1440         BCC .3       ...ALWAYS
 1450  .2     LDA (HIGHTR),Y
 1460         STA (HIGHDS),Y
 1470  .3     DEY
 1480         BNE .2       LOOP TO END OF THIS 256 BYTES
```

```
1490         LDA (HIGHTR),Y  MOVE ONE MORE BYTE
1500         STA (HIGHDS),Y
1510  .4     DEC HIGHTR+1    DOWN TO NEXT BLOCK OF 256
1520         DEC HIGHDS+1
1530         DEX          PAGE COUNT
1540         BNE .3
1550         RTS
```

```
=============================================================================
DOCUMENT :AAL-8101:DOS3.3:S.EDIT.COMMAND.txt
=============================================================================

 1000  *---------------------------------
 1010  *  EDIT COMMAND FOR S-C ASSEMBLER II VERSION 4.0
 1020  *
 1030  *      WRITTEN BY MIKE LAUMER
 1040  *                 DECEMBER 6, 1980
 1050  *---------------------------------
 1060         .OR $0800
 1070         .TF B.EDIT2
 1080  *---------------------------------
 1090  *  SYSTEM EQUATES
 1100  *---------------------------------
 1110  MON.COUT   .EQ $FDED
 1120  MON.BELL   .EQ $FF3A
 1130  MON.RDKEY  .EQ $FD0C
 1140  MON.CLREOP .EQ $FC42
 1150  MON.VTAB   .EQ $FC22
 1160  CH     .EQ $24
 1170  CV     .EQ $25
 1180  DOS.REENTRY .EQ $03D0
 1190  *---------------------------------
 1200  *  ASSEMBLER EQUATES
 1210  *---------------------------------
 1220  GNL    .EQ $1026
 1230  NML    .EQ $1063
 1240  PLNO   .EQ $1779
 1250  GNB    .EQ $12C5
 1260  DOIT   .EQ $1874
 1270  SEARCH .EQ $164B
 1280  SERTXT .EQ $14F6
 1290  SERNXT .EQ $14FE
 1300  NTKN   .EQ $12AF
 1310  A0L    .EQ $3A,3B
 1320  A1L    .EQ $3C,3D
 1330  SRCP   .EQ $DD,DE
 1340  WBUF   .EQ $0200
 1350  CURRENT.LINE.NUMBER .EQ $D3,D4
 1360  *---------------------------------
 1370  *  ENTRY POINT FOR BRUN.  ACTIVATES
 1380  *  THE USR ASSEMBLER COMMAND.
 1390  *---------------------------------
 1400  ENTRY  LDA #EDIT
 1410         STA $1007    PATCH ASM USR COMMAND
 1420         LDA /EDIT
 1430         STA $1008
 1440         LDA #$60     PATCH NML TO MAKE IT
 1450         STA $1125        A SUBROUTINE
 1460         LDA #$4C
 1470         STA NML
 1480         STA $1078
```

```
1490        LDA #NEW.NML
1500        STA NML+1
1510        LDA /NEW.NML
1520        STA NML+2
1530        JMP DOS.REENTRY
1540 *---------------------------------
1550 *  PATCH ROUTINES FOR ASSEMBLER
1560 *---------------------------------
1570 NEW.NML JSR MY.NML
1580        JMP GNL
1590 MY.NML LDY #0
1600        JSR $128D
1610        JSR $114A
1620        JMP $1066
1630 *---------------------------------
1640 *  LOCAL VARIABLES FOR EDIT COMMAND
1650 *---------------------------------
1660 NEXT   .DA 0
1670 END    .DA 0
1680 CHAR   .DA #0
1690 EDPTR  .DA #0
1700 FKEY   .DA #0
1710 *---------------------------------
1720 EDIT   DEX
1730        DEX
1740        BMI .2        NO ARGUMENTS
1750        BEQ .4        1 ARGUMENT
1760        JSR .3        2 ARGUMENTS
1770        LDX #A1L      FIND END PTR
1780        JSR SERNXT
1790        LDA $E6
1800        STA END
1810        LDA $E7
1820        STA END+1
1830 .1     LDA NEXT+1
1840        STA SRCP+1
1850        PHA
1860        LDA NEXT
1870        STA SRCP
1880        CMP END
1890        PLA
1900        SBC END+1     PAST END LINE?
1910        BCS .2        YES, EXIT
1920        JSR E.LIST    NO, LIST AND EDIT
1930        JMP .1        TRY FOR NEXT LINE
1940 .3     LDX #A0L      FIND START PTR
1950        JSR SERTXT
1960        LDA $E4
1970        STA SRCP
1980        STA NEXT      SAVE NEXT LINE ADRS
1990        LDA $E5
2000        STA SRCP+1
2010        STA NEXT+1
2020 .2     RTS
```

```
2030  .4     JSR .3       SEARCH FOR LINE
2040         BCC .2       NOT FOUND EXIT
2050  E.LIST JSR E.POSN   POSITION FOR EDIT
2060         JSR MON.CLREOP  PREPARE DISPLAY
2070         JSR GNB      GET LINE SIZE
2080         CLC
2090         ADC NEXT     COMPUTE NEXT LINE ADRS
2100         STA NEXT
2110         TYA
2120         ADC NEXT+1
2130         STA NEXT+1
2140         JSR GNB      GET LINE NUMBER FOR DISPLAY
2150         STA CURRENT.LINE.NUMBER
2160         JSR GNB
2170         STA CURRENT.LINE.NUMBER+1
2180         SEC
2190         ROR $F8      STUFF WBUF FLAG
2200         JSR PLNO
2210         LSR $F8      TURN OFF FLAG
2220         LDA #$20     SPACE AFTER LINE #
2230         LDX #0
2240  .1     STX EDPTR
2250         ORA #$80     FORCE VIDEO BIT
2260         STA WBUF+4,X STORE INTO INPUT BUFFER
2270         CMP #$A0     TEST FOR CONTROL CHAR
2280         BCS .2       OK, IF NOT
2290         AND #$7F     OUTPUT INVERSE ALPHA
2300  .2     JSR MON.COUT PRINT CHAR
2310         JSR NTKN     GET NEXT TOKEN
2320         LDX EDPTR
2330         INX
2340         CMP #0       END TOKEN?
2350         BNE .1       NO, PRINT IT
2360         STA WBUF+4,X YES, PUT IT IN TOO
2370  E.LINE LDX #0
2380  E.0    STX EDPTR
2390  E.1    JSR E.INPUT  GET INPUT CHAR
2400  E.2    LDA #EDTB
2410         STA $2
2420         LDA /EDTB
2430         STA $3
2440         LDA #CHAR
2450         STA $12
2460         LDA /CHAR
2470         STA $13
2480         JSR SEARCH   SEARCH EDIT COMMAND TABLE
2490         BNE .2       NOT IN TABLE
2500         LDX EDPTR
2510         JSR DOIT     EXECUTE COMMAND ROUTINE
2520         BCC E.0      NO DISPLAY ON RETURN
2530         BCS .5       DISPLAY ON RETURN
2540  .2     LDX EDPTR    MUST BE TYPE OVER
2550         LDA CHAR
2560         CMP #$A0
```

```
2570        BCS .4
2580  .3     JSR MON.BELL ERR IF CONTROL KEY
2590        JMP E.1
2600  .4     LDA WBUF+5,X SEE IF END OF LINE
2610        BNE .6       TYPE OVER IF NOT
2620        STA WBUF+6,X SHIFT OVER END OF LINE
2630  .6     LDA CHAR     STUFF CHAR INTO BUFFER
2640        STA WBUF+5,X
2650        CPX #256-5-2 TEST BUFFER SIZE
2660        BEQ .5       TYPE OVER LAST CHAR IN BUFFER
2670        INX          INSTEAD OF BUFFER END
2680  .5     JSR E.DISP   DISPLAY LINE
2690        JMP E.0      GET NEXT EDIT COMMAND
2700  *---------------------------------
2710  E.POSN LDA #19      POSITION TO LINE 19,
2720        STA CV
2730        LDA #0       COLUMN 0
2740        STA CH
2750        JMP MON.VTAB
2760  *---------------------------------
2770  E.DISP STX EDPTR
2780        JSR E.POSN   POSITION DISPLAY
2790        LDX #$FF
2800  .1     INX
2810        LDA WBUF,X   GET BUFFER CHAR
2820        BEQ .3       END OF BUFFER
2830        CMP #$A0     CONTROL CHAR?
2840        BCS .2       NO
2850        AND #$7F     PRINT INVERSE ALPHA
2860  .2     JSR MON.COUT PRINT CHAR
2870        JMP .1       NEXT CHAR
2880  .3     JSR MON.CLREOP CLEAN ANY REMAINING SCREEN
2890        LDX EDPTR
2900        RTS
2910  *---------------------------------
2920  E.BEG  LDX #0   SET CURSOR TO BEGINNING OF LINE
2930        CLC
2940        RTS
2950  *---------------------------------
2960  E.DEL  LDA WBUF+5,X IS THIS THEN END OF
2970        BEQ .2
2980  .1     INX
2990        LDA WBUF+5,X SHIFT TO LOWER MEMORY
3000        STA WBUF+4,X TO DELETE CHAR
3010        BNE .1
3020        LDX EDPTR
3030  .2     SEC          RETURN WITH DISPLAY
3040        RTS
3050  *---------------------------------
3060  E.END  LDA WBUF+5,X END OF BUFFER?
3070        BEQ .1       YES
3080        INX          NO
3090        BNE E.END    TRY END AGAIN
3100  .1     CLC          RETURN NO DISPLAY
```

```
3110         RTS
3120  *---------------------------------
3130  E.FIND LDA WBUF+5,X END OF BUFFER?
3140         BNE .2       NO
3150  .1     STA FKEY     YES SO ERR
3160         JSR MON.BELL RING BELL
3170         CLC          RETURN NO DISPLAY
3180         RTS
3190  .2     JSR E.INPUT  GET 1 CHAR
3200         STA FKEY     SAVE KEY TO LOCATE
3210  .3     INX
3220         LDA WBUF+5,X TEST BUFFER
3230         BEQ .1       END OF BUFFER
3240         CMP FKEY     NO, SEE IF KEY
3250         BNE .3       NO, GO FORWARD
3260         JSR E.INPUT  TRY ANOTHER KEY
3270         CMP FKEY     SAME CHAR?
3280         BEQ .3       YES, SEARCH AGAIN
3290         PLA
3300         PLA
3310         STX EDPTR    NO, EXIT POINTING HERE
3320         JMP E.2
3330  *---------------------------------
3340  E.BKSP TXA          AT BEGINNING?
3350         BEQ .1       YES, STAY THERE
3360         DEX          BACKUP
3370  .1     CLC          RETURN NO DISPLAY
3380         RTS
3390  *---------------------------------
3400  E.OVR  JSR E.INPUT  READ CHAR
3410         JMP E.INS1   SKIP CONTROL CHECK
3420  *---------------------------------
3430  E.INS  JSR E.INPUT  READ CHAR
3440         CMP #$A0     CONTROL CHAR POPS USER OUT
3450         BCC E.INS2   OF INSERT
3460  E.INS1 CPX #256-5-2 END OF BLOCK
3470         BEQ .1       YES STAY THERE
3480         INX
3490  .1     STX EDPTR
3500  .2     PHA          CHAR TO INSERT
3510         LDA WBUF+4,X SAVE CHAR TO MOVE
3520         TAY
3530         PLA          GET CHAR TO INSERT
3540         STA WBUF+4,X PUT OVER SAVED CHAR
3550         INX
3560         TYA          INSERT SAVED CHAR
3570         BNE .2       IF NOT BUFFER END
3580         STA WBUF+4,X STUFF END CODE
3590         STA WBUF+256-5-1  INSURE A  END CODE
3600         LDX EDPTR
3610         JSR E.DISP   DISPLAY LINE
3620         JMP E.INS    GET NEXT INSERT CHAR
3630  E.INS2 PLA          SEND CHAR TO
3640         PLA          COMMAND SEARCH
```

```
3650         LDX EDPTR
3660  *---------------------------------
3670         JMP E.2
3680  E.RETQ LDA #0       CLEAR REST OF LINE
3690         STA WBUF+5,X
3700         JSR E.DISP   DISPLAY LINE
3710  E.RET  LDX #$FF     SUBMIT LINE TO ASSEMBLER
3720  .1     INX          COMPUTE LINE SIZE
3730         LDA WBUF,X
3740         BNE .1
3750         DEX
3760  .2     STX $E1      SAVE SIZE
3770         PLA
3780         PLA
3790         JMP MY.NML   SUBMIT THE LINE
3800  *---------------------------------
3810  E.TAB  CPX #20      < COL 20?
3820         BCS .1       NO
3830         LDA WBUF+5,X END OF BUFFER?
3840         BEQ .1       YES
3850         INX          MOVE FORWARD
3860         CPX #7       TAB MATCH?
3870         BEQ .1
3880         CPX #11      TAB MATCH?
3890         BNE E.TAB
3900  .1     CLC          RETURN WITHOUT DISPLAY
3910         RTS
3920  *---------------------------------
3930  E.RIT  LDA WBUF+5,X END OF BUFFER
3940         BNE .1       NO
3950         STA WBUF+6,X
3960         LDA #$A0     PUT A BLANK
3970         STA WBUF+5,X TO EXTEND LINE
3980         CPX #256-5-2
3990         BEQ .2
4000  .1     INX          MOVE AHEAD
4010  .2     CLC          RETURN NO DISPLAY
4020         RTS
4030  *---------------------------------
4040  E.ABORT LDA #$DC    OUTPUT BACKSLASH
4050         STA WBUF+5
4060         LDA #0
4070         STA WBUF+6
4080         JSR E.DISP   SHOW CANCEL
4090         JMP GNL      GET NEXT COMMAND
4100  *---------------------------------
4110  E.INPUT LDA #19
4120         STA CV
4130         TXA          POSITION TO CURSOR
4140         CLC
4150         ADC #5
4160  .1     CMP #40      THIS LINE?
4170         BCC .2       YES
4180         SEC
```

```
4190         SBC #40
4200         INC CV       ON NEXT LINE
4210         BNE .1
4220  .2     STA CH
4230         JSR MON.VTAB SET BASL
4240         JSR MON.RDKEY INPUT A CHAR
4250         STA CHAR
4260         RTS
4270  *---------------------------------
4280  *  COMMAND TABLE
4290  *---------------------------------
4300  EDTB   .DA #3,#1    ITEM SIZE, KEY SIZE
4310   .DA #$82,E.BEG-1   ^B
4320   .DA #$84,E.DEL-1   ^D
4330   .DA #$85,E.END-1   ^E
4340   .DA #$86,E.FIND-1  ^F
4350   .DA #$88,E.BKSP-1  ^H
4360   .DA #$89,E.INS-1   ^I
4370   .DA #$8D,E.RET-1   ^M
4380   .DA #$8F,E.OVR-1   ^O
4390   .DA #$91,E.RETQ-1  ^Q
4400   .DA #$94,E.TAB-1   ^T
4410   .DA #$95,E.RIT-1   ^U
4420   .DA #$98,E.ABORT-1 ^X
4430   .DA #0
```

```
================================================================================
DOCUMENT :AAL-8101:DOS3.3:S.GENERAL.MOVE.txt
================================================================================
```

```
1000  *--------------------------------
1010  *      GENERAL MOVE SUBROUTINE
1020  *--------------------------------
1030  *      BRUN THE PROGRAM TO SET UP AS CONTROL-Y
1040  *      MONITOR ROUTINE
1050  *--------------------------------
1060  *      USE LIKE MONITOR MOVE SUBROUTINE:
1070  *      A1L,A1H -- SOURCE START ADDRESS
1080  *      A2L,A2H -- SOURCE END ADDRESS
1090  *      A4L,A4H -- DESTINATION START ADDRESS
1100  *--------------------------------
1110  BLOCK.SIZE .EQ $00,01
1120  A1L    .EQ $3C
1130  A1H    .EQ $3D
1140  A2L    .EQ $3E
1150  A2H    .EQ $3F
1160  A4L    .EQ $42
1170  A4H    .EQ $43
1180  CONTROL.Y  .EQ $3F8
1190  *--------------------------------
1200  CONTROL.Y.SETUP
1210         LDA #$4C     JMP OPCODE
1220         STA CONTROL.Y
1230         LDA #GENERAL.MOVE
1240         STA CONTROL.Y+1
1250         LDA /GENERAL.MOVE
1260         STA CONTROL.Y+2
1270         RTS
1280  *--------------------------------
1290  GENERAL.MOVE
1300         PHA          SAVE REGISTERS
1310         TYA
1320         PHA
1330         TXA
1340         PHA
1350         INC A2L      BUMP END ADDRESS ONCE
1360         BNE .1
1370         INC A2H
1380  .1     SEC          COMPUTE SIZE OF BLOCK
1390         LDA A2L
1400         SBC A1L
1410         STA BLOCK.SIZE
1420         LDA A2H
1430         SBC A1H
1440         STA BLOCK.SIZE+1
1450         TAX
1460         INX          NUMBER OF BLOCKS TO MOVE
1470         LDA A1L      DETERMINE DIRECTION
1480         CMP A4L
```

```
1490        LDA A1H
1500        SBC A4H
1510        BCC .2       A1 < A4
1520        JSR MOVE.DOWN
1530        JMP .3
1540 .2     JSR MOVE.UP
1550 .3     PLA          RESTORE REGS
1560        TAX
1570        PLA
1580        TAY
1590        PLA
1600        RTS
1610 *---------------------------------
1620 MOVE.DOWN
1630        LDY #0
1640        DEX          ANY WHOLE BLOCKS LEFT?
1650        BEQ .2       NO
1660 .1     LDA (A1L),Y  MOVE 256 BYTES
1670        STA (A4L),Y
1680        INY
1690        BNE .1
1700        INC A1H      POINT AT NEXT BLOCK
1710        INC A4H
1720        DEX          ANY MORE WHOLE BLOCKS?
1730        BNE .1       YES
1740 .2     LDX BLOCK.SIZE   ANY EXTRA BYTES IN A SHORT BLOCK?
1750        BEQ .4       NONE LEFT
1760 .3     LDA (A1L),Y
1770        STA (A4L),Y
1780        INY
1790        DEX
1800        BNE .3
1810 .4     RTS
1820 *---------------------------------
1830 MOVE.UP
1840        CLC          COMPUTE DESTINATION END + 1
1850        LDA A4L
1860        ADC BLOCK.SIZE
1870        STA A4L
1880        LDA A4H
1890        ADC BLOCK.SIZE+1
1900        STA A4H
1910        LDY #0
1920        BEQ .3       ...ALWAYS
1930 *---MOVE A WHOLE BLOCK------------
1940 .1     LDA (A2L),Y  MOVE BYTES 255 THRU 1 IN BLOCK
1950        STA (A4L),Y
1960 .2     DEY
1970        BNE .1
1980        LDA (A2L),Y  MOVE LOWEST BYTE IN BLOCK
1990        STA (A4L),Y
2000 .3     DEC A2H
2010        DEC A4H
2020        DEX          ANY MORE BLOCKS?
```

```
2030          BNE .2        YES
2040  *---MOVE SHORT BLOCK IF ANY-------
2050          LDX BLOCK.SIZE
2060          BEQ .5        NONE LEFT
2070  .4      DEY
2080          LDA (A2L),Y
2090          STA (A4L),Y
2100          DEX
2110          BNE .4
2120  .5      RTS
```

```
==============================================================================
DOCUMENT :AAL-8101:DOS3.3:Test.AmperGosub.txt
==============================================================================
```

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8102:Articles:AppleNoiseSound.txt
==============================================================================

Making Noise and Other Sounds

The Apple's built-in speaker is one of its most delightful features. To be sure, it is very limited; but I have used it for everything from sound effects in games to music in six parts (weird-sounding guitar chords) and even speech. Too many ways to put all in one AAL article! I will describe some of the sound effects I have used, and maybe you can go on from there.

The speaker hardware is very simple. A flip-flop controls the current through the speaker coil. Everytime you address $C030, the flip-flop changes state. This in turn reverses the current through the speaker coil. If the speaker cone was pulled in, it pops out; if it was out, it pulls in. If we "toggle" the state at just the right rate, we can make a square-wave sound. By changing the time between reversals dynamically, we can make very complex sounds. We have no control over the amplitude of the speaker motions, only the frequency.

Simple Tone: This program generates a tone burst of 128 cycles (or 256 half-cycles, or 256 pulses), with each half-cycle being 1288 Apple clocks. Just to make it easy, let's call Apple's clock 1MHz. It is really a little faster, but that will be close enough. So the tone will be about 388 Hertz (cycles per second, if you are as old as me!).

How did I figure out those numbers? To get the time for a half-cycle (which I am going to start calling a pulse), I added up the Apple 6502 cycles for each instruction in the loop. LDA SPEAKER takes 4 cycles. DEX is 2 cycles, and BNE is 3 cycles when it branches. The DEX-BNE pair will be executed 256 times for each pulse, but the last time BNE does not branch; BNE only takes 2 cycles when it does not branch. The DEY-BNE pair will branch during each pulse, so we use 5 cycles there. So the total is 4+256*5-1+5=1288 cycles. I got the frequency by the formula f=1/T; T is the time for a whole cycle, or 2576 microseconds.

Apple "Bell" Subroutine: Inside your monitor ROM there is a subroutine at $FBE2 which uses the speaker to make a bell-like sound. Here is a copy of that code. Notice that the pulse width is controlled by calling another monitor subroutine, WAIT.

Machine-Gun Noise: What if we use a random pulse width? Then we get something called noise, instead of a tone. We can create a burst of pulses of random-sounding width by using values from some arbitrary place in the Apple's memory as loop counts. The program uses the 256 values starting at $BA00 (which is inside DOS). If you make just one burst like that, it doesn't sound like much. But if you make ten in a row, you get a pattern of repetitious random noise bursts that in this case sounds like machine-gun fire. Doesn't it? Well, close enough....

Laser "SWOOP" Sound: We can change the pulse width by making it go from wide to narrow in steps of 5 microseconds. It sounds like a low tone that gradually slides higher and higher until it is beyond the range of the human ear (or the Apple speaker). I used this program in a "space war" game to go with the laser fire. Even though the sound was entirely generated before the laser even appeared on the screen, it looks and sounds like the light beam and sound are simultaneous.

I have indicated in line 1110 that you should try experimenting with some other values for the maximum pulse width count. I have included a separate entry point at SWOOP2 to make ten swoops in a row. Try the various values for the maximum width and run each one from SWOOP2. You might also experiment with running the pulse width in the opposite direction (from narrow to wide) by changing line 1200 to INC PULSE.WIDTH.

Another Laser Blast: This one sounds very much the same as the swoop of the previous program, but it uses less memory. You should try experimenting with the pulse widths of the first and last pulses in lines 1060 and 1130. You could also try changing the direction by substituting a DEX in line 1120.

Inch-Worm Sounds: I stumbled onto this one by accident, while looking for some sound effects for a lo-res graphics demo. The demo shows what is supposed to be an inch-worm, inching itself across the screen. By plugging various values (as indicated in lines 1100 and 1130), I got some sounds that synchronized beautifully with the animation. Complete with an exhausted sigh at the end!

Touch-Tones Simulator: I used this one with a telephone demo program. The screen shows a touch tone pad. As you press digits on the keyboard, the corresponding button on the screen lights up (displays in inverse mode). Then the demo program CALLs this machine language code to produce the twin-tone sound that your telephone makes. It isn't perfect, you can't fool the Bell System. But it makes a good demo!

I will describe the program from the top down. The four variables in page zero are kept in a "safe" area, inside Applesoft's floating point accumulator. Applesoft doesn't use these locations while executing a CALLed machine language routine.

The Applesoft demo program stores the button number (0-9) in location $E7. This could be done with "POKE 231,DGT", but I had more fun using "SCALE=DGT". SCALE= is a hi-res graphics command, but all it really does is store the value as a one-byte integer in $E7. Since we aren't using hi-res graphics, the location is perfectly safe to use.

CALL 768 gets us to line 1150, TWO.TONES. This is the main routine. It uses the button number to select the two tone numbers from LOW.TONES and HIGH.TONES. ONE.TONE is called to play first the low tone, then the high tone, back and forth, for ten times each. This is my attempt to fool the ear, to make it sound like both are being played at once.

ONE.TONE wiggles the speaker for LENGTH half-cycles. Each half-cycle is controlled by either the UPTIME or DOWNTIME counts. These three parameters are selected from three tables, according to the tone number selected by TWO.TONES. Lines 1270-1340 pick up the values from the three tables and load the page zero variables. Lines 1360-1500 do the actual speaker motions and time everything. The purpose of having two routines, one for uptime and one for downtime, is to be able to more closely approximate the frequency. For example, if the loop count we ought to use is 104.5, we could use an uptime of 104 and a down time of 105; this makes the total time for the full cycle correct. The redundant BEQ in line 1420 is there to make the loop times for UPTIME and DOWNTIME exactly the same.

Since you do not have my Applesoft program, which drives this, I wrote a simulated drive to just "push" the buttons 0-9. Lines 1650-1790 do this. I separated each button push by a call to the monitor WAIT subroutine, to make them easier to distinguish.

Morse Code Output: I have always thought that computers really only need one output line and one input line for communicating with humans. I could talk to my Apple with a code key, and it could beep back at me. One of the first programs I attempted in 6502 language was a routine to echo characters in Morse code. I looked it up about two hours ago, and shuddered at my sloppy, inefficient, hard to follow code. So, I wrote a new one.

I broke the problem down into three littler ones: 1) getting the characters which are to be output; 2) converting the ASCII codes to the right number of dots and dashes; and 3) making tones and spaces of the right length.

SETUP.MORSE (lines 1190-1240) links my output routine through the monitor output vector. Line 1240 JMPs to $3EA to re-hook DOS after me.

MORSE (lines 1260-1310) are an output filter. If the character code is less than $B0, I don't know how to send it in Morse code; therefore, I just go to $FDF0 to finish the output on the screen. Codes exist for these other characters, but I did not look them up. If you want a complete routine, you should modify line 1260 to CMP #$A0 and add the extra codes to the code table (lines 1130-1170).

SEND.CHAR looks up the Morse code for the character in the code table, and splits it into the number of code elements (low-order three bits) and the code elements themselves (high-order five bits). If a code element is zero, a short beep (dot) is sounded. If an element is one, three calls to the short beep routine make one long beep (dash). Between elements, a silence equal to the length of a short beep intervenes. After the last beep of a character, a longer silence, equal to three short silences, is produced. A 00 code from the code table makes a silent gap of three times the inter-character gap.

EL.SPACE and EL.DIT are nearly identical. The only difference is that EL.DIT makes a sound by addressing the speaker, while EL.SPACE does not. The value of EL.PITCH determines the pulse width, and EL.SPEED determines the number of pulses for an inter-element-space or a short beep. If the code stream is too fast for you, you can slow it down by increasing either or both of these two numbers.

```
==============================================================================
DOCUMENT :AAL-8102:Articles:AS.Str.Swapper.txt
==============================================================================
```

A String Swapper for Applesoft

Practically every program rearranges data in some way. Many times you must sort alphanumeric data, and Applesoft makes this relatively easy. At the heart of most sort algorithms you will have to swap two items.

If the items are numbers, you might do it like this: T=A(I) : A(I)=A(J) : A(J)=T. If the items are in string variables, you might use this: T$=A$(I) : A$(I)=A$(J) : A$(J)=T.

Before long, Applesoft's wonderful string processor eats up all available memory and your program screeches to a halt with no warning. You think your computer died. Just about the time you reach for the power switch, it comes to life again (if you aren't too impatient!); the garbage collection procedure has found enough memory to continue processing. If only Applesoft had a command to swap the pointers of two strings, this wouldn't happen.

What are pointers? Look on page 137 of your Applesoft Reference Manual. The third column shows how string variables are stored in memory. Each string, whether a simple variable or an element of an array, is represented by three bytes: the first byte tells how many bytes are in the string value at this time; the other two bytes are the address of the first byte of the string value. The actual string value may be anywhere in memory. I am calling the three bytes which define a string a "pointer".

All right, how can we add a string swap command? The authors of Applesoft thoughtfully provided us with the "&" command; it allows us to add as many new commands to the language as we want. (Last month I showed you how to add a computed GOSUB command using the &.) We could make up our own swap command; perhaps something like &SWAP A$(I) WITH A$(J). However, to keep it a little simpler, I wrote it this way: &A$(I),A$(J).

The program is in two sections. The first part, called SETUP, simply sets up the &-vector at $3F5, $3F6, and $3F7. It stores a "JMP SWAP" instruction there. When Applesoft finds an ampersand (&) during execution, it will jump to $3F5; our JMP SWAP will start up the second section.

SWAP calls on two routines inside the Applesoft ROMs: PTRGET ($DFE3) and SCAN.COMMA ($DEBE). I found the addresses for these routines in the article "Applesoft Internal Entry Points", by John Crossley, pages 12-18 of the March/April 1980 issue of The Apple Orchard. I also have disassembled and commented the Applesoft ROMs, so I checked to see if there were any bad side effects. Both routines assume that Applesoft is about to read the next character of your program. PTRGET assumes you are sitting on the first character of a variable name. SCAN.COMMA hopes you are sitting on a comma.

SWAP merely calls PTRGET to get the address of the pointer for the first variable, check for an intervening comma, and then calls PTRGET again to get the pointer address for the second variable. Then lines 1350-1430 exchange the three bytes for the two pointers.

How about a demonstration? I have a list of 20 names (all are subscribers to the Apple Assembly Line), and I want to sort them into alphabetical order. Since I am just writing this to demonstrate using the swap command, I will use one of the WORST sort algorithms: the bubble sort.

Line 100 clears the screen and prints a title line. Line 110 loads the swap program and calls SETUP at 768 ($0300). Line 120 reads in the 20 names from the DATA statement in line 130, and calls a subroutine at line 200 to print the names in a column.

Lines 150-170 are the bubble sort algorithm. If two names are out of order, they are swapped at the end of line 160. Line 180 prints the sorted list of names in a second column.

```
==============================================================================
DOCUMENT :AAL-8102:Articles:Front.Page.Misc.txt
==============================================================================
```

Stuffing Object Code in Protected Places

Several users of Version 4.0 have asked for a way to defeat the protection mechanism, so that they can store object code directly into the language card.  One customer has a EPROM burner which accepts code at $D000.  He wants to let the assembler write it out there directly, even though he could use the .TA directive and later a monitor move command.  Or, he could use the .TF directive, and a BLOAD into his EPROM.

For whatever reason, if you really want to do it, all you have to do is type the following patch just before you assemble:  $1A25:EA EA. In case you want to put it back, or check before you patch, what should be there is B0 28.

Bug Reports

1.  Several readers have reported a problem with the COPY program in the December issue.  As written, if you try to copy a block of lines to a point before the first line of the program, the block is inserted between the first and second bytes of the first line.  Ouch!  To fix it, insert lines 2221-2225 and change line 2250:

```
   2221         LDA A2L
   2222         CMP A1L
   2223         LDA A2H
   2224         SBC A1H
   2225         BCC .5

   2250 .5      LDA SS       MOVE IN SOURCE BLOCK
```

2.  When I typed up Lee Meador's article for the January issue, I inadvertently changed one address to a crazy value.  The address $2746 in the 4th paragraph on page 9 should be $1246.

3.  The Variable Cross Reference program for Applesoft from the November issue leaves something behind after it has run.  If you LIST the Applesoft program after running VCR, the line number of the first line will come out garbage.  This only happens the first time you use the LIST command.  For some reason, typing CALL 1002 before the LIST will fix it.  I haven't found out the cause or cure yet.  If you find it first, let me know!

In This Issue...

================================================================================
DOCUMENT :AAL-8102:Articles:GRAM.Buy.Printr.txt
================================================================================

Buying a Printer for your Apple II...................Mike Laumer

I purchased my first printer in November just before Thanksgiving. The process of selecting a printer can be confusing, painful, and very expensive. Here is my tale.

After writing printer drivers for other people's printers for several years, I was not convinced that the IDS 225 or the Paper Tiger were for me. They are fairly bulky, noisy, and the print quality was not up to the quality I am used to every day at work. The Trendcom 100 was quieter, but only 40 columns wide. The Trendcom 200 and Apple Silentype are 80 columns, but 40 columns per second is rather slow when you want to print 60 pages. From my experience thermal paper yellows and is hard to write on with ball point pens. The only thing I really liked about these printers was the price. The AXIOM printer (which prints on aluminum coated paper by blasting off the aluminum with electrical sparks, exposing a black paper beneath) was faster, but the weird paper looked expensive and did not come in fan-fold. I did like the speed and price. Several new manufacturers began advertising printers that looked good, but I could never watch them operate at a computer store, and I heard negative comments about them.

Enter the Japanese! I was getting desperate for a printer, ready to buy almost anything. I begain hearing rumors about the new EPSON MX-80 printer: $650, reliable, 80 columns per second, bi-directional printing, a possible graphics ROM add-on.... Sounded good, so I went shopping.

[ Store #1 ] I asked, "Do you sell the MX-80 printer?" They said, "It will be in next week, on Wednesday." I came back Wednesday, and saw the MX-80 working on an Apple II. The print clarity was the best I had seen on an inexpensive printer. It was comparable to the Centronix 779, which was huge, very noisy, and twice the price. "How much does it cost?" It was $130 more than advertised, but it included interface and cables.

[ Store #2 ] I went to another store, a new one I had never seen before. They had a bunch of Atari home computers (cute, aren't they?). "Do you sell anything for the Apple II? I see you don't have any Apples on the floor." The salesman was busy with a customer and a take-out lunch. I looked around, and noticed an MX-80 on a table. After getting his attention (he was quite busy eating his sandwich, and asking if I didn't mind), I asked the salesman a few questions about the printer and its price. "Only $499", he said. "How come the low price?" "We don't have a bunch of other printers to unload that are clearly beat by the price-performance of the MX-80." The Apple interface would run about $50, he thought. It looked like a good deal, so I went home to discuss it with my wife.

[ Store #2 ]   HOW NOT TO SELL ANYTHING....   My wife thought it sounded good, too.  I returned to the store a few days later.  The same salesman was there selling an Atari home computer (to me they are just programmable video games).  It was 15-20 minutes before he was done, but the prospect of the low printer price gave me patience...I waited.  After the sale, he picked up his sandwich and let me ask some more questions.  That's when I found out about the graphics ROM that Epson plans to offer in the future.  "We will be raising the price to $599 next week, but it is still $499 this week.  However, we are out of stock right now.  I can get you one by the middle of next week." But I really wanted to get one for the holiday weekend, since I could do a lot of computer work then.  "No way.  There just won't be any until next week.  And, you will have to pay now to get the price." This sales pitch was getting just a little suspicious...but the price still had me hooked.  I was trying to justify buying now, paying now, saving now, picking up later.  Then he began saying how he was the first Epson dealer in Dallas, and that the other stores had complained to Epson about his price.  He had to raise his price or Epson would not let him sell their printer any more.  "I sold 23 printers already this week", he bragged, as he hauled out a wad of checks from his pocket to show me.  "I can't spend any more time with you now.  My profit margin is too low to justify more than five minutes."  (There were no other customers in the store.)  Well, he convinced me, all right.  "Fine!"  I walked out the door, driving right over to....

[ Store #1 ]   "Do you have the MX-80 in stock?" I asked.  "Yes we do", replied the cashier.  "I would like to buy one", I stated.  The sales girl went into a back room, returning with a big box and a small box.  She took my charge card and rang up the sale.  I went home and had a great weekend.

Lesson for the Day:

1.  After all the rip-off's from the early days of the microcomputer market, nobody gets my money in advance unless they have built a reputation in the community.  I never saw this store before, and they wanted my money in advance after a strange sales push.

2.  Anyone who displays customer checks so casually to other customers gets an immediate black mark with me.  I wouldn't like mine to be treated in such a cavalier manner.

3.  I don't like to spend my lunch hour talking to someone stuffing his face while I am hungry.

4.  If it isn't profitable for the salesman to try to sell me his printer, I really want to know.  I'll go to someone who does believe it to be profitable, and is a  lot more courteous about selling it. If my $600 has no profit for him, I am not going to pay in advance and lose it when he goes bankrupt the next day, before I get my printer.

5.  Don't ever hire a turkey (even in late November!) who does a good job sending your customers to someone else's store.  Especially buying customers.

```
=====================================================================
DOCUMENT :AAL-8102:Articles:GRAM.Ftr.Laumer.txt
=====================================================================
```

The Future of Personal Computers...................Mike Laumer

The days of 8-bit microcomputers are numbered. First 16-bit, and now 32-bit chips are creeping out of the laboratories. INTEL, Hewlett-Packard, TI, and Motorola are shrinking the supercomputers down to 1/4-inch square slivers of silicon.

Motorola's 68000 microprocessor chip uses a 16-bit memory and input/output bus, but internally it has a 32-bit architecture. Texas Instruments has just announced the 99000, an upward-compatible enhancement of the 9900. The 99000 has new instructions and the fastest clock in the country...18 MHz!

The boys in the labs at Hewlett-Packard are spreading the word about their new 32-bit design. It multiplies two 16-bit numbers in 1.6 microseconds, and divides a 32-bit number by a 16-bit one in 3.5 microseconds. That's 12 times faster than the TI 9900! They are also working on a 528K bit ROM (equivalent to 64K bytes on one chip!) and a 128K RAM.

The INTEL 32-bit micro (iAPX 432) was designed together with the operating system; it supports multiprocessing and multitasking from the ground up. They claim to be abel to stack them in parallel to boost system throughput and performance up to the level of an IBM 370/158. It also executes an instruction set which easily supports ADA (a new programming language which is set to be the standard language for the Defense Department). INTEL already had to expand the ADA language to take advantage of the new architecture. The operating system itself is also coded in the ADA language.

The home computers of the mid and late 1980's will be very nice indeed! And maybe we won't even have to wait that long. Read this little clipping from EETimes:

If this is true, it may mean that the Apple IV is less than a year away!

Now in production are the INTEL 8086, Motorola 68000, and TI 9900; several more are on the way. The new micro's will be 2 to 5 times faster than the 8-bit processors, and be able to access up to 1000 times the memory.

The speed advantage of the 16-bit and 32-bit chips is not very large if floating point numbers must still be processed with software subroutines. Software floating point routines are about 1000 times slower than large-scale computer hardware. But now INTEL and others are bringing out hardware co-processor chips which implement the floating-point math. They are 100 times faster than software emulation.

The ability to address significantly more than 64K of memory space brings on the need for memory management techniques. Some manufacturers will offer memory mapping, memory protection, virtual memory, and segmented memory. From the standpoint of an application program, it is most useful to have directly accesible memory. Virtual memory is the second choice. Memory protection and memory mapping are necessary in a multi-tasking environment, or in a timesharing system.

Great new products are foreseen in memories, too. You know that the Apple II's memory chips are 16K chips; it takes 8 of them to make 16K bytes, and 32 to make 64K bytes. Well, there are now 64K memory chips; it would only take 8 of them to get 64K bytes. Of course, the Apple II would have to modified or redesigned to make use of them. The Apple III is designed to accept them, I think.

Bubble memories are also available, with 1,000,000 bits per device. These memories operate like little solid state disk drives, and their best application would be as the "roll in/roll out" device for a virtual memory system. They are faster than mechanical disk drives: in the time it takes a moving arm disk to begin to read or write the first byte of data, a bubble memory will have already transferred 4K to 16K bytes of data. Bubble memory technology is still new, so they have a high price. In 3 or 4 years they will be inexpensive enough to put into personal computers.

I can hardly wait to get my first Apple Umpteenth, with 32-bit architecture, a 50 MHz clock, hardware floating point math (25-digit precision), ten million bytes of bubble memory, one million bytes of RAM, built-in peripherals including a printer, 4 disk drives, and a CRT...and it will probably fit in my pocket!

<<<written circa 1980>>>

==============================================================================
DOCUMENT :AAL-8102:Articles:GRAM.Hello.AS.txt
==============================================================================

Two Boots Are Better Than One..............Bob Sander-Cederlof

If you have been trying to write programs for the whole Apple community, or just for yourself and a few friends, then you have probably run into the problem. Your friends or customers do not all have the same kind of Apple! Some of them have the plain old Apple II, and only have Integer BASIC. Others have the newer Apple II Plus, and only have Applesoft BASIC in ROM. (Of course, there are some who have both BASICs, either in ROM or with the Pascal Language System.

The problem is that the boot program, or the so-called HELLO program, must be in either Integer BASIC or Applesoft. It cannot be both at once! So if you use an Applesoft version, the friend without Applesoft gets the "LANGUAGE NOT AVAILABLE" message when he boots up the disk. Or if you use an Integer BASIC boot program, the person with an Apple II Plus and no Integer BASIC gets the message.

There is an answer! I discovered it by reading the documentation that comes with the Apple Writer Text Editing System. The key is to remember that if the boot program is written in Applesoft, and if furthermore there is no Applesoft in ROM in your machine, then DOS tries to load and run an Integer BASIC file with the name APPLESOFT! So, INIT your disk with an Applesoft boot program named HELLO; then include on the disk also a similar boot program written in Integer BASIC and store it on the disk under the file name "APPLESOFT"!

When you boot this disk, DOS will try to boot the program named HELLO. If you have Applesoft on ROM, this will succeed, and you are up and running. If you do not have Applesoft, DOS will attempt to load it from the disk by RUNning the Integer BASIC file named Applesoft (which is really your other boot program!!). Isn't the Apple wonderful?

==============================================================================
DOCUMENT :AAL-8102:Articles:Multiply.6502.txt
==============================================================================

Multiplying on the 6502

Brooke Boering wrote an excellent article, "Multiplying on the 6502", in MICRO--The 6502 Journal, December, 1980, pages 71-74. If you are wondering how to do it, or you want a faster routine for a special application, look up that article.

Brooke begins by explaining and timing the multiply subroutine found in the old Apple Monitor ROM. The time to multiply two 16-bit values and get a 32-bit result varies from 935 to 1511 microseconds, depending on how many "1" bits are in the multiplier. He proceeds to modify that subroutine to cut the execution time by 40%!

Finally, he presents two limited versions which are still quite useful in some applications. His 8x16 multiply averages only 383 microseconds, and his 8x8 version averages 192 microseconds.

Here is the code for his 16x16 version, which averages 726 microseconds. It has the same setup as the routine in the Apple ROM. On entry, the multiplicand should be in AUXL,AUXH ($54,55); the multiplier should be in ACL,ACH ($50,51); whatever is in XTNDL,XTNDH ($52,53) will be added to the product. Normally, XTNDL and XTNDH should be cleared to zero before starting to multiply. However, I have used this routine to convert from decimal to binary; I put the next digit in XTNDL and clear XTNDH, and then multiply the previous result by ten. The "next digit" is automatically added to the product that way. (I have corrected the typographical error in the listing as published in MICRO.)

<<<code here>>>

I wrote a test routine for the multiply, so that I could check it out. After assembling the whole program, I typed "MGO SETUP.Y" to link the control-Y Monitor Command to my test routine. Control-Y will parse three 16-bit hexadecimal values this way: val1<val2.val3cY stores val1 in $42,$43; val2 in $3C,$3D; and val3 in $3E,$3F. ("cY" stands for control-Y.)

I define val1 to be the initial value for XTNDL,XTNDH; this should normally be zero. The two values to be multiplied are val2 and val3. After TESTMPY receives control from the control-Y processor, it moves the three values into the right locations for the multiply subroutine. Then JSR RMUL calls the multiply routine. The following lines (1570-1640) print the 32-bit result by calling a routine in the monitor ROM which prints a byte in hex from the A-register.

```
==============================================================================
DOCUMENT :AAL-8102:DOS3.3:Demo.Str.Swap.txt
==============================================================================
```

```
.dâ:ó:∫"DEMO USE OF 'STRING SWAP' ROUTINE"ZnÜA$(20):∫Á(4)"BLOAD
B.STRING.SWAP":å768xxÅI–1¡20:áA$(I):Ç:P–1:∞200
     ÇÉAMES,BURKE,PUTNEY,LEE,LEVY,RAMSDELL,BISHOP,RANDALL,LANDSMAN,LEI
PER,OSLISLO,KOVACS,MEADOR,KRIEGSMAN,MERCIER,WHITE,LEVY,BLACK,SCHORNAK,
STITT     å≤BUBBLE SORT"  ñM–20a    †M–M…1:SW–0:ÅI–1¡M:≠A$(I»1)—
A$(I)ƒSW–1:ØA$(I»1),A$(I):≤SWAPo    ™Ç:≠SWƒ160     ¥P–20:∞200:Äú
     »¢3:ÅI–1¡20:ñP:∫A$(I):Ç:±
```

=============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.APPLE.BELL.txt
=============================================================================

```
1000  *---------------------------------
1010  *      APPLE "BELL" ROUTINE
1020  *---------------------------------
1030         .OR $FBE2    IN MONITOR ROM
1040         .TA $800
1050  *---------------------------------
1060  WAIT       .EQ $FCA8    MONITOR DELAY ROUTINE
1070  SPEAKER    .EQ $C030
1080  *---------------------------------
1090  M.FBE2 LDY #192     # OF HALF-CYCLES
1100  BELL2  LDA #12      SET UP DELAY OF 500 MICROSECONDS
1110         JSR WAIT     FOR A HALF CYCLE OF 1000 HERTZ
1120         LDA SPEAKER  TOGGLE SPEAKER
1130         DEY          COUNT THE HALF CYCLE
1140         BNE BELL2    NOT FINISHED
1150         RTS
```

```
===============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.INCH.WORM.txt
===============================================================================
```

```
 1000  *---------------------------------
 1010  *       INCH-WORM SOUNDS
 1020  *---------------------------------
 1030  SPEAKER    .EQ $C030
 1040  PULSE.WIDTH .EQ $00
 1050  PULSE.STEP  .EQ $01
 1060  PULSE.LIMIT .EQ $02
 1070  *---------------------------------
 1080  INCH.WORM
 1090         LDA #1       SET STEP TO 1
 1100  *   (ALSO TRY 77, 129, 179)
 1110         STA PULSE.STEP
 1120         LDA #176     SET PULSE.WIDTH AND LIMIT TO 176
 1130  *   (ALSO TRY 88)
 1140         STA PULSE.WIDTH
 1150         STA PULSE.LIMIT
 1160  .1     LDA SPEAKER  TOGGLE SPEAKER
 1170         LDX PULSE.WIDTH  DELAY LOOP FOR PULSE WIDTH
 1180  .2     PHA          LONGER DELAY LOOP
 1190         PLA
 1200         DEX          END OF PULSE?
 1210         BNE .2       NO
 1220         CLC          CHANGE PULSE WIDTH BY STEP
 1230         LDA PULSE.WIDTH
 1240         ADC PULSE.STEP
 1250         STA PULSE.WIDTH
 1260         CMP PULSE.LIMIT  UNTIL IT REACHES THE LIMIT
 1270         BNE .1
 1280         RTS
```

```
===============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.LASER.BLAST.txt
===============================================================================

 1000  *--------------------------------
 1010  *      ANOTHER LASER BLAST
 1020  *--------------------------------
 1030  SPEAKER    .EQ $C030
 1040  *--------------------------------
 1050  BLAST  LDY #10      NUMBER OF SHOTS
 1060  .1     LDX #64      PULSE WIDTH OF FIRST PULSE
 1070  .2     TXA          START A PULSE WITHIN A SHOT
 1090  .3     DEX          DELAY FOR ONE PULSE
 1100         BNE .3
 1105         TAX
 1110         LDA SPEAKER  TOGGLE SPEAKER
 1120         INX
 1130         CPX #192     PULSE WIDTH OF LAST PULSE
 1140         BNE .2
 1150         DEY          FINISHED SHOOTING?
 1160         BNE .1       NO
 1170         RTS
```

```
=======================================================================
DOCUMENT :AAL-8102:DOS3.3:S.LASER.SWOOP.txt
=======================================================================

 1000  *---------------------------------
 1010  *      LASER "SWOOP" SOUND
 1020  *---------------------------------
 1030  SPEAKER    .EQ $C030
 1040  PULSE.COUNT .EQ $00
 1050  PULSE.WIDTH .EQ $01
 1060  SWOOP.COUNT .EQ $02
 1070  *---------------------------------
 1080  SWOOP  LDA #1       ONE PULSE AT EACH WIDTH
 1090         STA PULSE.COUNT
 1100         LDA #160     START WITH MAXIMUM WIDTH
 1110  *  (ALSO TRY VALUES OF 40, 80, 128, AND 160.)
 1120         STA PULSE.WIDTH
 1130  .1     LDY PULSE.COUNT
 1140  .2     LDA SPEAKER  TOGGLE SPEAKER
 1150         LDX PULSE.WIDTH
 1160  .3     DEX          DELAY LOOP FOR ONE PULSE
 1170         BNE .3
 1180         DEY          LOOP FOR NUMBER OF PULSES
 1190         BNE .2       AT EACH PULSE WIDTH
 1200         DEC PULSE.WIDTH  SHRINK PULSE WIDTH
 1210         BNE .1       TO LIMIT OF ZERO
 1220         RTS
 1230  *---------------------------------
 1240  *      MULTI-SWOOPER
 1250  *---------------------------------
 1260  SWOOP2 LDA #10      NUMBER OF SWOOPS
 1270         STA SWOOP.COUNT
 1280  .1     JSR SWOOP
 1290         DEC SWOOP.COUNT
 1300         BNE .1
 1310         RTS
```

```
======================================================================
DOCUMENT :AAL-8102:DOS3.3:S.MACHINE.GUN.txt
======================================================================

 1000  *---------------------------------
 1010  *      MACHINE-GUN NOISE
 1020  *---------------------------------
 1030  SPEAKER    .EQ $C030
 1040  CNTR       .EQ $00
 1050  *---------------------------------
 1060  NOISE  LDX #64      LENGTH OF NOISE BURST
 1070  *---------------------------------
 1080         LDA #10      NUMBER OF NOISE BURSTS
 1090         STA CNTR
 1100  .2     LDA SPEAKER  TOGGLE SPEAKER
 1110         LDY $BA00,X  GET PULSE WIDTH PSEUDO-RANDOMLY
 1120  .1     DEY          DELAY LOOP FOR PULSE WIDTH
 1130         BNE .1
 1140         DEX          GET NEXT PULSE OF THIS NOISE BURST
 1150         BNE .2
 1160         DEC CNTR     GET NEXT NOISE BURST
 1170         BNE .2
 1180         RTS          RETURN
```

```
==========================================================================
DOCUMENT :AAL-8102:DOS3.3:S.MORSE.CODE.txt
==========================================================================

 1000  *--------------------------------
 1010  *      MORSE CODE OUTPUT
 1020  *--------------------------------
 1030  SPEAKER    .EQ $C030
 1040  DUMMY      .EQ $C000
 1050  *--------------------------------
 1060  SAVEX  .BS 1
 1070  SAVEY  .BS 1
 1080  EL.COUNT .BS 1
 1090  EL.CODE  .BS 1
 1100  EL.SPEED .EQ 120
 1110  EL.PITCH .EQ 80
 1120  *--------------------------------
 1130  CODES  .HS FD7D3D1D0D0585C5E5F5  0, 1-9
 1140         .HS 000000000000
 1150         .HS 004284A4830124C3040274A344C2  @, A-M
 1160         .HS 82E364D443038123146394B4C4    N-Z
 1170         .HS 000000000000
 1180  *--------------------------------
 1190  SETUP.MORSE
 1200         LDA #MORSE
 1210         STA $36
 1220         LDA /MORSE
 1230         STA $37
 1240         JMP $3EA
 1250  *--------------------------------
 1260  MORSE  CMP #$B0     SEE IF PRINTING CHAR
 1270         BCC .1       NO
 1280         PHA          SAVE CHAR ON STACK
 1290         JSR SEND.CHAR
 1300         PLA          GET CHAR OFF STACK
 1310  .1     JMP $FDF0
 1320  *--------------------------------
 1330  SEND.CHAR
 1340         STX SAVEX
 1350         STY SAVEY
 1360         SEC
 1370         SBC #$B0
 1380         TAX
 1390         LDA CODES,X
 1400         STA EL.CODE
 1410         AND #7       GET ELEMENT COUNT
 1420         BEQ .4       NO CODE
 1430         STA EL.COUNT
 1440  .1     ASL EL.CODE   PUT NEXT ELEMENT INTO CARRY
 1450         BCC .2       MAKE 'DIT'
 1460         JSR EL.DIT   MAKE 'DAH' FROM 3 DITS
 1470         JSR EL.DIT
 1480  .2     JSR EL.DIT   MAKE 'DIT'
```

```
1490        JSR EL.SPACE
1500        DEC EL.COUNT
1510        BNE .1
1520  .3    JSR CH.SPACE
1530        LDX SAVEX
1540        LDY SAVEY
1550        RTS
1560  .4    JSR CH.SPACE
1570        JSR CH.SPACE
1580        JMP .3
1590  *---------------------------------
1600  CH.SPACE
1610        JSR EL.SPACE
1620        JSR EL.SPACE
1630  EL.SPACE
1640        LDY #EL.SPEED
1650  .1    LDX #EL.PITCH
1660        LDA DUMMY
1670  .2    DEX
1680        BNE .2
1690        DEY
1700        BNE .1
1710        RTS
1720  *---------------------------------
1730  EL.DIT LDY #EL.SPEED
1740  .1    LDX #EL.PITCH
1750        LDA SPEAKER
1760  .2    DEX
1770        BNE .2
1780        DEY
1790        BNE .1
1800        RTS
```

```
==============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.MULTIPLY.txt
==============================================================================

 1000  *---------------------------------
 1010  *      FASTER 16X16 MULTIPLY
 1020  *      BY BROOKE W. BOERING
 1030  *      NEARLY AS PUBLISHED IN MICRO--THE 6502 JOURNAL
 1040  *      PAGE 72, DECEMBER, 1980.
 1050  *---------------------------------
 1060  ACL    .EQ $50
 1070  ACH    .EQ $51
 1080  XTNDL  .EQ $52
 1090  XTNDH  .EQ $53
 1100  AUXL   .EQ $54
 1110  AUXH   .EQ $55
 1120  *---------------------------------
 1130  RMUL   LDY #16      16-BIT MULTIPLIER
 1140  .1     LDA ACL      (AC * AUX) + XTND
 1150         LSR          CHECK NEXT BIT OF MULTIPLIER
 1160         BCC .2       IF ZERO, DON'T ADD MULTIPLICAND
 1170         CLC          ADD MULTIPLICAND TO PARTIAL PRODUCT
 1180         LDA XTNDL
 1190         ADC AUXL
 1200         STA XTNDL
 1210         LDA XTNDH
 1220         ADC AUXH
 1230         STA XTNDH
 1240  .2     ROR XTNDH    SHIFT PARTIAL PRODUCT
 1250         ROR XTNDL
 1260         ROR ACH
 1270         ROR ACL
 1280         DEY          NEXT BIT
 1290         BNE .1       UNTIL ALL 16
 1300         RTS
 1310  *---------------------------------
 1320  *      TEST ROUTINE FOR MULTIPLY
 1330  *---------------------------------
 1340  SETUP.Y
 1350         LDA #$4C     PUT "JMP TESTMPY" IN $358-35A
 1360         STA $3F8
 1370         LDA #TESTMPY
 1380         STA $3F9
 1390         LDA /TESTMPY
 1400         STA $3FA
 1410         RTS
 1420  *---------------------------------
 1430  TESTMPY
 1440         LDA $3C      MOVE A1L,A1H TO ACL,ACH
 1450         STA ACL
 1460         LDA $3D
 1470         STA ACH
 1480         LDA $3E      MOVE A2L,A2H TO AUXL,AUXH
```

```
1490        STA AUXL
1500        LDA $3F
1510        STA AUXH
1520        LDA $42      MOVE A4L,A4H TO XTNDL,XTNDH
1530        STA XTNDL
1540        LDA $43
1550        STA XTNDH
1560        JSR RMUL     MULTIPLY
1570        LDA XTNDH    PRINT 32-BIT RESULT
1580        JSR $FDDA
1590        LDA XTNDL
1600        JSR $FDDA
1610        LDA ACH
1620        JSR $FDDA
1630        LDA ACL
1640        JMP $FDDA
```

```
==============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.SIMPLE.TONE.txt
==============================================================================
```

```
 1000  *---------------------------------
 1010  *      SIMPLE TONE
 1020  *---------------------------------
 1030  SPEAKER    .EQ $C030
 1040  *---------------------------------
 1050  TONE   LDY #0       START CYCLE COUNTER
 1060         LDX #0       START DELAY COUNTER
 1070  .1     LDA SPEAKER  TOGGLE SPEAKER
 1080  .2     DEX          DELAY LOOP
 1090         BNE .2
 1100         DEY          QUIT AFTER 128 CYCLES
 1110         BNE .1
 1120         RTS
```

```
==============================================================
DOCUMENT :AAL-8102:DOS3.3:S.STRING.SWAP.txt
==============================================================

 1000  *---------------------------------
 1010  *      STRING SWAP FOR APPLESOFT
 1020  *      "BRUN B.STRING.SWAP" TO SET IT UP;
 1030  *      THEN "&A$,B$" MEANS SWAP A$ AND B$.
 1040  *---------------------------------
 1050         .OR $300
 1060         .TF B.STRING.SWAP
 1070  *---------------------------------
 1080  AMPERSAND.VECTOR    .EQ $3F5
 1090  *---------------------------------
 1100  PTRGET              .EQ $DFE3   SCAN FOR VARIABLE NAME,
 1110  *                               SEARCH FOR ITS ADDRESS,
 1120  *                               LEAVE ADDRESS IN $83,$84
 1130  *                               AND A,Y
 1140  *---------------------------------
 1150  SCAN.COMMA          .EQ $DEBE   IF NEXT CHARACTER IS
 1160  *                               IS A COMMA, SCAN OVER
 1170  *                               IT; IF NOT, SYNTAX ERROR.
 1180  *---------------------------------
 1190  A.PNTR .EQ $85,86
 1200  B.PNTR .EQ $83,84
 1210  *---------------------------------
 1220  SETUP  LDA #SWAP    SET UP AMPERSAND VECTOR
 1230         STA AMPERSAND.VECTOR+1
 1240         LDA /SWAP
 1250         STA AMPERSAND.VECTOR+2
 1260         LDA #$4C     JMP OPCODE
 1270         STA AMPERSAND.VECTOR
 1280         RTS
 1290  *---------------------------------
 1300  SWAP   JSR PTRGET   GET POINTER TO FIRST STRING
 1310         STA A.PNTR
 1320         STY A.PNTR+1
 1330         JSR SCAN.COMMA  CHECK FOR COMMA
 1340         JSR PTRGET
 1350         LDY #2       PREPARE TO SWAP 3 BYTES
 1360  .1     LDA (A.PNTR),Y
 1370         PHA
 1380         LDA (B.PNTR),Y
 1390         STA (A.PNTR),Y
 1400         PLA
 1410         STA (B.PNTR),Y
 1420         DEY          NEXT BYTE
 1430         BPL .1
 1440         RTS          RETURN
```

```
==============================================================================
DOCUMENT :AAL-8102:DOS3.3:S.TOUCH.TONES.txt
==============================================================================

 1000  *-----------------------------------
 1010  *      TOUCH TONES SIMULATOR
 1020  *-----------------------------------
 1030  SPEAKER    .EQ $C030
 1040  *-----------------------------------
 1050  DOWNTIME   .EQ $9D
 1060  UPTIME     .EQ $9E
 1070  LENGTH     .EQ $9F
 1080  CHORD.TIME .EQ $A0
 1090  *-----------------------------------
 1100  BUTTON     .EQ $E7  SET BY "SCALE= # "
 1110  *                   USE VALUES FROM 0 THRU 9
 1120  *-----------------------------------
 1130         .OR $300
 1140  *-----------------------------------
 1150  TWO.TONES
 1160         LDA #10
 1170         STA CHORD.TIME
 1180  .3     LDX BUTTON
 1190         LDA LOW.TONES,X
 1200         JSR ONE.TONE
 1210         LDA HIGH.TONES,X
 1220         JSR ONE.TONE
 1230         DEC CHORD.TIME
 1240         BNE .3
 1250         RTS
 1260  *-----------------------------------
 1270  ONE.TONE
 1280         TAY
 1290         LDA DOWNTIME.TABLE,Y
 1300         STA DOWNTIME
 1310         LDA UPTIME.TABLE,Y
 1320         STA UPTIME
 1330         LDA LENGTH.TABLE,Y
 1340         STA LENGTH
 1350  *-----------------------------------
 1360  PLAY   LDY UPTIME
 1370         LDA SPEAKER
 1380         DEC LENGTH
 1390         BEQ .4       FINISHED
 1400  .1     DEY
 1410         BNE .1
 1420         BEQ .2
 1430  .2     LDY DOWNTIME
 1440         LDA SPEAKER
 1450         DEC LENGTH
 1460         BEQ .4
 1470  .3     DEY
 1480         BNE .3
```

```
1490        BEQ PLAY
1500 .4     RTS
1510 *---------------------------------
1520 DOWNTIME.TABLE
1530        .HS 8E807468514942
1540 *---------------------------------
1550 UPTIME.TABLE
1560        .HS 8E807469514942
1570 *---------------------------------
1580 LENGTH.TABLE
1590        .HS 1412100F201D1A
1600 *---------------------------------
1610 LOW.TONES
1620        .HS 030000000010101020202
1630 HIGH.TONES
1640        .HS 0504050604050604050 6
1650 *---------------------------------
1660 *      SIMULATED DRIVER
1670 *---------------------------------
1680 MON.WAIT   .EQ $FCA8
1690 PUNCH.ALL
1700        LDA #0
1710        STA BUTTON
1720 .1     JSR TWO.TONES
1730        LDA #0
1740        JSR MON.WAIT
1750        INC BUTTON
1760        LDA BUTTON
1770        CMP #10
1780        BCC .1
1790        RTS
```

==============================================================================
DOCUMENT :AAL-8103:Articles:A.Beaut.Dump.txt
==============================================================================

A Beautiful Dump . . . . . . . . . . . . . . . . Robert H. Bernard

The old saying, "You can't tell the players without a scorecard," is certainly true for program debugging, and sometimes the only way is to look into memory and see what is there. The Apple II Monitor has a memory dump command, but I found it inadequate: it's formatted for a 40-column screen, it doesn't show ASCII codes, and getting output on a printer is a hassle.

So I sat down and wrote a quick assembly language memory dump modeled after a System/360 core dump (remember when computer memory was called "core"?), with both hex and ASCII. My first attempt took up more than one page of memory and was trapped where I assembled it by absolute internal references. I massaged it until it fit in less than a page and made it relocatable ("run anywhere") by making all internal jumps into relative branches. (A "page" in 6502 jargon is 256 bytes, with addresses running from xx00 through xxFF.)

Next I decided to add a printer feature; while I was at it I made it use 80 columns on the printer, 40 on the screen.

Next I made it print the bytes in groups of four, with a space between every four bytes. Sixteen bytes are printed per line on the screen, 32 on an 80-column printer. Spacing in groups of four makes it easier to spot certain address locations. If a byte value is a printable ASCII code, I print the character above the hexadecimal value. (Values $00-$1F and $80-$9F do not print.)

Then I wanted options to browze one screenful at a time, and backup when I passed the place I wanted to look at.

You probably think that by now the program is at least two, and maybe more, pages long. Not so! All the while I was able to keep it in only one page (which doesn't say much for my original code).

The end result (after 21 versions!) is listed here for your examination and pleasure.

Operating Instructions: BRUN the program anywhere in memory that you have a free page (256 bytes). When the "?" prompt appears, enter the address of the memory you want to dump in any of the following ways. After the address or address range, type the return key.

```
S.E         To dump memory from S to E on the screen.

S-E         To dump memory from S to E on the printer.

S,E         To dump memory from S to E on the screen,
            but pauses after each screenful;
```

```
             press space bar to continue,
             or press control-C to stop.

S            To dump from S, pausing after each line;
             press space bar to dump next line,
             press letter "B" to back up one line,
             or press control-C to stop.
```

================================================================================
DOCUMENT :AAL-8103:Articles:Amper.Cmd.Int.txt
================================================================================

& Command Interface for S-C Assembler II

Here is yet another way to add new commands to Version 4.0. You are somewhat familiar with the use of the & in Applesoft. This little program patches the assembler so that you can add as many new commands as you wish.

I have shown as examples the EDIT, COPY, and SYM commands. You need to fill in the correct starting address in lines 1250 and 1260.

Use the .TF directive to direct the object code to a file. Then use BRUN to install the patch. Lines 1100-1120 patch the assembler to hook in the code at lines 3010-3100. After it is hooked in, make a new copy of the assembler by using BSAVE ASMDISK 4.0 WITH &,A$FD7,L$.... (Fill in the appropriate length, depending on what else you have added to the assembler in the past.)

==============================================================================
DOCUMENT :AAL-8103:Articles:DOS321.RWTS.Lst.txt
==============================================================================

Commented Listing of DOS 3.2.1 RWTS

I promised in the original AAL flyer that I would print dis-assemblies of things like DOS. Here is the first installment. RWTS is described in some detail in the DOS Reference Manual, pages 94-98.

There are not too many differences between the various versions of RWTS. Each one, from 3.1 to 3.2 to 3.2.1 to 3.3, seems mainly to clean up errors of the previous ones. I will probably print some DOS 3.3 listings in the future, as well as more of 3.2.1.

There is a bug in the 3.2.1 version (a bad address), at line 2200. It works anyway, but it is sloppy. Another problem I have discovered the hard way: the "previous slot #" in the IOB should be a slot that has a disk controller in it. If not, RWTS may do strange things to whatever is in that slot. I put in "0", and it turned on my language card! Zap! No more Applesoft!

```
=============================================================================
DOCUMENT :AAL-8103:Articles:Front.Page.txt
=============================================================================
```

The Apple Assembly Line is still growing!  I now am sending out over 300 copies per month!  It is also growing in size, as you can see: this is the first 20 page issue.

In This Issue...



Second "Disk of the Quarter"

The second AALDQ is ready!  If you would like to have the source code on disk in S-C Assembler II Version 4.0 format for all the programs which have appeared in AAL issues 4, 5, and 6, then send me $15.  I will send you the disk, and you already have the documentation.  DQ#1, covering issues 1, 2, and 3, is also still available at the same price.

Some New Books about the 6502

Apple Machine Language, by Don Inman and Kurt Inman, published by Reston (a Prentice-Hall Company).  Hard cover, 296 pages, $14.95.  If you are an absolute beginner, this is the book for you.  You start by typing in an Applesoft program which helps you POKE in machine language code, and CALL it.  Most of the examples involve lo-res graphics and sound.  One chapter describes the Apple Mini-Assembler (which resides in the Integer BASIC ROMs).  They never get around to a real assembler.

Practical Microcomputer Programming: the 6502, by W. J. Weller, published by Northern Technology Books.  Hard cover, 459 pages, $32.95.  Over 110 pages of the book are devoted to a listing of an assembler and a debugging package.  A coupon inside the back cover can be redeemed for a tape copy which will run on the Apple II.  By adding $7.50 to the coupon, you can get a disk version.  The package can be loaded from the disk, but there is no capability for keeping source or object files on disk.

```
==============================================================================
DOCUMENT :AAL-8103:Articles:Opcode.Chart.txt
==============================================================================

    x0      x1          x2      x3

0x  BRK     ORA (z,X)   hang    ASL (z,X)
                                ORA (z,X)

1x  BPL r   ORA (z),Y   hang    ASL (z),Y
                                ORA (z),Y

2x  JSR a   AND (z,X)   hang    ROL (z,X)
                                AND (z,X)

3x  BMI r   AND (z),Y   hang    ROL (z),Y
                                AND (z),Y

4x  RTI     EOR (z,X)   hang    LSR (z,X)
                                EOR (z,X)

5x  BVC r   EOR (z),Y   hang    LSR (z),Y
                                EOR (z),Y

6x  RTS     ADC (z,X)   hang    ROR (z,X)
                                ADC (z,X)

7x  BVS r   ADC (z),Y   hang    ROR (z),Y
                                ADC (z),Y

8x  nop2    STA (z,X)   nop2    A&X
                                --> (z,X)

9x  BCC r   STA (z),Y   hang    A&hea
                                --> (z),Y

Ax  LDY #v  LDA (z,X)   LDX #v  LDX #v
                                LDA (z,X)
                                LDX (z,X)

Bx  BCS r   LDA (z),Y   hang    LDA (z),Y
                                LDX (z),Y

Cx  CPY #v  CMP (z,X)   nop2    DEC (z,X)
                                CMP (z,X)

Dx  BNE r   CMP (z),Y   hang    DEC (z),Y
                                CMP (z),Y

Ex  CPX #v  SBC (z,X)   nop2    INC (z,X)
                                SBC (z,X)

Fx  BEQ r   SBC (z),Y   hang    INC (z),Y
```

```
                    SBC (z),Y

x4        x5        x6        x7

nop2      ORA z     ASL z     ASL z
                              ORA z

nop2      ORA z,X   ASL z,X   ASL z,X
                              ORA z,X

BIT z     AND z     ROL z     ROL z
                              AND z

nop2      AND z,X   ROL z,X   ROL z,X
                              AND z,X

nop2      EOR z     LSR z     LSR z
                              EOR z

nop2      EOR z,X   LSR z,X   LSR z,X
                              EOR z,X

nop2      ADC z     ROR z     ROR z
                              ADC z

nop2      ADC z,X   ROR z,X   ROR z,X
                              ADC z,X

STY z     STA z     STX z     A&X
                              --> z

STY z,X   STA z,X   STX z,Y   A&X
                              --> z,Y

LDY z     LDA z     LDX z     LDX z
                              LDA z

LDY z,X   LDA z,X   LDX z,Y   LDX z,Y
                              LDA z,Y

CPY z     CMP z     DEC z     DEC z
                              CMP z

nop2      CMP z,X   DEC z,X   DEC z,X
                              CMP z,X

CPX z     SBC z     INC z     INC z
                              SBC z

nop2      SBC z,X   INC z,X   INC z,X
                              SBC z,X

x8   x9        xA   xB
```

```
PHP   ORA #v   ASL   AND #v

CLC   ORA a,Y  nop   ASL a,Y
                     ORA a,Y

PLP   AND #v   ROL   AND #v

SEC   AND a,Y  nop   ROL a,Y
                     AND a,Y

PHA   EOR #v   LSR   AND #v
                     LSR

CLI   EOR a,Y  nop   LSR a,Y
                     EOR a,Y

PLA   ADC #v   ROR   AND #v
                     ROR

SEI   ADC a,Y  nop   ROR a,Y
                     ADC a,Y

DEY   nop2     TXA   #v&X
                     --> A

TYA   STA a,Y  TXS   A&X-->S
                     S&hea+1
                     --> a,Y

TAY   LDA #v   TAX   LDA #v
                     TAX

CLV   LDA a,Y  TSX   a,Y & S
                     -->AXS

INY   CMP #v   DEX   A&X-#v
                     --> X

CLD   CMP a,Y  nop   DEC a,Y
                     CMP a,Y

INX   SBC #v   NOP   SBC #v

SED   SBC a,Y  nop   INC a,Y
                     SBC a,Y

xC       xD       xE       xF

nop3     ORA a    ASL a    ASL a
                           ORA a

nop3     ORA a,X  ASL a,X  ASL a,X
                           ORA a,X
```

```
BIT a    AND a    ROL a    ROL a
                           AND a

nop3     AND a,X  ROL a,X  ROL a,X
                           AND a,X

JMP a    EOR a    LSR a    LSR a
                           EOR a

nop3     EOR a,X  LSR a,X  LSR a,X
                           EOR a,X

JMP (a)  ADC a    ROR a    ROR a
                           ADC a

nop3     ADC a,X  ROR a,X  ROR a,X
                           ADC a,X

STY a    STA a    STX a    A&X
                           --> a

nop3     STA a,X  X&hea+1  A&X
                  --> a,Y  --> a,X

LDY a    LDA a    LDX a    LDX a
                           LDA a

LDY a,X  LDA a,X  LDX a,Y  LDX a,Y
                           LDA a,Y

CPY a    CMP a    DEC a    DEC a
                           CMP a

nop3     CMP a,X  DEC a,X  DEC a,X
                           CMP a,X

CPX a    SBC a    INC a    INC a
                           SBC a

nop3     SBC a,X  INC a,X  INC a,X
                           SBC a,X

A     A-register (Accumulator)
S     S-register (Stack Pointer)
X     X-register
Y     Y-register

a     2-byte absolute address
r     1-byte relative address
v     1-byte immediate value
z     1-byte pagezero address

hea   high-byte of effective address
      93:  the byte at z+1
```

```
      9B:  3rd byte of instruction
      9E:  3rd byte of instruction

&     and-function (logical product)

hang  computer hangs up, only way to
      regain control is to hit RESET

nop   1-byte instruction, no operation
nop2  2-byte instruction, no operation
nop3  3-byte instruction, no operation

-->   "result is stored in"
```

=================================================================
DOCUMENT :AAL-8103:Articles:Unused.Opcodes.txt
=================================================================

So-Called Unused Opcodes

The 6502 has 104 so-called unused opcodes. The various charts and reference manuals I have checked either leave them blank or call them "unused", "no-operation", or "future expansion". The 6502 has been around since 1976; I think we have waited long enough to know there will be no "expansion". But are they really unused? Do they have any effect if we try to execute them? Are they really no-ops? If so, how many bytes does the processor assume for each one?

These questions had never bothered me until I was looking through some disassembled memory and thought I found evidence of someone USING the "unused". It turned out they were not, but my curiosity was aroused. Just for fun, I built a little test routine and tried out the $FF opcode. Lo and behold! The 6502 thinks it is a 3-byte instruction, and it changes the A-register and some status bits!

About 45 minutes later I pinned it down: FFxxyy performs exactly the same as the two instructions FExxyy and FDxxyy. It is just as though I had executed one and then the other. In other words, anywhere in a program I find:

```
     INC VARIABLE,X
     SBC VARIABLE,X
```

I can substitute:

```
     .HS FF
     .DA VARIABLE
```

You might wonder if I will ever find that sequence. I did try writing a program to demonstrate its use. It has the advantage of saving 3 bytes, and 4 clock cycles. (The SBC instruction is executed DURING the 7 cycles of the INC instruction!)

```
     TEST LDX INDEX
          LDA #10       FOR COUNTER(X)=10 TO 39
          STA COUNTER,X
     .1   LDA COUNTER,X GET COUNTER(X)
          JSR $FDDA     PRINT IT OUT (OR WHATEVER)
          LDA #39       LIMIT
          .HS FF        DO INC AND SBC
          .DA COUNTER   ON COUNTER,X
          BCS .1        NEXT
          RTS
```

Are there any more? Before I could rest my curiosity, I had spent at least ten more hours, and had figured out what all 104 "unused opcodes" really do!

The center-fold chart shows the fruit of my detective work. The shaded opcodes are the "unused" ones. I don't know if every 6502 behaves the same as mine or not. Mine appears to be made by Synertek, and has a date code of 7720 (20th week of 1977). It could be that later versions or chips from other sources (MOS Technology or Rockwell) are different. If you find yours to be different, please let me know!

Twelve of the opcodes, all in column "x2", hang up the 6502; the only way to get out is to hit RESET or turn off the machine.

There are 27 opcodes which appear to have no effect on any registers or on memory. These could be called "NOP", but some of them are considered by the 6502 to have 2 or 3 bytes. I have labeled them "nop", "nop2", and "nop3" to distinguish how many bytes the 6502 thinks it is using. You could call nop2 "always skip one byte" and nop3 "always skip two bytes".

The action most of the rest perform can be deduced by looking at the other opcodes in the same row. For example, all of the xF column (except 8F and 9F) perform two instructions together: first the corresponding xE opcode, and then the corresponding xD opcode. In the same way, most of the opcodes in column x7 combine the x6 and x5 opcodes. The x3 column mirrors the x7 and xF columns, but with different addressing modes. And finally, the xB column mimics the other three columns, but with more exceptions. Most of the exceptions are in the 8x and 9x rows.

A few of the opcodes seem especially interesting and potentially useful. For example, A3xx performs three steps: first it loads xx into the X-register; then using this new value of X, it moves the byte addressed by (xx,X) into both the A- and X- registers. Another way of looking at this one is to say that whatever value xx has is doubled; then the two pagezero bytes at 2*xx and 2*xx+1 are used as the address for loading the A- and X-registers. You could use this for something, couldn't you?

There are five instructions which form the logical product of the A- and X-registers (without disturbing either register) and store the result in memory. If we call this new instruction "SAX", for "Store A&X", we have:

```
     83   SAX (z,X)            8F   SAX a
     87   SAX z                9F   SAX a,X
     97   SAX z,Y
```

We get seven forms of the combination which shift a memory location using ASL, and then inclusive OR the results into A with an ORA instruction. If we call this new instruction ALO, we have:

```
     03   ALO (z,X)            1B   ALO a,Y
     13   ALO (z),Y            0F   ALO a
     07   ALO z                1F   ALO a,X
     17   ALO z,X
```

The same seven forms occur for the combinations ROL-AND, LSR-EOR, and ROR-ADC. Note that if you don't care what happens to the A-register, and the status register, these 28 instructions make two extra addressing modes available to the shift instructions: (z,X) and (z),Y.

Opcodes 4B and 6B might also be useful. You can do an AND-immediate followed by LSR or ROR on the A-register.

Opcodes 93, 9B, and 9E are really weird! It took a lot of head-scratching to figure out what they do.

```
     93   Forms the logical product of the A-register
          and byte the at z+1 (which I call "hea")
          and stores it at (z),Y.

     9B   Forms the logical product of the A- and X-
          registers, and stores the result in the S-
          register (stack pointer)!  Ouch!
          Then it takes up the third byte of the
          instruction (yy from 9B xx yy) and adds one
          to it (I call it "hea+1").  Then it forms
          the logical product of the new S-register
          and "hea+1" and stores the result at "a,Y".
          Whew!

     9E   Forms the logical product of the X-register
          and "hea+1" and stores the result at "a,Y".
```

We get six forms of the new "LAX" instruction, which loads the same value into both the A- and X-registers:

```
     B3   LAX (z),Y            AB   LAX #v
     A7   LAX z                AF   LAX a
     B7   LAX z,Y              BF   LAX a,Y
```

I skipped over BB, because it is another extremely weird one. It forms the logical product of the byte at "a,Y" and S-register, and stores the result in the A-, X-, and S-registers. No wonder they didn't tell us about it!

Right under that one is the CB instruction. Well, good buddy (please excuse the CB talk!), it forms the logical product of the A- and X-registers, subtracts the immediate value (second byte of CB xx), and puts the result into the X-register.

The Cx and Dx rows provide us with seven forms that do a DEC on a memory byte, and then CMP the result with the A-register. Likewise, the Ex and Fx rows give us seven forms that perform INC followed by SBC.

It is a good thing to be aware that the so-called "unused" opcodes can be quite dangerous if they are accidentally executed. If your program goes momentarily wild and executes some data, chances are something somewhere will get strangely clobbered.

Since all of the above information was deduced by testing and observation, I cannot be certain that I am 100% correct. I may have overlooked or mis-interpreted some results, or even made a clerical error. Furthermore, as I said before, my 6502 may be different from yours. You can test your own, to see if it works like mine.

And if the whole exercise seems academic to you, you can at least enjoy the first legible and complete hexadecimal opcode chart for the 6502.

So-Callec Unused Opcodes

The 6502 has 104 so-called unused opcodes. The various charts and reference manuals I have checkcd either leaue them blank or call them "unused", "no-operation", or "future expansion". The 6502 has been around since 1976; I think we have waited long enough to know there will be no "expansion". But are they really unused? Do they have any effect if we try to execute them? Are they really no-ops? If so, how many bytes does the processor assume for each one?

These questions had never bothered me until I was looking through some disassembled memory and thought I found evidence of someone USING the "unused". It turned out they were not, but my curiosity was aroused. Just for fun, I built a little test routine and tried out the $FF opcode. Lo and behold! The 6502 thinks it is a 3-byte instruction, and it changes the A-register and some status bits!

About 45 minutes later I pinned it down: FFxxyy performs exactly the same as the two instructions FExxyy and FDxxyy. It is just as though I had executed one and then the other. In other words, anywhere in a program I find:

```
     INC VARIABLE,X
     SBC VARIABLE,X
```

I can substitute:

```
     .HS FF
     .DA VARIABLE
```

You might wonder if I will ever find that sequence. I did try writing a program to demonstrate its use. It has the advantage of saving 3 bytes, and 4 clock cycles. (The SBC instruction is executed DURING the 7 cycles of the INC instruction!)

<show sample program using FF opcode here>

Are there any more? Before I could rest my curiosity, I had spent at least ten more hours, and had figured out what all 104 "unused opcodes" really do!

The center-fold chart shows the fruit of my detective work. The shaded opcodes are the "unused" ones. I don't know if every 6502 behaves the same as mine or not. Mine appears to be made by Synertek, and has a date code of 7720 (20th week of 1977). It could be that later versions or chips from other sources (MOS Technology or

Rockwell) are different. If you find yours to be different, please let me know!

Twelve of the opcodes, all in column "x2", hang up the 6502; the only way to get out is to hit RESET or turn off the machine.

There are 27 opcodes which appear to have no effect on any registers or on memory. These could be called "NOP", but some of them are considered by the 6502 to have 2 or 3 bytes. I have labeled them "nop", "nop2", and "nop3" to distinguish how many bytes the 6502 thinks it is using. You could call nop2 "always skip one byte" and nop3 "always skip two bytes".

The action most of the rest perform can be deduced by looking at the other opcodes in the same row. For example, all of the xF column (except 8F and 9F) perform two instructions together: first the corresponding xE opcode, and then the corresponding xD opcode. In the same way, most of the opcodes in column x7 combine the x6 and x5 opcodes. The x3 column mirrors the x7 and xF columns, but with different addressing modes. And finally, the xB column mimics the other three columns, but with more exceptions. Most of the exceptions are in the 8x and 9x rows.
A few of the opcodes seem especially interesting and potentially useful. For example, A3xx performs three steps: first it loads xx into the X-register; then using this new value of X, it moves the byte addressed by (xx,X) into both the A- and X- registers. Another way of looking at this one is to say that whatever value xx has is doubled; then the two pagezero bytes at 2*xx and 2*xx+1 are used as the address for loading the A- and X-registers. You could use this for something, couldn't you?

There are five instructions which form the logical product of the A- and X-registers (without disturbing either register) and store the result in memory. If we call this new instruction "SAX", for "Store A&X", we have:

```
     83   SAX (z,X)
     87   SAX z
     97   SAX z,Y
     8F   SAX a
     9F   SAX a,X
```

We get seven forms of the combination which shift a memory location using ASL, and then inclusive OR the results into A with an ORA instruction. If we call this new instruction ALO, we have:

```
     03   ALO (z,X)            1B   ALO a,Y
     13   ALO (z),Y            0F   ALO a
     07   ALO z                1F   ALO a,X
     17   ALO z,X
```

The same seven forms occur for the combinations ROL-AND, LSR-EOR, and ROR-ADC. Note that if you don't care what happens to the A-register, and the status register, these 28 instructions make two extra addressing modes available to the shift instructions: (z,X) and (z),Y.

Opcodes 4B and 6B might also be useful. You can do an AND-immediate followed by LSR or ROR.

Opcodes 93, 9B, and 9E are really weird! It took a lot of head-scratching to figure out what they do.

```
     93   Forms the logical product of the A-register
          and byte the at z+1 (which I call "hea")
          and stores it at (z),Y

     9B   Forms the logical product of the A- and X-
          registers, and stores the result in the S-
          register (stack pointer)!  Ouch!
          Then it takes up the third byte of the
          instruction (yy from 9B xx yy) and adds one
          to it (I call it "hea+1").  Then it forms
          the logical product of the new S-register
          and "hea+1" and stores the result at "a,Y".
          Whew!

     9E   Forms the logical product of the X-register
          and "hea+1" and stores the result at "a,Y".
```

We get six forms of the new "LAX" instruction, which loads the same value into both the A- and X-registers:

```
     B3   LAX (z),Y          AB   LAX #v
     A7   LAX z              AF   LAX a
     B7   LAX z,Y            BF   LAX a,Y
```

I skipped over BB, because it is another extremely weird one. It forms the logical product of the byte at "a,Y" and S-register, and stores the result in the A-, X-, and S-registers. No wonder they didn't tell us about it!

Right under that one is the CB instruction. Well, good buddy (please excuse the CB talk!), it forms the logical product of the A- and X-registers, subtracts the immediate value (second byte of CB xx), and puts the result into the X-register.

The Cx and Dx rows provide us with seven forms that do a DEC on a memory byte, and then CMP the result with the A-register. Likewise, the Ex and Fx rows give us seven forms that perform INC followed by SBC.

Since all of the above information was deduced by testing and observation, I cannot be certain that I am 100% correct. I may have overlooked or mis-interpreted some results, or even made a clerical error. Furthermore, as I said before, my 6502 may be different from yours. You can test your own, to see if it works like mine. And if the whole exercise seems academic to you, you can at least enjoy the first legible, complete hexadecimal opcode chart for the 6502.

```
=======================================================================
DOCUMENT :AAL-8103:DOS3.3:AsmDisk4.0.Mod.txt
=======================================================================

INT
MON C,I,O
BLOAD ASMDISK 4.0
CALL-151
C089
C089
BLOAD BMC A$D001,A$D001
BLOAD EDITASM A$D13C,A$DI3C
C08A
101C:20 CC 24
24CC:AC 88 C0 20 80 1F 60
24D3:20 D9 24 4C 26 10 A0 00 20 8D 12 20 4A 11
24E1:4C 66 10 00 00 00 00 00 00 00
1063:4C D3 24
1078:4C
1125:60 EA EA
1246:43 4F 50 00 D0
126E:45 44 49 3B D1
20D4:4C B0 24
20D7:4C C7 24
20DA:4C B5 24
24B0:A5 DB 20 FA 19 A5 DC 20 FA 19 20 8B 12 C9 2C
24BF:F0 03 4C 8E 18 4C B5 20 A5 DB 18 90 EB
1009:4C 4E 1E
NOMON C,I,O
1000G
```

```
==============================================================================
DOCUMENT :AAL-8103:DOS3.3:DOS321.BD00BE9F.txt
==============================================================================

 1000  *      .LIF
 1010  *---------------------------------
 1020  *      DOS 3.2.1 DISASSEMBLY $BD00-BE9F
 1030  *      BOB SANDER-CEDERLOF      3-3-81
 1040  *---------------------------------
 1050  CURRENT.TRACK       .EQ $478
 1060  DRIVE.1.TRACK       .EQ $478 THRU 47F (INDEX BY SLOT)
 1070  DRIVE.2.TRACK       .EQ $4F8 THRU 4FF (INDEX BY SLOT)
 1080  SEARCH.COUNT        .EQ $4F8
 1090  RETRY.COUNT         .EQ $578
 1100  SLOT                .EQ $5F8
 1110  SEEK.COUNT          .EQ $6F8
 1120  *---------------------------------
 1130  PHASE.OFF           .EQ $C080
 1140  PHASE.ON            .EQ $C081
 1150  MOTOR.OFF           .EQ $C088
 1160  MOTOR.ON            .EQ $C089
 1170  ENABLE.DRIVE.1      .EQ $C08A
 1180  ENABLE.DRIVE.2      .EQ $C08B
 1190  Q6L                 .EQ $C08C
 1200  Q6H                 .EQ $C08D
 1210  Q7L                 .EQ $C08E
 1220  Q7H                 .EQ $C08F
 1230  *---------------------------------
 1240  SECTOR     .EQ $2D
 1250  TRACK      .EQ $2E
 1260  VOLUME     .EQ $2F
 1270  DRIVE.NO   .EQ $35
 1280  DCT.PNTR   .EQ $3C,3D
 1290  BUF.PNTR   .EQ $3E,3F
 1300  MOTOR.TIME .EQ $46,47
 1310  IOB.PNTR   .EQ $48,49
 1320  *---------------------------------
 1330  PRE.NYBBLE          .EQ $B800
 1340  WRITE.SECTOR        .EQ $B86A
 1350  READ.SECTOR         .EQ $B8FD
 1360  READ.ADDRESS        .EQ $B965
 1370  POST.NYBBLE         .EQ $B9C1
 1380  SEEK.TRACK.ABSOLUTE .EQ $BA1E
 1390  *---------------------------------
 1400  ERR.WRITE.PROTECT   .EQ $10
 1410  ERR.WRONG.VOLUME    .EQ $20
 1420  ERR.BAD.DRIVE       .EQ $40
 1430  *---------------------------------
 1440         .OR $BD00
 1450         .TA $800
 1460  *---------------------------------
 1470  RWTS   STY IOB.PNTR SAVE ADDRESS OF IOB
 1480         STA IOB.PNTR+1
```

```
        LDY #2
        STY SEEK.COUNT  UP TO 2 RE-CALIBRATIONS
        LDY #4
        STY SEARCH.COUNT
        LDY #1       POINT AT SLOT# IN IOB
        LDA (IOB.PNTR),Y  SLOT# FOR THIS OPERATION
        TAX
        LDY #15      POINT AT PREVIOUS SLOT#
        CMP (IOB.PNTR),Y  SAME SLOT?
        BEQ .3       YES
        TXA          SAVE NEW SLOT ON STACK
        PHA
        LDA (IOB.PNTR),Y  GET OLD SLOT#
        TAX
        PLA          STORE NEW SLOT #
        PHA          INTO OLD SLOT# SPOT
        STA (IOB.PNTR),Y
*---------------------------------
*       SEE IF OLD MOTOR STILL SPINNING
*---------------------------------
        LDA Q7L,X    GO INTO READ MODE
.1      LDY #8       IF DATA DOES NOT CHANGE
        LDA Q6L,X       FOR 96 MICROSECONDS,
.2      CMP Q6L,X       THEN THE DRIVE IS STOPPED
        BNE .1       WOOPS! IT CHANGED!
        DEY          TIME UP YET?
        BNE .2       NO, KEEP CHECKING
        PLA          GET NEW SLOT # AGAIN
        TAX
*---------------------------------
.3      LDA Q7L,X    SET UP TO READ
        LDA Q6L,X
        LDA Q6L,X    GET CURRENT DATA
        PHA          7 CYCLE DELAY
        PLA
        STX SLOT
        CMP Q6L,X    SEE IF DATA CHANGED
        PHP          SAVE ANSWER ON STACK
        LDA MOTOR.ON,X   TURN ON MOTOR
        LDY #6       COPY POINTERS INTO PAGE ZERO
.4      LDA (IOB.PNTR),Y
        STA DCT.PNTR-6,Y
        INY          DCT.PNTR .EQ $3C,3D
        CPY #10      BUF.PNTR .EQ $3E,3F
        BNE .4
        LDY #3       GET MOTOR ON TIME FROM DCT
        LDA (DCT.PNTR),Y
        STA MOTOR.TIME+1  HIGH BYTE ONLY
        LDY #2       GET DRIVE #
        LDA (IOB.PNTR),Y
        LDY #16      SEE IF SAME AS OLD DRIVE#
        CMP (IOB.PNTR),Y
        BEQ .5       YES
        STA (IOB.PNTR),Y  UPDATE OLD DRIVE #
```

```
2030          PLP           SET Z STATUS
2040          LDY #0        TO FLAG MOTOR OFF
2050          PHP
2060   .5     ROR           CHECK LSB OF DRIVE #
2070          BCC .6        DRIVE 2
2080          LDA ENABLE.DRIVE.1,X
2090          BCS .7        ...ALWAYS
2100   .6     LDA ENABLE.DRIVE.2,X
2110   .7     ROR DRIVE.NO SET SIGN BIT IF DRIVE 1
2120          PLP           WAS MOTOR PROBABLY OFF?
2130          PHP
2140          BNE .9        NO, DEFINITELY ON
2150   *--------------------------------
2160   *      DELAY FROM 150 TO 180 MILLISECONDS,
2170   *      DEPENDING ON WHAT GARBAGE IS IN A-REG
2180   *--------------------------------
2190          LDY #7        YES, WAIT A WHILE
2200   .8     JSR $BA7F     ***BUG!!!*** SHOULD BE $BA7B
2210          DEY           BUT IT WORKS ANYWAY....
2220          BNE .8
2230          LDX SLOT      RESTORE SLOT#
2240   *--------------------------------
2250   .9     LDY #4        GET TRACK #
2260          LDA (IOB.PNTR),Y
2270          JSR SEEK.TRACK
2280          PLP           WAS MOTOR DEFINITELY ON?
2290          BNE PROCESS.COMMAND  YES, MOTOR ON
2300   *--------------------------------
2310   *      MOTOR WAS OFF, SO WAIT REST OF MOTOR ON TIME
2320   *      FOR APPLE DISK II, MOTOR ON TIME IS 1 SECOND.
2330   *      PART OF THIS TIME IS COUNTED DOWN WHILE SEEKING
2340   *      FOR THE TRACK.
2350   *--------------------------------
2360   .10    LDY #18       ABOUT 100 MICROSECONDS PER TRIP
2370   .11    DEY
2380          BNE .11
2390          INC MOTOR.TIME
2400          BNE .10
2410          INC MOTOR.TIME+1
2420          BNE .10
2430   *--------------------------------
2440   *      MOTOR ON AND UP TO SPEED, SO LET'S
2450   *      FIND OUT WHAT THE COMMAND IS AND DO IT!
2460   *--------------------------------
2470   PROCESS.COMMAND
2480          LDY #12       GET COMMAND
2490          LDA (IOB.PNTR),Y
2500          BEQ .8        NULL COMMAND, LET'S LEAVE
2510          CMP #4        FORMAT?
2520          BEQ .9        YES
2530          ROR           SET CARRY=1 IF READ, =0 IF WRITE
2540          PHP           SAVE ON STACK
2550          BCS .1        READ
2560          JSR PRE.NYBBLE  WRITE
```

```
2570  .1     LDY #48      UP TO 48 RETRIES
2580         STY RETRY.COUNT
2590  .2     LDX SLOT     GET SLOT NUMBER AGAIN
2600         JSR READ.ADDRESS
2610         BCC .5       GOOD ADDRESS READ
2620  .21    DEC RETRY.COUNT
2630         BPL .2       KEEP TRYING
2640  .3     LDA CURRENT.TRACK  GET TRACK WE WANTED
2650         PHA          SAVE IT
2660         LDA #96      PRETEND TO BE ON TRACK 96
2670         JSR SETUP.TRACK
2680         DEC SEEK.COUNT
2690         BEQ .6       NO MORE RE-CALIBRATES
2700         LDA #4
2710         STA SEARCH.COUNT
2720         LDA #0       LOOK FOR TRACK 0
2730         JSR SEEK.TRACK
2740         PLA          GET TRACK WE REALLY WANT
2750  .4     JSR SEEK.TRACK
2760         JMP .1
2770  *--------------------------------
2780  .5     LDY $2E      TRACK# IN ADDRESS HEADER
2790         CPY CURRENT.TRACK
2800         BEQ .10      FOUND RIGHT TRACK
2810         LDA CURRENT.TRACK
2820         PHA          SAVE TRACK WE REALLY WANT
2830         TYA          SET UP TRACK WE ACTUALLY FOUNG
2840         JSR SETUP.TRACK
2850         PLA          TRACK WE WANT
2860         DEC SEARCH.COUNT
2870         BNE .4       TRY AGAIN
2880         BEQ .3       TRY TO RE-CALIBRATE AGAIN
2890  *--------------------------------
2900  *      DRIVE ERROR, CANNOT FIND TRACK
2910  *--------------------------------
2920  .6     PLA          REMOVE CURRENT.TRACK
2930         LDA #ERR.BAD.DRIVE
2940  .7     PLP
2950         JMP ERROR.HANDLER
2960  *--------------------------------
2970  *      NULL COMMAND, ON THE WAY OUT....
2980  *--------------------------------
2990  .8     BEQ RWTS.EXIT
3000  *--------------------------------
3010  *      FORMAT COMMAND
3020  *--------------------------------
3030  .9     LDY #3       GET VOLUME# WANTED
3040         LDA (IOB.PNTR),Y
3050         STA VOLUME   SET IN PLACE AND GO FORMAT
3060         JMP FORMAT
3070  *--------------------------------
3080  *      READ OR WRITE COMMAND
3090  *--------------------------------
3100  .10    LDY #3       GET VOLUME# WANTED
```

```
        LDA (IOB.PNTR),Y
        PHA             SAVE DESIRED VOLUME# ON STACK
        LDA VOLUME
        LDY #14         STORE ACTUAL VOLUME NUMBER FOUND
        STA (IOB.PNTR),Y
        PLA             GET DESIRED VOLUME# AGAIN
        BEQ .11         IF =0, DON'T CARE
        CMP VOLUME      SEE IF RIGHT VOLUME
        BEQ .11         YES
        LDA #ERR.WRONG.VOLUME
        BNE .7          UH OH!
*--------------------------------
.11     LDY #5          GET SECTOR# WANTED
        LDA SECTOR      AND THE ONE WE FOUND
        CMP (IOB.PNTR),Y   AND COMPARE THEM.
        BNE .21         NOT THE RIGHT SECTOR
        PLP             GET COMMAND FLAG AGAIN
        BCC WRITE
        JSR READ.SECTOR
        PHP             SAVE RESULT; IF BAD, WILL BE COMMAND
        BCS .21         BAD READ
        PLP             THROW AWAY
        JSR POST.NYBBLE
        LDX SLOT
RWTS.EXIT
        CLC
        .HS 24          "BIT" TO SKIP NEXT INSTRUCTION
*--------------------------------
ERROR.HANDLER
        SEC             INDICATE AN ERROR
        LDY #13         STORE ERROR CODE
        STA (IOB.PNTR),Y
        LDA MOTOR.OFF,X
        RTS
*--------------------------------
WRITE   JSR WRITE.SECTOR
        BCC RWTS.EXIT
        LDA #ERR.WRITE.PROTECT
        BCS ERROR.HANDLER   ...ALWAYS
*--------------------------------
*       SEEK TRACK SUBROUTINE
*       (A) = TRACK# TO SEEK
*       (DRIVE.NO) IS NEGATIVE IF DRIVE 1
*                 AND POSITIVE IF DRIVE 2
*--------------------------------
SEEK.TRACK
        PHA             SAVE TRACK#
        LDY #1          CHECK DEVICE CHARACTERISTICS TABLE
        LDA (DCT.PNTR),Y   FOR TYPE OF DISK
        ROR             SET CARRY IF TWO PHASES PER TRACK
        PLA             GET TRACK# AGAIN
        BCC .1          ONE PHASE PER TRACK
        ASL             TWO PHASES PER TRACK, SO DOUBLE IT
        JSR .1          FIND THE TRACK
```

```
              LSR CURRENT.TRACK  DIVIDE IT BACK DOWN
              RTS
       *----------------------------------
       .1     STA TRACK
              JSR GET.SLOT.IN.Y
              LDA DRIVE.1.TRACK,Y
              BIT DRIVE.NO   WHICH DRIVE?
              BMI .2       DRIVE 1
              LDA DRIVE.2.TRACK,Y
       .2     STA CURRENT.TRACK   WHERE WE ARE RIGHT NOW
              LDA TRACK    WHERE WE WANT TO BE
              BIT DRIVE.NO WHICH DRIVE?
              BMI .3       DRIVE 1
              STA DRIVE.2.TRACK,Y   DRIVE 2
              BPL .4       ...ALWAYS
       .3     STA DRIVE.1.TRACK,Y
       .4     JMP SEEK.TRACK.ABSOLUTE
       *----------------------------------
       *      CONVERT SLOT*16 TO SLOT IN Y-REG
       *----------------------------------
       GET.SLOT.IN.Y
              TXA          SLOT*16 FROM X-REG
              LSR
              LSR
              LSR
              LSR
              TAY          SLOT INTO Y
              RTS
       *----------------------------------
       *      SET UP CURRENT TRACK LOCATION
       *      IN DRIVE.1.TRACK OR DRIVE.2.TRACK VECTORS,
       *      INDEXED BY SLOT NUMBER.
       *
       *      (A) = TRACK# TO BE SET UP
       *----------------------------------
       SETUP.TRACK
              PHA          SAVE TRACK # WE WANT TO SET UP
              LDY #2       GET DRIVE NUMBER FROM IOB
              LDA (IOB.PNTR),Y
              ROR          SET CARRY IF DRIVE 1, CLEAR IF 2
              ROR DRIVE.NO MAKE NEGATIVE IF 1, POSITIVE IF 2
              JSR GET.SLOT.IN.Y
              PLA          GET TRACK #
              ASL          DOUBLE IT
              BIT DRIVE.NO  WHICH DRIVE?
              BMI .1       DRIVE 1
              STA DRIVE.2.TRACK,Y
              BPL .2       ...ALWAYS
       .1     STA DRIVE.1.TRACK,Y
       .2     RTS
       *----------------------------------
       FORMAT
```

```
==================================================================
DOCUMENT :AAL-8103:DOS3.3:S.AmperIntf.txt
==================================================================

 1000  *---------------------------------
 1010  *      & COMMAND INTERFACE
 1020  *
 1030  *      &<COMMAND STRING>
 1040  *
 1050  *---------------------------------
 1060  *
 1070  *      ORIGIN MUST BE SET SO THAT LAST BYTE
 1080  *      IS AT $0FFF.
 1085         .OR $FD1
 1090  *---------------------------------
 1100         LDA #AMPERSAND.INTERFACE-$103D
 1110         STA $103C
 1120         RTS
 1130  *---------------------------------
 1140         JMP $1000
 1150  *---------------------------------
 1160  AOPTBL .HS 0503
 1170         .AS /EDI/
 1180         .DA EDIT-1
 1190         .AS /COP/
 1200         .DA COPY-1
 1210         .AS /SYM/
 1220         .DA STPRNT-1
 1230         .HS 00      END OF TABLE
 1240  *---------------------------------
 1250  EDIT   .EQ $1010
 1260  COPY   .EQ $1010
 1270  STPRNT .EQ $1E4E
 3000  *---------------------------------
 3010  AMPERSAND.INTERFACE
 3020         CMP #'&
 3030         BEQ .1
 3040         JMP $1063
 3050  .1     LDA #AOPTBL
 3060         STA $02
 3070         LDA /AOPTBL
 3080         STA $03
 3090         LDA #1
 3100         JMP $1047
```

```
=============================================================================
DOCUMENT :AAL-8103:DOS3.3:S.BernardMemD.txt
=============================================================================

 1000  *-----------------------------------
 1010  *
 1020  * APPLE II RELOCATABLE MEMORY DUMP PROGRAM
 1030  *      BY ROBERT H. BERNARD
 1040  *         35 DOGWOOD LANE
 1050  *         WESTPORT, CT 06880
 1060  *
 1070  *         JANUARY 17, 1981
 1080  *
 1090  *      COMMERCIAL RIGHTS RESERVED
 1100  *
 1110  *-----------------------------------
 1120  *      MONITOR ROM ROUTINES
 1130  *-----------------------------------
 1140  MON.COUT   .EQ $FDED
 1150  MON.RDKEY  .EQ $FD0C
 1160  MON.GTLNZ  .EQ $FD67
 1170  MON.ZMODE  .EQ $FFC7
 1180  MON.GETNUM .EQ $FFA7
 1190  MON.CROUT  .EQ $FD8E
 1200  MON.PRNTYX .EQ $F940
 1210  MON.PRBL2  .EQ $F94A
 1220  MON.PRBYTE .EQ $FDDA
 1230  MON.MON    .EQ $FF65
 1240  MON.HOME   .EQ $FC58
 1250  MON.SETMOD .EQ $FE18
 1260  MON.OUTPOR .EQ $FE95    SET OUTPUT PORT TO SLOT (A)
 1270  MON.SETVID .EQ $FE93    SET VIDEO
 1280  *-----------------------------------
 1290  *      I/O ADDRESSES
 1300  *-----------------------------------
 1310  KBD    .EQ $C000    KEYBOARD
 1320  KBSTRB .EQ $C010    KBD RESET STROBE
 1330  *-----------------------------------
 1340  *      PAGE-ZERO VARIABLES
 1350  *-----------------------------------
 1360  PGCNT  .EQ $2E      LINES LEFT THIS PAGE
 1370  ITEMCT .EQ $30      ITEMS PER LINE
 1380  OPTION .EQ $31      SAME AS MON "MODE"
 1390  PROMPT .EQ $33      LOC OF GETLN PROMPT CHAR
 1400  YSAV   .EQ $34      POINTER TO IN BUFFER
 1410  FRADRL .EQ $3C      STARTING ADR LO ORDER
 1420  FRADRH .EQ $3D      ..HI ORDER
 1430  TOADRL .EQ $3E      ENDING ADR LO ORDER
 1440  TOADRH .EQ $3F      ..HI ORDER
 1450  *-----------------------------------
 1460  *      USER-CHANGEABLE PARAMETERS
 1470  *-----------------------------------
 1480  SCITMS .EQ 16       BYTES PER LINE SCREEN
```

```
1490  PRITMS .EQ 32       BYTES PER LINE PRINTER
1500  ITMSPG .EQ 8        ITEMS PER PAGE
1510  PRSLOT .EQ 1        PRINTER SLOT
1520  *--------------------------------
1530         .OR $0800
1540  *--------------------------------
1550  MEMDMP JSR MON.SETVID   SET PR#0
1560         LDA #$BF     '?' FOR BOUNDS
1570         STA PROMPT   SET PROMPT CHAR
1580         JSR MON.GTLNZ    CR, THEN GET INPUT
1590         JSR MON.ZMODE    SET HEX DECODE MODE
1600         JSR MON.GETNUM
1610         STY YSAV     REMEMBER SCAN POS.
1620         CPX #0       ANY ADR SCANNED?
1630         BNE .3       YES
1640         RTS          NO. TERMINATE
1650         .DA MON.MON      MONITOR ENTRY (IN CASE YOU WANT
1660  *                   TO CHANGE RETURN TO "JMP MON.MON")
1670  *
1680  .3     LDA #-SCITMS  BYTES PER SCREEN LINE
1690         STA ITEMCT   ITEMS PER LINE
1700         JSR MON.SETMOD   SET TO SCAN 2ND ARG
1710         CMP #$AD     IS OPTION = '-' ?
1720         BNE .2       NO. CHECK OTHERS
1730         INC OPTION   MAKE '.'
1740         LDA #PRSLOT  PRINTER SLOT NO
1750         JSR MON.OUTPOR   SET OUTPUT PORT
1760         LDA #-PRITMS  BYTES PER PRINTER LINE
1770         STA ITEMCT   ITEMS PER LINE
1780         BNE .1       GO GET 2ND ARG
1790  *
1800  .2     CMP #$AE     '.' ?
1810         BEQ .1       YES. 2 ARGS
1820         CMP #$AC     ','?
1830         BNE SETPGL   ONLY ONE ARG
1840  .1     LDY YSAV     PTR TO IN BUFFER
1850         JSR MON.GETNUM   SCAN 2ND ARG
1860         STY YSAV     PTR TO IN BUFFER
1870  SETPGL LDA #ITMSPG  ITEMS PER PAGE
1880         STA PGCNT
1890  *
1900  NEXTLN JSR MON.CROUT    SKIP A LINE
1910         LDA ITEMCT   -ITEMS PER LINE
1920         AND FRADRL   STARTING ADR 0 MOD ITEMCT
1930         STA FRADRL
1940         TAX
1950         LDY FRADRH   ..TO PRINT
1960         JSR MON.PRNTYX   PRINT IT IN HEX
1970         LDX ITEMCT   NO OF BYTES THIS LINE
1980         LDY #0       POINTER
1990         BEQ NOBLNK   DON'T SPACE FIRST TIME
2000  *
2010  CHKKEY LDA KBD      KEY DOWN?
2020         BPL CKDONE   NO
```

```
2030        LDA KBSTRB   YES. CLEAR KEYBOARD
2040        SEC          PREPARE FOR
2050  MDMP2  BCS MEMDMP   JMP TO START
2060  *
2070  NXTCHR TYA          TEST FOR
2080        AND #$03       0 MOD 4
2090        BNE NOBLNK
2100        LDA #$A0
2110        JSR MON.COUT PRINT A BLANK
2120  NOBLNK LDA #$A0
2130        JSR MON.COUT PRINT A BLANK
2140        LDA (FRADRL),Y    GET CHAR TO PRINT
2150        CMP #$20     CNTRL CHAR?
2160        BCC .1       YES. SUBSTITUTE BLANK
2170        CMP #$80     CNTRL CHAR?
2180        BCC .2       NO. OK TO PRINT
2190        CMP #$A0     CNTRL CHAR?
2200        BCS .2       NO. OK TO PRINT
2210  .1     LDA #$A0     SUBSTITUTE BLANK
2220  .2     JSR MON.COUT
2230        INY          POINT AT NEXT
2240        INX          DONE ON THIS LINE?
2250        BNE NXTCHR   NO
2260        JSR MON.CROUT    YES. CR
2270  * PREPARE TO PRINT SAME ITEMS IN HEX
2280        LDX #3
2290        JSR MON.PRBL2    OUTPUT (X) BLANKS
2300        LDX ITEMCT   ITEMS PER LINE
2310        LDY #0       POINTER
2320        BEQ NXTHEX   (JMP)
2330  *
2340  SETPL1 BCS SETPGL   JUMP TO SET PG LENGTH
2350  CKOPT  CMP #$AC     NO. OPTION=',' ?
2360  NXTLN1 BNE NEXTLN   NO. JUMP TO PRINT
2370  CKDONE LDA FRADRL   TEST IF DONE
2380        CMP TOADRL
2390        LDA FRADRH
2400        SBC TOADRH
2410        BCC NEXTLN   FROM < TO
2420  MDMP1  BCS MDMP2    JMP TO START
2430  *
2440  NXTHEX TYA          TEST FOR
2450        AND #$03       0 MOD 4
2460        BNE .1       IF NOT, SKIP BLANK
2470        LDA #$A0
2480        JSR MON.COUT PRINT A BLANK
2490  .1     LDA (FRADRL),Y BYTE TO OUTPUT
2500        JSR MON.PRBYTE   OUTPUT IN HEX
2510        INY          NEXT
2520        INX          DONE ON THIS LINE?
2530        BNE NXTHEX   NO
2540        JSR MON.CROUT    YES. CR
2550  *      ADVANCE DUMP ADDRESS
2560        SEC          PREPARE FOR SUBTRACT
```

```
2570        LDA FRADRL   INCREMENT ADDRESS
2580        SBC ITEMCT   -ITEMS PER LINE
2590        STA FRADRL
2600        BCC .2       NO CARRY
2610        INC FRADRH   PAGE BOUNDARY
2620        BEQ MDMP1    END OF MEMORY
2630 .2     LDA OPTION
2640        CMP #$AE     '.'? (OPTION 1)
2650        BEQ CHKKEY   NO. CHECK IF KEY DOWN
2660 CHKPAG DEC PGCNT    PAGE END?
2670        BNE CKOPT    NO. CHECK OPTION
2680 PAUSE  JSR MON.RDKEY    GET A CHAR
2690        CMP #$83     CNTRL-C?
2700        BEQ MDMP1    YES. START OVER
2710        CMP #$C2     WAS CHAR READ A 'B'?
2720        BEQ BACKUP   YES
2730        LDA OPTION
2740        CMP #$AC     OPTION=',' ?
2750        BEQ SETPL1   YES
2760 ADVNCE INC PGCNT    ONE MORE TIME
2770        BNE NXTLN1   JMP TO NXTLN
2780 *
2790 BACKUP LDA FRADRL   CARRY IS SET
2800        SBC #144     BACKUP SCITMS*(ITMSPG+1) BYTES
2810        STA FRADRL   SAVE LO ORDER
2820        BCS .1       NO CARRY
2830        DEC FRADRH   PROPOGATE CARRY
2840 .1     JSR MON.HOME     CLEAR SCREEN
2850        SEC          SIMULATE JMP
2860        BCS SETPL1   ..TO SETPGL
2870 *
2880 ZZSIZE .EQ *-MEMDMP PROGRAM SIZE
9999        .LIF
```

================================================================================
DOCUMENT :AAL-8103:DOS3.3:Welman.Modifier.txt
================================================================================

( DTC removed -- lots of garbage characters )

==========================================================================
DOCUMENT :AAL-8104:Articles:AS.Substr.srch.txt
==========================================================================

Substring Search Function for Applesoft

Lee Reynolds' article in the January 1981 Call A.P.P.L.E. touched off this project. When you are searching through text arrays for keywords, or through a mailing list for someone who lives on "XYZ Street", Applesoft can be vveeerrrrryyy slow. This subroutine, linked in through the famous ampersand feature, will give you the speed your Apple is famous for.

Lee's program was quite similar to this one, but it did not allow the keyword or the string-to-be-searched to be expressions. He left that extension as "an exercise for the reader". Being one reader badly in need of exercise, I took up the challenge.

Although it is not really necessary, I used one of the newly discovered "secret" opcodes (which I wrote about last month) at line 2060. If you like, you can replace that line with:

```
     2060 GS1    LDA (FACMO),Y
     2065        TAX
```

Here is a sample Applesoft program which uses the Substring Search Subroutine. Line 10 loads the subroutine and calls 768 to link in the ampersand vector. Line 120 reads in your search key. If you just hit the RETURN key, the program quits.

Line 130 gets the next string to be searched from the DATA list. If the value is ".", we are at the end of the list, so it loops back to line 110.

Line 140 calls our substring search subroutine to see if the key string can be found in the search string. If not, it jumps back to line 130 to get another search string. Lines 150-180 print the search string, emphasizing the portion that matched the key string by printing it in inverse.

==================================================================
DOCUMENT :AAL-8104:Articles:DOS.Format.List.txt
==================================================================

Commented Listing of DOS 3.2.1 Format

Here is the second installment of DOS disassembly, covering the area from $BEA0 through $BFFF. If you read the listing in last month's AAL carefully, you probably noted that it ended with the label definition "FORMAT", but no code followed. Well, here it is!

FORMAT turns a blank diskette into one with address headers recorded on every track. Otherwise, the disk is empty. No directory is written into track $11 yet, nor is any DOS recorded yet in tracks 0, 1, and 2. When you use the INIT command, the first step exectured is to format the disk; after formatting, a DOS image and empty directory are written; then your HELLO program is SAVEd.

By the way, there are a lot of differences between DOS 3.2.1 and DOS 3.3 FORMAT routines. Later in this issue of AAL you will find a commented listing of the DOS 3.3 version. If you compare the two, you will find at least these major differences:

1. DOS 3.2.1 formats 13 sectors per track, DOS 3.3 formats 16 sectors per track.
2. DOS 3.2.1 writes an address header followed by a long series of $FF bytes where the data should be; DOS 3.3 writes an address header followed by a standard data block (the data is all $00 bytes).
3. DOS 3.2.1 writes an address header starting with $D5AAB5; DOS 3.3 writes an address header starting with $D5AA96.
4. DOS 3.2.1 verifies correct format by trying to read sector 0 immediately after formatting the last sector; no other verification is made. DOS 3.3 tries to read EVERY sector just formatted; it does a complete check of the track.
5. DOS 3.2.1 writes the sectors in the order 0, 10, 7, 4, 1, 11, 8, 5, 2, 12, 9, 6, 3; DOS 3.3 writes them in sequential order 0, 1, 2, ... , 15.

The Apple Disk Interface depends on critical software timing to operate correctly. You will find many strange sequences of code (such as PHA, PLA, NOP, PHA, PLA between $BF47 and $BF4B) which are for timing purposes. If you are interested in counting cycles, the timing for each opcode-address mode combination are listed in the Quick Reference Card that came with your S-C ASSEMBLER II Version 4.0.

Commented Listing of DOS 3.3 Format

As promised three or four pages ago, here is my rendition of the DOS 3.3 Format routine.

==============================================================================
DOCUMENT :AAL-8104:Articles:Front.Page.txt
==============================================================================

Volume 1 -- Issue 7 April, 1981

As of today the total distribution of the Apple Assembly Line is nearly 350. Let's shoot for 1000 by the end of 1981! I will have a full page ad in the next eight issues of NIBBLE, so I think 1000 is a reasonable goal. Thank you for your support!

In This Issue...



Cross Reference (XREF) for S-C ASSEMBLER II

Bob Kovacs has a new product, one which many of you have asked me for. It enables you to produce a complete cross reference listing of all symbols used in an assembly language program. See his ad on page 7 for a description and ordering information.

I am honored to have three companies (Rak-Ware, Decision Systems, and Flatland Software) producing software to complement my assembler!

80 Columns on Your Printer

For some reason unknown to me Apple's Parallel Interface Card comes with at least three different ROM's. There seems to me no indication on the package which one you are getting, and no listing in the manual of the exact ROM on the board. This leads to confusion, because some ROM versions will print 80-column assembly listings at the drop of a hat (Just type PR#1 and ASM, and you have it!); but others require you special treatment.

If you have the latter type, I have found that this works:

```
     :PR#1         (assuming slot # 1)
     :$579:50      ($578 + slot#     )
     :ASM
```

```
================================================================
DOCUMENT :AAL-8104:Articles:Hiding.Undr.DOS.txt
================================================================
```

Hiding Things Under DOS..........................Rick Hatcher

In issue number 5/1980 of NIBBLE, a small article by William Reynolds III tells how to do something I have wondered about for a long time. That is how to move the HIMEM pointer down so that machine language code or something else can be put out of the way and protected. For example: I have a lower-case routine I like to use on key input; I also like to use the character display routine from Lawrence Hall of Science which is hooked into the control-Y pointer. This is one way to dump memory in both hex and ASCII. I have looked for protected areas but until now the only place seemed to be from $300 to $3CF. This is a little over 200 bytes, and I needed about 400.

Neil Konzen's Program Line Editor (from Call A.P.P.L.E.) moves the file buffers down and leaves space between the buffers and DOS...but the manual which I sneaked a look at does not tell how to do it. The article in NIBBLE on page 40 finally revealed the secret. The file buffers are located by a pointer at locations $9D00 and $9D01 (least significant byte first, as usual). A DOS routine at $A7D4 builds the buffers using this pointer and the value of MAXFILES (at $AA57). [note: all addresses assume a 48K system]

All you have to do is change the address at $9D00.9D01 and call the routine at $A7D4. I wanted to create a space of $200 bytes (512 decimal). The normal value at $9D00.9D01 is $9CD3. I changed it to $9AD3, and then typed A7D4G in the monitor. The value of HIMEM was automatically changed to $9400 from the usual $9600. The protected area is from $9B00 to $9CFF. The buffers are located from $9400 to $9AFF and DOS is located from $9D00 to BFFF. If a MAXFILES command is used it changes HIMEM but the buffer top at $9AFF stays unchanged.

To make space like this from an Applesoft program, here is all you need:

```
     100 POKE 40193,154
     110 POKE 40192,211
     120 CALL 42964
```

It isn't so easy in Integer BASIC, because the routine moves HIMEM without moving the program down in memory. (Remember Integer BASIC programs are at the top of memory up against HIMEM; Applesoft programs are at the low end of memory.) The NIBBLE article gives a method for Integer BASIC, but I haven't tried it.

I use an Applesoft HELLO program which first does the three lines above, and then BRUNs or BLOADs the code I want to hide. The BRUN portion sets up the I/O hooks at $36.39 and sets up the control-Y vector at $3F8. I use the BLOAD if I want the code resident but not hooked in.

Once the space is made, it stays there.  If you INIT a slave disk, the slave has the same change.

The NIBBLE article reveals a few more details about the buffers in which you may be interested.

==============================================================
DOCUMENT :AAL-8104:Articles:Part.1.txt
==============================================================

Patch S-C Assembler II for More Errors

Some of you have asked for a way to see all your errors at once. If you patch Version 4.0 in this simple way, you will see all error messages during one ASM, instead of aborting the assembly after the first error.

Look at $1752 to $1754; you should see 20 81 1A. If you do, then make this patch:

```
     :$1752:4C 8E 18
```

Now try an assembly of some source code with several errors in it. You will see all the errors on your screen. Or if your printer is on, they will all print.

Personally, I liked it better the other way. But if you never make more than one error per program, you won't be able to tell the difference!

Fast String Input Routine for Applesoft

Yet another use for the imperious ampersand! This program will read a line from the keyboard or a text file into a string variable. It will accept commas and colons without complaint, too. No more "EXTRA IGNORED" messages, and much less chance of garbage collection tying things up.

The program is shown here with the origin set to $0300, the most popular place in your Apple. If that taxi is already full, you can change the origin to whatever you like. In fact, the subroutine itself is completely relocatable. You can put it anywhere in memory you like, just so you set $3F6 and 3F7 to point to it.

Lines 1160-1220 are executed if you BRUN a file with this program on it. They put a JMP GET into $3F5, so that the "&" will call my subroutine. Once this code is executed, you can execute statements like "&GET A$" to read a line into a string.

Lines 1240-1500 are the input subroutine. At line 1240 the token following the ampersand is tested; it should be $BE, which is the token for "GET". If not, JMP $DEC9 makes your screen say "SYNTAX ERROR"!

Lines 1270 and 1280 set up the address of the string variable in locations $83 and $84. We will use this later to tell Applesoft where the input line is.

Lines 1290-1360 change the prompt symbol to a bell (in case you backspace too much) and call on the monitor input routine to read a line. After the line is read, the prompt is restored to whatever it was before. The length of the input line is in the X-register, and the line itself is in the buffer starting at $0200.

Lines 1370 and 1380 call on Applesoft to set aside space for the input line in the string area. This may force garbage collection if you are about out of memory at the time. GETSPA leaves the address of the start of the slot set aside for our input line in locations $71 and $72.

Lines 1390-1460 store the length and address of the input line into the string variable. The address is of the slot GETSPA just reserved.

Lines 1470-1500 call on MOVSTR to copy the input line from the monitor's input buffer (at $0200) into the slot reserved by GETSPA.

Now if you want to read some data off the disk which might have commas and colons in it, you can do it like this:

```
100 PRINT CHR$(4) "OPEN MY.FILE"
110 PRINT CHR$(4) "READ MY.FILE"
120 FOR I = 1 TO 10
130 & GET A$(I)
140 NEXT I
```

Applesoft Internal Entry Points

An excellent article appeared just over a year ago (by the same title) in The Apple Orchard, Volume 1, Number 1, March/April 1980. John Crossley of Apple Computer, Inc. wrote it. He revealed most of the usable entry points within the Applesoft ROM, and many details on how they work and how to use them. If you don't have that magazine, go get one right away. They are available at some stores, through some local Apple clubs, and directly from the publisher (the Internatioal Apple Corps). There are a few typographical errors, but you should be able to figure them out by comparing with a disassembly.

To get you started, I have made up a list of my own which includes the starting addresses for all the keyword routines.
I got these from the ROM itself. The keyword list starts at $D0D0, and a parallel list of addresses starts at $D000. The addresses in the list are all low-byte-first, and are all pointing to one byte before the actual start. That is because Applesoft branches to the appropriate routine by placing the address from this list on the stack and then using RTS (see AAL issue #1, page 11, for an explanation of this technique).

This chart shows all the token values for Applesoft, and the address where the token is processed.

```
 token   keyword  addr
```

```
80  128  END      D870
81  129  FOR      D766
82  130  NEXT     DCF9
83  131  DATA     D995
84  132  INPUT    DBB2
85  133  DEL      F331
86  134  DIM      DFD9
87  135  READ     DBE2
88  136  GR       F390
89  137  TEXT     F399
8A  138  PR#      F1E5
8B  139  IN#      F1DE
8C  140  CALL     F1D5
8D  141  PLOT     F225
8E  142  HLIN     F232
8F  143  VLIN     F241
90  144  HGR2     F3D8
91  145  HGR      F3E2
92  146  HCOLOR=  F6E9
93  147  HPLOT    F6FD
94  148  DRAW     F769
95  149  XDRAW    F76F
96  150  HTAB     F7E7
97  151  HOME     FC58
98  152  ROT=     F721
99  153  SCALE=   F727
9A  154  SHLOAD   F775
9B  155  TRACE    F26D
9C  156  NOTRACE  F26F
9D  157  NORMAL   F273
9E  158  INVERSE  F277
9F  159  FLASH    F280
A0  160  COLOR=   F24F
A1  161  POP      D96B
A2  162  VTAB     F256
A3  163  HIMEM:   F286
A4  164  LOMEM:   F2A6
A5  165  ONERR    F2CB
A6  166  RESUME   F318
A7  167  RECALL   F3BC
A8  168  STORE    F39F
A9  169  SPEED=   F262
AA  170  LET      DA46
AB  171  GOTO     D93E
AC  172  RUN      D912
AD  173  IF       D9C9
AE  174  RESTORE  D849
AF  175  &        03F5
B0  176  GOSUB    D921
B1  177  RETURN   D96B
B2  178  REM      D9DC
B3  179  STOP     D86E
B4  180  ON       D9EC
B5  181  WAIT     E784
```

```
 token    keyword  addr

B6  182  LOAD     D8C9
B7  183  SAVE     D8B0
B8  184  DEF      E313
B9  185  POKE     E77B
BA  186  PRINT    DAD5
BB  187  CONT     D896
BC  188  LIST     D6A5
BD  189  CLEAR    D66A
BE  190  GET      DBA0
BF  191  NEW      D649
C0  192  TAB(
C1  193  TO
C2  194  FN
C3  195  SPC(
C4  196  THEN
C5  197  AT
C6  198  NOT
C7  199  STEP
C8  200  +
C9  201  -
CA  202  *
CB  203  /
CC  204  ^
CD  205  AND
CE  206  OR
CF  207  >
D0  208  =
D1  209  <
D2  210  SGN      EB91
D3  211  INT      EC24
D4  212  ABS      EBB0
D5  213  USR      000A
D6  214  FRE      E2DF
D7  215  SCRN(    D413
D8  216  PDL      DFCE
D9  217  POS      E300
DA  218  SQR      EE8E
DB  219  RND      EFAF
DC  220  LOG      E942
DD  221  EXP      EF0A
DE  222  COS      EFEB
DF  223  SIN      EFF2
E0  224  TAN      F03B
E1  225  ATN      F09F
E2  226  PEEK     E765
E3  227  LEN      E6D7
E4  228  STR$     E3C6
E5  229  VAL      E708
E6  230  ASC      E6E6
E7  231  CHR$     E647
E8  232  LEFT$    E65B
```

```
E9  233  RIGHT$   E687
EA  234  MID$     E691
```

=============================================================================
DOCUMENT :AAL-8104:Articles:Text.File.IO.txt
=============================================================================

Text File I/O in Assembly Language Programs

A surprisingly large number of people have written or called to ask the same question:

"How can I read or write a text file from my program? I know I can issue OPEN, READ, WRITE, and CLOSE commands just like in Applesoft -- by outputting a control-D and the command string. But after that, where is the data?"

It is really very simple, and after I tell you, you may be just as embarrassed as they were!

Remember that in Applesoft, after opening a file and setting it up to read with the OPEN and READ commands, you actually read it with normal INPUT statements. In assembly language you do the same thing. You can either input a line by calling the monitor routine at $FD6F, or you can read character-by-character by calling the character input routine at $FD0C. After a JSR $FD0C, the input character will be in the A-register. After a JSR $FD6F, the input line will be in the monitors buffer starting at $0200, and the X-register will contain the number of characters in the line (not counting the carriage return).

Also remember that after using the OPEN and WRITE commands, all you do in Applesoft to write on a text file is use the normal PRINT statement. In the same way, from assembly language, you just call the monitor print character routine at $FDED. The character to be written should be in the A-register, and then use JSR $FDED.

Here is a little program which opens a text file and reads it into a buffer at $4000. It demonstrates a few more tricks you might need to know, as well.

Lines 1180-1270 patch DOS so that it thinks you are executing an Applesoft program. (If you really are calling this from a RUNning Applesoft program, you can skip lines 1190 and 1200.) We want to be able to issue DOS commands by printing control-D and the command string, so we have to be RUNning. We want to be able to tell when the end-of-file comes without getting an "OUT OF DATA" error, so we turn on the Applesoft ON ERR flag and set it up to branch to our own END.OF.DATA routine.

Lines 1310-1350 print the DOS OPEN and READ commands. The message printer is a very simple loop at lines 1630-1690.

Lines 1380-1500 read the characters from the file and store them in a buffer at $4000. I save the stack pointer before the loop so I can restore it after the end-of-file occurs. Lines 1530-1570 restore the stack pointer, close the file, and return to DOS.

I really should clean up the mess I created with lines 1180-1270, but I will leave that as an exercise for the reader.

```
==================================================================
DOCUMENT :AAL-8104:DOS3.3:Demo.Txt.Fl.Rd.txt
==================================================================

 1000  *---------------------------------
 1010  *      DEMONSTRATION OF READING A TEXT FILE
 1020  *---------------------------------
 1030  PROMPT.CHAR        .EQ $33
 1040  CURRENT.LINE.NO    .EQ $75,76
 1050  BUF.PNTR           .EQ $9D,9E
 1060  DOS.LANGUAGE.FLAG  .EQ $AAB6
 1070  ONERR.FLAG         .EQ $D8
 1080  DOS.ONERR.PNTR     .EQ $9D5A,9D5B
 1090  DOS.REENTRY        .EQ $3D0
 1100  MON.RDKEY          .EQ $FD0C
 1110  MON.COUT           .EQ $FDED
 1120  *---------------------------------
 1130  TEXT.READER
 1140  *---------------------------------
 1150  *      PATCH DOS SO END OF FILE WILL
 1160  *      BRANCH TO MY "END.OF.DATA"
 1170  *---------------------------------
 1180         LDA #1       TELL DOS WE ARE IN APPLESOFT
 1190         STA DOS.LANGUAGE.FLAG
 1200         STA CURRENT.LINE.NO+1  NOT IN DIRECT MODE
 1210         STA PROMPT.CHAR        NOT DIRECT MODE
 1220         LDA #$FF     TURN ON "ON ERR"
 1230         STA ONERR.FLAG
 1240         LDA #END.OF.DATA
 1250         STA DOS.ONERR.PNTR
 1260         LDA /END.OF.DATA
 1270         STA DOS.ONERR.PNTR+1
 1280  *---------------------------------
 1290  *      OPEN THE FILE
 1300  *---------------------------------
 1310         LDY #QOPEN-QTS
 1320         JSR QUOTE.PRINT
 1330         LDY #QREAD-QTS
 1340         JSR QUOTE.PRINT
 1350  *---------------------------------
 1360  *      READ THE FILE
 1370  *---------------------------------
 1380         TSX
 1390         STX OLD.STACK.PNTR
 1400         LDA #BUFFER
 1410         STA BUF.PNTR
 1420         LDA /BUFFER
 1430         STA BUF.PNTR+1
 1440  .1     JSR MON.RDKEY   READ CHARACTER
 1450         LDY #0
 1460         STA (BUF.PNTR),Y
 1470         INC BUF.PNTR
 1480         BNE .1
```

```
       INC BUF.PNTR+1
       BNE .1       ...ALWAYS
*---------------------------------
END.OF.DATA
       LDX OLD.STACK.PNTR
       TXS
       LDY #QCLOSE-QTS
       JSR QUOTE.PRINT
       JMP DOS.REENTRY
*---------------------------------
*      PRINT A MESSAGE
*      MESSAGE STARTS AT QTS,Y
*      MESSAGE ENDS WITH 00 BYTE
*---------------------------------
QUOTE.PRINT
.1     LDA QTS,Y
       BEQ .2
       JSR MON.COUT
       INY
       BNE .1       ...ALWAYS
.2     RTS
*---------------------------------
QTS    .EQ *
QOPEN  .HS 84       CONTROL-D
       .AS -/OPEN TESTFILE/
       .HS 8D00
QREAD  .HS 84       CONTROL-D
       .AS -/READ TESTFILE/
       .HS 8D00
QCLOSE .HS 84       CONTROL-D
       .AS -/CLOSE/
       .HS 8D00
*---------------------------------
OLD.STACK.PNTR .BS 1
*---------------------------------
BUFFER     .EQ $4000
*---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8104:DOS3.3:DOS321BEAO.BFFF.txt
==============================================================================

 1000  *       .LIST OFF
 1010  *---------------------------------
 1020  *       DOS 3.2.1 DISASSEMBLY $BEA0-BFFF
 1030  *       BOB SANDER-CEDERLOF      3-26-81
 1040  *---------------------------------
 1050  CURRENT.TRACK       .EQ $478
 1060  *---------------------------------
 1070  PHASE.OFF           .EQ $C080
 1080  PHASE.ON            .EQ $C081
 1090  MOTOR.OFF           .EQ $C088
 1100  MOTOR.ON            .EQ $C089
 1110  ENABLE.DRIVE.1      .EQ $C08A
 1120  ENABLE.DRIVE.2      .EQ $C08B
 1130  Q6L                 .EQ $C08C
 1140  Q6H                 .EQ $C08D
 1150  Q7L                 .EQ $C08E
 1160  Q7H                 .EQ $C08F
 1170  *---------------------------------
 1180  SECTOR     .EQ $2D
 1190  VOLUME     .EQ $2F
 1200  TRACK.CNTR .EQ $41
 1210  DATA.CNTR  .EQ $46
 1220  SYNC.CNT   .EQ $47
 1230  CONST.AA   .EQ $4A
 1240  FILL.CNTR  .EQ $4B
 1250  FMT.SECTOR .EQ $4B
 1260  *---------------------------------
 1270  READ.ADDRESS        .EQ $B965
 1280  SEEK.TRACK.ABSOLUTE .EQ $BA1E
 1290  RWTS.EXIT           .EQ $BE37
 1300  ERROR.HANDLER       .EQ $BE39
 1310  *---------------------------------
 1320  ERR.BAD.DRIVE       .EQ $40
 1330  *---------------------------------
 1340         .OR $BEA0
 1350         .TA $800
 1360  *---------------------------------
 1370  FORMAT LDA #128     SET CURRENT TRACK REAL HIGH
 1380         STA CURRENT.TRACK   SO DRIVE WILL HOME
 1390         LDA #0              TO TRACK 0
 1400         STA TRACK.CNTR  INIT COUNTER FOR INIT ROUTINE
 1410         JSR SEEK.TRACK.ABSOLUTE
 1420  *---------------------------------
 1430         LDA #$AA     SAVE $AA IN PAGE ZERO FOR TIMING
 1440         STA CONST.AA
 1450  *---------------------------------
 1460  *      FILL ENTIRE TRACK WITH SYNC BYTES
 1470  *---------------------------------
 1480         LDY #80      START WITH 80 SYNC-BYTES
```

```
1490  FILL.TRACK.WITH.SYNC
1500         STY SYNC.CNT  # OF SYNC BYTES BETWEEN SECTORS
1510         LDA #39      WRITE SYNC'S OVER ENTIRE TRACK
1520         STA FILL.CNTR
1530         LDA Q6H,X    GET READY TO WRITE
1540         LDA Q7L,X
1550         LDA #$FF     WRITE $FF EVERYWHERE
1560         STA Q7H,X    ALL SET TO WRITE....
1570         CMP Q6L,X
1580         BIT $00      DELAY 3 CYCLES
1590  .1     DEY
1600         BEQ .3
1610         PHA
1620         PLA          THESE ARE JUST FOR TIMING
1630         NOP          NEED 27 CYCLES BTWN WRITES
1640  .2     PHA
1650         PLA
1660         NOP
1670         NOP
1680         STA Q6H,X    WRITE SYNC BYTE
1690         CMP Q6L,X
1700         BCS .1       ...ALWAYS
1710  .3     DEC FILL.CNTR  TRACK FULL YET?
1720         BNE .2       NO
1730  *---------------------------------
1740  *      WRITE 13-SECTOR HEADERS ON TRACK
1750  *
1760  *          EACH SECTOR CONSISTS OF AN ADDRESS BLOCK
1770  *          AND A DATA BLOCK.
1780  *          ADDRESS:  D5 AA B5 V1 V2 T1 T2
1790  *                    S1 S2 C1 C2 DE AA EB
1800  *          DATA:     FORMATTED TO ALL SYNC BYTES
1810  *---------------------------------
1820  FORMAT.TRACK
1830         LDY SYNC.CNT  # SYNC BYTES BTWN SECTORS
1840         NOP
1850         NOP
1860  .1     BNE .4       ...ALWAYS
1870  *---------------------------------
1880  .2     PHA          WRITE SYNC BYTES BEFORE SECTOR
1890         PLA
1900         PHA
1910         PLA
1920         CMP ($00,X)  DELAY 6 CYCLES
1930  .4     NOP
1940  .5     STA Q6H,X    WRITE NEXT SYNC BYTE
1950         CMP Q6L,X
1960         DEY
1970         BNE .2
1980  *---------------------------------
1990         LDA #$D5     WRITE D5 AA B5
2000         JSR WRITE.BYTE.2
2010         LDA #$AA
2020         JSR WRITE.BYTE.3
```

```
2030        LDA #$B5
2040        JSR WRITE.BYTE.3
2050        LDA VOLUME   WRITE VOLUME, TRACK, AND SECTOR
2060        JSR WRITE.BYTE.1
2070        LDA TRACK.CNTR
2080        JSR WRITE.BYTE.1
2090        LDA FMT.SECTOR
2100        JSR WRITE.BYTE.1
2110        LDA VOLUME   COMPUTE CHECKSUM
2120        EOR TRACK.CNTR
2130        EOR FMT.SECTOR
2140        PHA          WRITE CHECKSUM
2150        LSR
2160        ORA CONST.AA   #$AA, FOR TIMING
2170        STA Q6H,X
2180        CMP Q6L,X
2190        PLA
2200        ORA #$AA
2210        JSR WRITE.BYTE.2
2220        LDA #$DE     WRITE DE AA EB
2230        JSR WRITE.BYTE.3
2240        LDA #$AA
2250        JSR WRITE.BYTE.3
2260        LDA #$EB
2270        JSR WRITE.BYTE.3
2280        LDA #$FF     WRITE MORE SYNC BYTES
2290        JSR WRITE.BYTE.3
2300        LDY #2       FILL WHOLE DATA BLOCK WITH $FF
2310        STY DATA.CNTR
2320        LDY #173
2330        BNE .7       ...ALWAYS
2340  .6    DEY          FINISHED?
2350        BEQ .8       YES, AT LEAST THIS GROUP
2360        PHA          23 CYCLES PER BYTE
2370        PLA
2380        NOP
2390  .7    PHA
2400        PLA
2410        STA Q6H,X
2420        CMP Q6L,X
2430        BCS .6       ...ALWAYS
2440  .8    DEC DATA.CNTR  FINISHED?
2450        BNE .7       NOT YET, DO SECOND GROUP
2460  *---------------------------------
2470        LDY SYNC.CNT
2480        CLC
2490        BIT $00      DELAY
2500        STA Q6H,X
2510        LDA Q6L,X
2520        LDA FMT.SECTOR   COMPUTE NEXT SECTOR #
2530        ADC #10      SKEW FACTOR = 10
2540        STA FMT.SECTOR
2550        SBC #12
2560        BEQ CHECK.TRACK
```

```
       BCS .9       STORE VALUE MODULO 13
       .HS 2C       'BIT' OPCODE TO SKIP NEXT TWO BYTES
.9     STA FMT.SECTOR
       LDA #$FF
       JMP .5       DO NEXT SECTOR
*-------------------------------
*      CHECK WHETHER TRACK OVERLAPPED
*-------------------------------
CHECK.TRACK
       PHA          TIME DELAY
       PLA
       LDY SYNC.CNT
       LDA Q6H,X    SET UP TO READ
       LDA Q7L,X    SENSE WRITE PROTECT
       BMI .4       DRIVE ERROR
       DEY
.1     PHA          DELAY LOOP
       PLA
       PHA
       PLA
       PHA
       PLA
       DEY          FINISHED WITH DELAY YET?
       BNE .1       NO
       JSR READ.ADDRESS
       BCS .2       BAD READ
       LDA SECTOR   SHOULD BE SECTOR 0
       BEQ .3       YES!
.2     LDY SYNC.CNT  DIMINISH SYNC COUNT
       DEY          AND TRY AGAIN
       CPY #16      UNLESS NOT ENOUGH LEFT
       BCC .4       DRIVE ERROR
       JMP FILL.TRACK.WITH.SYNC
*-------------------------------
.3     INC TRACK.CNTR  NEXT TRACK
       LDA TRACK.CNTR
       CMP #35      FINISHED?
       BCS .5       YES
       ASL          DOUBLE FOR TRACK SEEK ROUTINE
       JSR SEEK.TRACK.ABSOLUTE
       LDY SYNC.CNT  BUMP SYNC.CNT BEFORE TRYING
       INY           NEXT TRACK
       INY
       STY SYNC.CNT
       JMP FILL.TRACK.WITH.SYNC
*-------------------------------
.4     LDA #ERR.BAD.DRIVE
       JMP ERROR.HANDLER
*-------------------------------
.5     JMP RWTS.EXIT
*-------------------------------
*      SUBROUTINES TO WRITE BYTE ON DISK
*-------------------------------
WRITE.BYTE.1
```

```
3110        PHA          ADDRESS BLOCK FORMAT
3120        LSR
3130        ORA CONST.AA
3140        STA Q6H,X
3150        CMP Q6L,X
3160        PLA
3170        CMP ($00,X)  DELAY 6 CYCLES
3180        ORA #$AA
3190 WRITE.BYTE.2
3200        NOP
3210 WRITE.BYTE.3
3220        PHA
3230        PLA
3240        NOP
3250        STA Q6H,X
3260        CMP Q6L,X
3270        RTS
3280 *----------------------------------
3290 *      VARIOUS ODDS AND ENDS
3300 *----------------------------------
3310        .HS 0160     LEFT OVER
3320 PATCH1 JMP $A5DD
3330 PATCH2 STA $AA63
3340        STA $AA70
3350        STA $AA71
3360        RTS
3370 PATCH3 JSR $A75B
3380        STY $AAB7
3390        RTS
3400 PATCH4 JSR $AE7E    FROM $B377
3410        LDX $B39B
3420        TXS
3430        JSR $A316
3440        TSX
3450        STX $B39B
3460        LDA #9       "DISK FULL" ERROR
3470        JMP $B385
```

==============================================================================
DOCUMENT :AAL-8104:DOS3.3:DOS33.BEAF.BFFF.txt
==============================================================================

```
 1000  *       .LIST OFF
 1010  *----------------------------------
 1020  *       DOS 3.3 DISASSEMBLY     $BEAF-BFFF
 1030  *       BOB SANDER-CEDERLOF        3-26-81
 1040  *----------------------------------
 1050  RETRY.COUNT         .EQ $578
 1060  *----------------------------------
 1070  PHASE.OFF           .EQ $C080
 1080  PHASE.ON            .EQ $C081
 1090  MOTOR.OFF           .EQ $C088
 1100  MOTOR.ON            .EQ $C089
 1110  ENABLE.DRIVE.1      .EQ $C08A
 1120  ENABLE.DRIVE.2      .EQ $C08B
 1130  Q6L                 .EQ $C08C
 1140  Q6H                 .EQ $C08D
 1150  Q7L                 .EQ $C08E
 1160  Q7H                 .EQ $C08F
 1170  *----------------------------------
 1180  SECTOR     .EQ $2D
 1190  CONST.AA   .EQ $3E
 1200  FMT.SECTOR .EQ $3F
 1210  VOLUME     .EQ $41
 1220  TRACK.CNTR .EQ $44
 1230  SYNC.CNT   .EQ $45
 1240  IOB.PNTR   .EQ $48,49
 1250  *----------------------------------
 1260  WRITE.SECTOR        .EQ $B82A
 1270  READ.SECTOR         .EQ $B8DC
 1280  READ.ADDRESS        .EQ $B944
 1290  RWTS.BUFFER         .EQ $BB00
 1300  WRITE.ADDRESS       .EQ $BC56
 1310  SEEK.TRACK          .EQ $BE5A
 1320  SETUP.TRACK         .EQ $BE95
 1330  *----------------------------------
 1340  ERR.CANT.FORMAT     .EQ $08
 1350  *----------------------------------
 1360         .OR $BEAF
 1370         .TA $800
 1380  *----------------------------------
 1390  FORMAT LDY #3       POINT AT VOLUME NUMBER
 1400         LDA (IOB.PNTR),Y
 1410         STA VOLUME
 1420         LDA #$AA     SET UP CONSTANT IN PAGE ZERO
 1430         STA CONST.AA FOR TIMING
 1440         LDY #86      CLEAR BUFFER TO ALL 00'S
 1450         LDA #0
 1460         STA TRACK.CNTR
 1470  .1     STA RWTS.BUFFER+255,Y
 1480         DEY          UPPER PORTION
```

```
1490        BNE .1
1500  .2     STA RWTS.BUFFER,Y
1510        DEY          LOWER PORTION
1520        BNE .2
1530        LDA #80      SET UP AS THOUGH IN TRACK 80
1540        JSR SETUP.TRACK
1550        LDA #40      START WITH 40 SYNC'S BTWN SECTORS
1560        STA SYNC.CNT
1570  *---------------------------------
1580  .3     LDA TRACK.CNTR
1590        JSR SEEK.TRACK
1600        JSR FORMAT.TRACK
1610        LDA #ERR.CANT.FORMAT
1620        BCS .5       ERROR
1630        LDA #48      TRY UP TO 48 TIMES
1640        STA RETRY.COUNT
1650  .4     SEC
1660        DEC RETRY.COUNT
1670        BEQ .5       OUT OF RETRIES, ERRCODE=$30
1680        JSR READ.ADDRESS
1690        BCS .4       ERROR, TRY AGAIN
1700        LDA SECTOR
1710        BNE .4       MUST BE SECOTR 0
1720        JSR READ.SECTOR
1730        BCS .4       ERROR, TRY AGAIN
1740        INC TRACK.CNTR   NEXT TRACK
1750        LDA TRACK.CNTR
1760        CMP #35      FINISHED?
1770        BCC .3       NOT YET
1780        CLC          INDICATE NO ERROR
1790        BCC .6       ...ALWAYS
1800  *---------------------------------
1810  .5     LDY #13      POINT AT ERROR SLOT IN IOB
1820        STA (IOB.PNTR),Y
1830        SEC          FLAG ERROR
1840  .6     LDA MOTOR.OFF,X   STOP DRIVE
1850        RTS
1860  *---------------------------------
1870  *      FORMAT A TRACK
1880  *---------------------------------
1890  FORMAT.TRACK
1900        LDA #0       START WITH SECTOR 0
1910        STA FMT.SECTOR
1920        LDY #128     EXTRA SYNC'S BEFORE FIRST SECTOR
1930        BNE .2       ...ALWAYS
1940  .1     LDY SYNC.CNT
1950  .2     JSR WRITE.ADDRESS
1960        BCS .10      ERROR, EXIT NOW
1970        JSR WRITE.SECTOR
1980        BCS .10      ERROR, EXIT NOW
1990        INC FMT.SECTOR  NEXT SECTOR
2000        LDA FMT.SECTOR
2010        CMP #16      FINISHED WITH THIS TRACK?
2020        BCC .1       NOT YET
```

```
2030  *--------------------------------
2040  *      VERIFY THE TRACK
2050  *--------------------------------
2060         LDY #15      START WITH SECOTR 15
2070         STY FMT.SECTOR
2080         LDA #48      RETRY UP TO 48 TIMES
2090         STA RETRY.COUNT
2100  .3     STA SECTOR.FLAGS,Y  CLEAR ALL THE SECTOR FLAGS
2110         DEY
2120         BPL .3
2130         LDY SYNC.CNT DELAY A WHILE
2140  .4     JSR .10      12 CYCLES
2150         JSR .10      12 CYCLES
2160         JSR .10      12 CYCLES
2170         PHA          PHA+PLA=7 CYCLES
2180         PLA
2190         NOP          NOP+DEY+BNE=7 CYCLES
2200         DEY
2210         BNE .4       WHOLE LOOP = 50 CYCLES
2220         JSR READ.ADDRESS
2230         BCS .8       ERROR, TRY AGAIN
2240         LDA SECTOR   BETTER BE SECTOR 0
2250         BEQ .6       IT IS, HURRAY!
2260         LDA #16      REDUCE # SYNC'S BY TWO
2270         CMP SYNC.CNT UNLESS ALREADY < 16
2280         LDA SYNC.CNT
2290         SBC #1
2300         STA SYNC.CNT
2310         CMP #5       IF SYNC.CNT < 5, THERE IS NO HOPE
2320         BCS .8       >=5, TRY AGAIN
2330         SEC          FLAG COULDN'T DO IT
2340         RTS
2350  .5     JSR READ.ADDRESS
2360         BCS .7       ERROR, TRY AGAIN
2370  .6     JSR READ.SECTOR
2380         BCC .11      GOOD!
2390  .7     DEC RETRY.COUNT
2400         BNE .5       TRY AGAIN
2410  .8     JSR READ.ADDRESS
2420         BCS .9
2430         LDA SECTOR
2440         CMP #15      SECTOR = 15?
2450         BNE .9       NO
2460         JSR READ.SECTOR
2470         BCC FORMAT.TRACK
2480  .9     DEC RETRY.COUNT
2490         BNE .8       TRY AGAIN
2500         SEC          FLAG WE COULDN'T DO IT
2510  .10    RTS          RETURN
2520  *--------------------------------
2530  .11    LDY SECTOR
2540         LDA SECTOR.FLAGS,Y
2550         BMI .7       ALREADY READ THIS ONE!
2560         LDA #$FF
```

```
2570        STA SECTOR.FLAGS,Y
2580        DEC FMT.SECTOR
2590        BPL .5
2600        LDA TRACK.CNTR
2610        BNE .12
2620        LDA SYNC.CNT
2630        CMP #16
2640        BCC .10
2650        DEC SYNC.CNT
2660        DEC SYNC.CNT
2670 .12    CLC
2680        RTS
2690 *---------------------------------
2700 SECTOR.FLAGS
2710        .HS FFFFFFFFFFFFFFFF
2720        .HS FFFFFFFFFFFFFFFF
2730 *---------------------------------
2740 PHYSICAL.SECTOR.VECTOR
2750        .HS 000D0B0907050301
2760        .HS 0E0C0A080604020F
2770 *---------------------------------
2780 *      CLOBBER WHATEVER IS IN RAM CARD
2790 *---------------------------------
2800 PATCH1 JSR $FE93    WHAT PATCH REPLACED
2810        LDA $C081    WRITE-ENABLE RAM CARD
2820        LDA $C081
2830        LDA #0       PUT ZERO IN BYTE WE LATER
2840        STA $E000    TEST TO SEE WHICH LANGUAGE
2850        JMP $B744    RETURN
2860 *---------------------------------
2870 *---------------------------------
2880 *      VARIOUS ODDS AND ENDS
2890 *---------------------------------
2900        .HS 000000
2910 PATCH2 STA $AA63
2920        STA $AA70
2930        STA $AA71
2940        RTS
2950 PATCH3 JSR $A75B
2960        STY $AAB7
2970        RTS
2980 PATCH4 JSR $AE7E    FROM $B377
2990        LDX $B39B
3000        TXS
3010        JSR $A316
3020        TSX
3030        STX $B39B
3040        LDA #9       "DISK FULL" ERROR
3050        JMP $B385
```

```
==============================================================================
DOCUMENT :AAL-8104:DOS3.3:FastStr.Input.txt
==============================================================================

 1000  *---------------------------------
 1010  *       FAST INPUT STRING ROUTINE
 1020  *       &GET <STRING VARIABLE>
 1030  *       ACCEPTS ANY CHARACTER, UNLIKE NORMAL INPUT
 1040  *---------------------------------
 1050  AMPERSAND.VECTOR    .EQ $3F5
 1060  LENGTH .EQ $9D
 1070  SYNTAX.ERROR .EQ $DEC9
 1080  PTRGET .EQ $DFE3
 1090  GETSPA .EQ $E452
 1100  MOVSTR .EQ $E5E2
 1110  *---------------------------------
 1120  MON.PROMPT .EQ $33
 1130  MON.RDLINE .EQ $FD6F
 1140  *---------------------------------
 1150         .OR $300
 1160         LDA #$4C     JUMP INSTRUCTION
 1170         STA AMPERSAND.VECTOR
 1180         LDA #GET
 1190         STA AMPERSAND.VECTOR+1
 1200         LDA /GET
 1210         STA AMPERSAND.VECTOR+2
 1220         RTS
 1230  *---------------------------------
 1240  GET    CMP #$BE     GET TOKEN
 1250         BEQ .1       YES
 1260         JMP SYNTAX.ERROR
 1270  .1     JSR $B1
 1280         JSR PTRGET   GET STRING DESCRIPTOR
 1290         LDA MON.PROMPT
 1300         PHA
 1310         LDA #$87     BELL FOR PROMPT
 1320         STA MON.PROMPT
 1330         JSR MON.RDLINE   INPUT A LINE
 1340         PLA
 1350         STA MON.PROMPT
 1360         STX LENGTH   SAVE LENGTH
 1370         TXA
 1380         JSR GETSPA   GET SPACE IN STRING AREA
 1390         LDY #0       MOVE DATA INTO VARIABLE
 1400         STA ($83),Y  LENGTH
 1410         LDA $71
 1420         INY
 1430         STA ($83),Y  LO-BYTE OF ADDRESS
 1440         LDA $72
 1450         INY
 1460         STA ($83),Y  HI-BYTE OF ADDRESS
 1470         LDY /$200    SET UP TO COPY STRING DATA
 1480         LDX #$200    INTO STRING AREA
```

```
1490        LDA LENGTH
1500        JMP MOVSTR   COPY IT NOW
```

==================================================================
DOCUMENT :AAL-8104:DOS3.3:Substr.search.txt
==================================================================

```
 1010  *--------------------------------
 1020  *
 1030  *       SUBSTRING SEARCH FUNCTION FOR APPLESOFT
 1040  *       ---------------------------------------
 1050  *
 1060  *       & SUB$( A$, B$, I )
 1070  *
 1080  *       SEARCHES FOR FIRST OCCURRENCE OF
 1090  *       B$ IN A$; PUTS RESULT IN I
 1100  *
 1110  *       RETURNS I=0 IF B$ IS NOT IN A$
 1120  *
 1130  *       (REFERENCE:  CALL A.P.P.L.E. ARTICLE
 1140  *       IN JANUARY 1981 ISSUE BY LEE REYNOLDS,
 1150  *       PAGES 26-30.)
 1160  *
 1170  *--------------------------------
 1180  FACMO       .EQ $A0
 1190  TEMPPT      .EQ $52
 1200  MAIN.LENGTH .EQ $18
 1210  MAIN        .EQ $19,1A
 1220  KEY.LENGTH  .EQ $1B
 1230  KEY         .EQ $1C,1D
 1240  *--------------------------------
 1250  ASSIGN .EQ $DA5C    STORE VALUE IN VARIABLE
 1260  SYNCHR .EQ $DEC0    REQUIRE (A) AS NEXT CHAR
 1270  FRMEVL .EQ $DD7B    EVALUATE FORMULA
 1280  SYNCOM .EQ $DEBE    REQUIRE COMMA
 1290  SYNRPN .EQ $DEB8    REQUIRE ")"
 1300  CHKSTR .EQ $DD6C    REQUIRE STRING
 1310  PTRGET .EQ $DFE3    GET POINTER
 1320  FRETMP .EQ $E604    FREE TEMPORARY STRING
 1330  SNGFLT .EQ $E301    FLOAT (Y)
 1340  *--------------------------------
 1350         .OR $300
 1360         .TF B.SUBSTRING SEARCH
 1370  *--------------------------------
 1380  SETUP.AMPERSAND
 1390         LDA #$4C     JMP OPCODE
 1400         STA $3F5
 1410         LDA #SUB
 1420         STA $3F6
 1430         LDA /SUB
 1440         STA $3F7
 1450         RTS
 1460  *--------------------------------
 1470  SUBQT  .AS "($BUS"  SUB$( BACKWARDS
 1480  *--------------------------------
 1490  SUB
```

```
1500        LDX #4       COMPARE FOR "SUB$("
1510 .1     LDA SUBQT,X
1520        JSR SYNCHR   COMPARE WITH INPUT
1530        DEX
1540        BPL .1
1550 *----------------------------------
1560        LDY #MAIN.LENGTH
1570        JSR GET.STRING
1580        LDY #KEY.LENGTH
1590        JSR GET.STRING
1600        JSR PTRGET   GET VARIABLE FOR RESULT
1610        STA $85
1620        STY $86
1630        JSR SYNRPN   REQUIRE RIGHT PAREN
1640 *----------------------------------
1650        JSR FREE.STRINGS
1660 *----------------------------------
1670        LDX #0       ANSWER OFFSET
1680 .2     LDA MAIN.LENGTH SEE IF IT CAN STILL FIT
1690        CMP KEY.LENGTH
1700        BCC .8       WILL NOT FIT
1710        LDY #0
1720 .3     LDA (KEY),Y
1730        CMP (MAIN),Y
1740        BNE .6
1750        INY
1760        CPY KEY.LENGTH
1770        BCC .3
1780        INX          X IS RESULT
1790        TXA
1800        TAY
1810 .4     JSR SNGFLT   FLOAT THE BYTE IN Y
1820        LDA $12
1830        PHA
1840        LDA $11
1850        JMP ASSIGN   STORE VALUE IN VARIABLE
1860 .6     INC MAIN
1870        BNE .7
1880        INC MAIN+1
1890 .7     INX
1900        DEC MAIN.LENGTH
1910        BNE .2
1920 .8     LDY #0       RESULT IS 0
1930        BEQ .4       ...ALWAYS
1940 *----------------------------------
1950 *      GET STRING EXPRESSION
1960 *----------------------------------
1970 GET.STRING
1980        STY GS2      PLUG OUTPUT VECTOR
1990        JSR FRMEVL   EVALUATE FORMULA
2000        JSR SYNCOM   REQUIRE TRAILING COMMA
2010        JSR CHKSTR   REQUIRE STRING
2020        LDY #2       GET STRING DATA
2030 * THE NEXT LINE IS A "SECRET" 6502 OPCODE,
```

```
2040  * WHICH DOES BOTH LDA (FACMO),Y AND LDX (FACMO),Y
2050  * AT THE SAME TIME.
2060  GS1    .DA #$B3,#FACMO
2070         STX *-*,Y    PLUGGED IN FROM ABOVE
2080  GS2    .EQ *-1
2090         DEY
2100         BPL GS1
2110         RTS
2120  *---------------------------------
2130  *      FREE UP ANY TEMPORARY STRINGS
2140  *---------------------------------
2150  FREE.ONE.STRING
2160         LDA TEMPPT+1
2170         LDY #0
2180         JSR FRETMP
2190  FREE.STRINGS
2200         LDA TEMPPT
2210         CMP #$56     EMPTY?
2220         BCS FREE.ONE.STRING
2230         RTS
```

```
================================================================================
DOCUMENT :AAL-8104:DOS3.3:Test.Str.Input.txt
================================================================================
```

```
å768A$–"ABC" -ØæA$((∫A$
```

```
==============================================================================
DOCUMENT :AAL-8104:DOS3.3:Test.Subst.Srch.txt
==============================================================================

*
∫Á(4)"BLOAD B.SUBSTRING
SEARCH":å768údÉASM,DELETE,FAST,FIND,HIDE,INCREMENT,LIST,LOAD,MEMORY,ME
RGE,MGO,NEW,PRT,RENUMBER,RESTORE,SAVE,SLOW,USER,VAL,.¢nÆ¬xÑ"KEY
STRING: ";K$:≠K$–""ƒÄŸÇáA$:≠A$–"."ƒ∫:´110ıåØSUB$(A$,K$,I):≠I–0ƒ130
     ñ≠Iœ1ƒ∫Ë(A$,I…1);     †û:∫K$;:ù<       ™L–
„(A$)…I»1…„(K$):≠Lœ0ƒ∫È(A$,L);G      ¥∫:´130
```

==============================================================================
DOCUMENT :AAL-8105:Articles:DontBeShiftless.txt
==============================================================================

Don't Be Shiftless

Now for another article aimed at that half of you who are really new to 6502 assembly language!

Sliding the bits in a byte back and forth, to the left or the right, is one of the traditional things computers like to do. Big computers have fancy instructions for doing it in many different ways, with special effects along the way. The 6502 only has four "shift" opcodes, so we have to work harder to get all the types of shifting our programs need.

Why shift anything? For various reasons, to suit your fancy. Since data in a byte is normally construed as a binary number, a shift left one bit-position will double the value and a shift right one bit-position will halve the value. If it is important to isolate a particular bit field out of a byte, and then to left or right justify the value which was stored in that field so that testing or arithmetic can be performed, you need shifting instructions. In order to implement multiply and divide on the 6502 you need shifting instructions. To position data for insertion into a bit field within a byte you need to shift. And more.

Show me a picture of a shift. Well, the 6502 makes that easy, because it is limited to shifting a byte to the left or the right, one bit-position at a time.

First let's look at the LSR instruction, which shifts right one bit-position. "LSR" stands for "Logical Shift Right". LSR will shift the contents of a byte one bit-position to the right, like this:

```
     Old value:   1 0 0 1 1 1 0 1

     <Do LSR>

     New value:   0 1 0 0 1 1 1 0
```

LSR shifts in a zero-bit on the left end; the bit that is shifted out the right end goes into the CARRY status bit.

In the sample above the binary value of the old byte is $9D in hex, or 157 decimal. After shifting, the value is $4E hex or 78 decimal (157/2 = 78.5).

The fact the the bit shifted "out" goes into the CARRY status bit makes it possible to test what that bit was. For example, if you need to test a byte to see if it is even or odd, you can LSR it once and then do BCC or BCS to test the carry bit. If carry is set, the number was odd; if clear, it was even. The bit stored in CARRY can have other uses we will discover later.

Now let's see the ASL ("Arithmetic Shift Left") do its thing. It will shift a byte one bit-position to the left, with a zero coming in the right end. The bit shifted out the left end goes into the CARRY status bit. See the similarity to the LSR instruction?

```
     Old value:  0 0 0 1 1 1 0 1

     <Do ASL>

     New value:  0 0 1 1 1 0 1 0
```

Note that the value is doubled; $1D (29) became $3A (58). This will not always be true; if the bit shifted out was a 1-bit, it will be doubled modulo 256. Integer BASIC users will know what that means, because they have the MOD function. For Applesoft-only people, it will mean here that the result is 256 less than the doubled value should be. Let's see an example: shifting 10011101 with ASL produces 00111010; $9D (157) becomes $3A (58), which is 256 less than 2*157.

More about the carry bit. Suppose I want to see if the third bit in a byte is 1 or 0. If the bit positions are numbered left to right from 7 down to 0 (like this: 7 6 5 4 3 2 1 0), I want to test bit 5. If I do three ASL's in a row, bit 5 will be in the CARRY status bit, and I can test it. Or, I could do two ASL's in a row, and look at the MINUS status bit. After a shift, the MINUS status bit is set if the new bit 7 is a 1-bit, or cleared if bit 7 is a 0-bit. The BPL and BMI instructions test the MINUS status bit.

There are two more shift instructions to look at: ROL and ROR. "ROL" is pronounced like a type of bread you eat at dinner, and "ROR" like the noise those giant cats at the zoo make. "ROL" stands for "Rotate One Left"; "ROR" means "Rotate One Right". They work just like LSR and ASL, except for what is shifted in to the byte. LSR shifts a zero-bit in the left end, and ASL shifts a zero-bit in the right end. ROL and ROR shift the old CARRY status bit in, just before the shifted-out bit comes into the CARRY bit.

```
                      Byte          Carry
     Old value:  1 0 0 1 1 1 0 1      1

     <Do ROL>

     New value:  0 0 1 1 1 0 1 1      1

     <Do ROL>

     New value:  0 1 1 1 0 1 1 1      0

     <Do ROL>

     New value:  1 1 1 0 1 1 1 0      0
```

```
     <Do ROR>

     New value:  0 1 1 1 0 1 1 1     0

     <Do ROR>

     New value:  0 0 1 1 1 0 1 1     1

     <Do ROR>

     New value:  1 0 0 1 1 1 0 1     1

     <Do ROR>

     New value:  1 1 0 0 1 1 1 0     1
```

What about shifting values which take two bytes? We can do it using combinations of the four opcodes. Suppose you want to shift a 16-bit value stored at $1234 and $1235 left one bit-position. You want a zero to enter the least significant bit position, which is bit 0 of $1234. You want the most significant bit, bit 7 of $1235, to be in CARRY when you are through. Here is the program:

```
     ASL $1234    0 --> bit 0, bit 7 --> CARRY
     ROL $1235    CARRY --> bit 0, then bit 7 into CARRY
```

Simple, isn't it!

Addressing Modes. The four shift instructions all have the same five addressing modes. There is a one-byte form which shifts the A-register. Some assemblers write this as "ASL A", and don't allow "A" to be used as a label elsewhere. The S-C ASSEMBLER II writes it as just "ASL", so you can use "A" as a label elsewhere if you wish. The other addressing modes are: zero page direct; zero page,X; absolute; and absolute,X. No indirect modes, or indexing by Y modes are available.

[If you remember the article a few months ago about the "secret" opcodes, you will also remember that the two indirect-indexed modes and the absolute,Y mode are available if you don't mind what happens to the A-register after the shift. Or, if what does happen is something you also wanted. You might look up the article.]

Some real examples. The Apple Monitor ROM has some good examples in it. Disassemble (or look in the Monitor listing in the Reference Manual) at $FBC1 (the BASCALC subroutine. If you have the old Monitor ROMs, the multiply and divide subroutines at $FB60 and $FB81 are good examples. The PRBYTE subroutine at $FDDA uses four LSR's to get at the first hex digit. The subroutine DIG at $FF8A is used to convert ascii hex numbers to binary. Let's look at that one here:

```
FF8A: A2 03     DIG    LDX #$03   LOOP 4 TIMES
FF8C: 0A               ASL        LEFT JUSTIFY DIGIT VALUE
```

```
FF8D: 0A                ASL
FF8E: 0A                ASL
FF8F: 0A                ASL
FF90: 0A         NXTBIT ASL         SHIFT DIGIT INTO A2L,A2H
FF91: 26 3E             ROL A2L
FF93: 26 3F             ROL A2H
FF95: CA                DEX
FF96: 10 F8             BPL NXTBIT
```

The ASCII value of the hex digit has already been modified so that the digit's value is in bits 3-0. The first four ASL's shift those 4 bits up to bits 7-4. The next ASL shifts the top bit into CARRY, and then the two ROL's shift that bit into the 16-bit value at A2L and A2H. The ASL-ROL-ROL loop is done four times, so all four bits are shifted into A2L,A2H.

In the Applesoft ROMs there is a subroutine which shifts a 32-bit value right any number of bit-positions. The subroutine is used in the floating point arithmetic package to adjust mantissas. It has the interesting feature (for speed's sake) of shifting 8 bits at a time until the shift count is less than 8. This is done by moving bytes with LDY-STY pairs. The code is at $E8DC thru $E912. The normal entry point is at $E8F0, with the number of bit-positions to be shifted in the A-register as a negative number, and with CARRY clear. The code above $E8F0 shifts right by bytes, and the code after $E8F0 shifts right by bits. The data to be shifted is in page zero, offset by the value in the X-register.

A somewhat similar subroutine is used to normalize the mantissa after a calculation. "Normalize" means to shift the mantissa left until the most significant bit is a one-bit. This code is at $E82E-E854 and $E874-E880. The first portion shifts left by bytes until the leading byte is non-zero (or until it has been determined that the whole value is zero). Once the leading byte is found to be non-zero, the second portion of code shifts left by bits until the leading bit is 1. The number of bit-positions shifted is counted as the subroutine moves along, and that value is subtracted from the exponent value of the floating point number ($E882-E88B).

Disassemble the routines I have pointed out in the various ROMs, and study them a while. Then try writing some of your own examples. Here is an assignment: write a subroutine that will shift a 16-bit value left or right from 0-15 bit positions. The value to be shifted is in page zero at $9D and $9E. The shift count is in the A-register. If the value in A is zero, return without doing anything. If A is negative, it indicates a shift right. If A is positive, it means to shift left. Okay? Give it a try!

================================================================
DOCUMENT :AAL-8105:Articles:DOS321.B800.Lst.txt
================================================================

Commented Listing of DOS 3.2.1 $B800-BCFF

Here is the third installment of DOS disassembly, covering the routines called by RWTS.

There are six major subroutines between $B800 and BCFF. PRE.NYBBLE and POST.NYBBLE convert between memory format and disk format. READ.ADDRESS reads the next address header. READ.SECTOR reads a sector, and WRITE.SECTOR writes one. SEEK.TRACK.ABSOLUTE moves the head in or out to the desired track. With the sole exception of initializing a disk, all actual disk I/O is done by these six subroutines.

The bits that are written on the disk are considerably different from those in memory. Some computer systems make the transformation with expensive hardware controllers, but Wozniak's unique system does most of the work in software. The 13-sector controller cannot read accurately data which has two or more consecutive zero-bits. Of course, almost every byte you want to write has two or more zero-bits in a row! Therefore the software must encode the bytes you want to write.

One way to encode the bytes is to take four bits at a time, and interleave them with "clock" bits. In fact, the data in the address headers is recorded this way. For example, to record the byte "xyxyxyxy" in an address header, the two bytes "1x1x1x1x" and "1y1y1y1y" will be written. This means a 256-byte sector will take 512 bytes on the disk surface (plus header and trailer).

DOS 3.2.1 (and previous versions) use a more elaborate scheme. Each 256-byte sector is recorded as 410 bytes on the disk surface. The subroutine PRE.NYBBLE converts the 256-byte buffer to 410 bytes of 5-bits each. then the 5-bit values are converted to 8-bit values from NYBBLE.TABLE. These 8-bit values are chosen carefully; they have the following properties: 1) the first bit is "1"; 2) no consecutive zero-bits; and 3) the values $AA and $D5 are not used. As a sector is read back into memory, BYTE.TABLE is used to convert the 8-bit codes back to 5-bit values. POST.NYBBLE converts the 410 5-bit values back to 256 8-bit bytes.

In case you are curious, PRE.NYBBLE moves the bits from 256-bytes to 410 bytes like this:

1. The first 5 bytes are rearranged into 8 bytes:

```
      5 input bytes       8 output bytes
     7 6 5 4 3 2 1 0     7 6 5 4 3 2 1 0

     A A A A A B B B     0 0 0 A A A A A
```

```
    C C C C C D D D     0 0 0 C C C C C
    E E E E E F F F     0 0 0 E E E E E
    G G G G G H I J     0 0 0 G G G G G
    K K K K K L M N     0 0 0 K K K K K
                        0 0 0 B B B H L
                        0 0 0 D D D I M
                        0 0 0 F F F J N
```

2. The 8 bytes are stored at the end of the 8 sections (at BB32, BB65, BB98, BBCB, BC32, BC65, AND BC98).

3. The second group of 5 bytes is rearranged into 8 bytes, and stored right before the first 8 (at BB31, BB64, ..., BC97).

4. The next 49 groups of 5 bytes are treated in the same way, with the last group being stored at BB00, BB33, BB66, BB99, BBCC, BC00, BC33, AND BC66.

5. The top 5 bits of the last byte are stored at BBFF, and the bottom 3 bits of the last byte are stored at BC99.

DOS 3.3 uses an even better scheme, but it requires a change in the controller ROMs. The change to one ROM gives you a different boot program; the other ROM makes the controller able to read two consecutive zero-bits accurately. (Note that SOME controller-drive combinations may be able to read two zero-bits in a row accurately WITHOUT the new ROM. Anyway, mine works!) DOS 3.3 converts the 256 bytes to 342 6-bit values; since each sector is shorter, more sectors can be written in each track. I may publish the disassembly of these same subroutines in the DOS 3.3 version next month.

Remember that DOS 3.2.1 puts 13-sectors on each track, with each sector having this format: sync bytes, address header, sync bytes, data block. Sync bytes are written to automatically synchronize the reading process, so that we can be sure we are not splitting bytes. Each sync byte is 8 one-bits followed by 1 zero-bit. The address header is 14-bytes long on the disk surface, and looks like this (in hex): D5 AA B5 vv vv ss ss tt tt cc cc DE AA EB. "vv vv" stands for the two bytes used to record the volume number; "ss ss" is the sector number; "tt tt" is the track number; and "cc cc" is the checksum of the volume, track, and sector. The data block is like this: D5 AA AD <410 bytes of data> <checksum> DE AA EB.

==============================================================
DOCUMENT :AAL-8105:Articles:Front.Page.txt
==============================================================

In This Issue...



Save Your Fingers, Save Your Eyes

Remember that all the source programs which appear in the Apple Assembly Line are available on disk, ready to assembly with the S-C Assembler II Version 4.0. Every three months I collect it all on a Quarterly Disk, and you can get it for only $15. QD#1 covers AAL issues 1-3 (October thru December 1980), and QD#2 covers AAL issues 4-6 (January thru March 1981). QD#3 will be out at the end of May, covering issues 7-9. Some AAL subscribers have chosen to set up a standing order for the Quarterly Disks, so they get them as soon as they are ready.

Not only does it save you a lot of typing time. You also are saved the hours you might spend looking for the inadvertant changes you made while you typed!

Another Utility from RAK-WARE

Bob Kovacs is sure keeping busy! Last month he announced the Cross Reference Utility which works with your S-C ASSEMBLER II source programs. This month he has a Global Search & Replace Utility ready (see his ad on page 4). It is a nice companion to his disassembler, because it gives you a fast way to change all the labels made up by the disassembler into meaningful names.

If You Need Disks...

For a limited time, I am able to offer you a good price on Verbatim DataLife disks. These are bulk packaged, 20 to a pack, with no labels and with white sleeves. They are the same ones I use myself. I will send you a package of 20 for only $50.

================================================================================
DOCUMENT :AAL-8105:Articles:GRAM.WPs.txt
================================================================================

Word Processors...................................Lee Meador

[The following is a summary of the talk given by Lee Meador at our Dallas Apple Corps general session on April 11th (1981). The summary was written by Bob Sander-Cederlof, from notes taken at the meeting.]

What is a Word Processor? I like to think of it as consisting of both a text editing system and a text formatting system. You will see advertisements for so-called word processors which do not include both of these elements, but by my definition they are incomplete.

The text editing system should make it easy for you to enter a large body of text, make corrections and changes, rearrange words and paragraphs, and so on. The text formatter is on the output side; it justifies the text within selected margins, paginates and adds page headings and numbers, and so on.

What is a Word Processor used for? Writing letters, reports, manuals, and even full-length books. Creating forms. Generating personalized form letters. Creating data bases, such as mailing lists. Writing and modifying the source code for computer programs. And whatever your imagination can suggest!

How much does a WP for the Apple cost? Anywhere from $75 to $1000, depending on the features you want. AppleWriter, the one sold by Apple Computer, costs only $75. [I am using AppleWriter now, to write this article. (Bob S-C)] At the other extreme, Word Star costs about $450 for the software, plus another $550 for the required Microsoft Z-80 Card and an 80-column card.

You may find some cheaper than AppleWriter, but I don't have any experience with them.

What features should I look for in a WP? I will break down the features into four categories: screen format, lower case entry/display, commands, and file structure.

Screen Format: Your Apple, straight from the factory, will only display 40 characters per line. You can buy a card to plug into one of the empty slots which will display 80 characters per line. Some word processors require the 80-column card to operate, such as the new Easy-Writer Professional Version and Word Star. Having 80 columns of display is a real advantage, because it lets you see the text the way it will look on your paper. The Pascal Language system editor works a lot easier with an 80-column card, too.

Other WP's get around the 40-column limitation in various ways. AppleWriter merely wraps the lines around at the edge of the screen. Words are split arbitrarily, unless you specifically type a carriage return. Normally you only use the carriage return at the end of a paragraph. This approach seems very crude, but some people like it. The text is re-formatted when you print it out so that lines fit between the margins you select and are broken between words.

Apple-PIE, Super Text II, Scribe, Manuscripter, and others break the lines between words. Most of these allow some sort of preview which allows you to re-format the text exactly as it will be printed by the text formatter, and then to scan around left-right-up-down to see it through the 40-by-24 Apple "window".

Magic Window keeps the text in an 80-column format all the time, and the screen acts as a floating window as you type. Some people like this, but it can be distracting.

Lower-Case Entry/Display: As you know, without some modification or special software, the Apple is UPPER-CASE ONLY. Dan Paymar has made a lot of money because of this oversight by Apple Computer! His $50-60 adapter plugs into your Apple mother board, and gives you lower-case display on the screen. You still have all the other display modes. For two years or more, the Paymar adapter was the only one on the market. Now there are several others, some with additional features.

Another approach to lower-case display is to draw the text on the screen using hi-res graphics. This is the approach used by my word processor (Alphonse II), which works in Hebrew and Greek as well as English. The disadvantages to using hi-res graphics is the extra memory for the hi-res image, and the extra time to draw and scroll.

Most WP's will work without any adapter. They display lower-case characters in normal video, and display upper-case in inverse video. Many WP's are set up to work this way unless you specify that the Paymar Adapter is present. AppleWriter requires a small patch to operate with the Paymar adapter (the patch code is available by writing to Dan Paymar.)

The Apple keyboard is wired up so that the shift key does not distinguish upper/lower case letters. You always get upper case letters, and all the shift key does is allow you to get the punctuation symbols. Some WP's get around this by using control keys or the escape key to swith between upper and lower case modes. These usually have both a shift-lock mode and a captilize-next-character mode. AppleWriter is like this.

Super Text II will work with control-codes for signalling upper case, or with a very simple modification to your Apple you get TRUE shift key operation. You have to solder a wire (provided in the Super Text II package!) on the bottom of the keyboard to one terminal of the shift key. The other end of the wire goes into the game paddle connector, so it can be monitored by software. (Your paddles still work normally.)

If you buy the Videx keyboard expander, you get normal typewriter-like shift key operation. Most 80-column boards probide some means for entering lower-case letters. Yet another option is to replace the entire keyboard in your Apple with a more expensive one having all the features built-in. $$$$!!!!

Commands: There are generally three types of commands you are interested in. Editing commands allow you to move text around, insert new text, add new text, delete text (all or some), correct spelling, replace one word or phrase with another throughout your text, and so on. Most WP's use control-characters for the editing commands. Some use ESCAPE followed by a letter, and others use a menu-driven approach. Naturally, some use a combination of all three.

Formatting commands are usually embedded in the text. Super Text II uses embedded control characters. AppleWriter uses separate lines like "!lm10" (which sets the left margin at 10). Some WP's use a format form which you "fill out" just before printing. The format commands are used to define the left, right, top, and bottom margins, single or double spacing, whether you want lines to be centered or justified, whether you want blanks to be inserted to make all lines the same length, and so on.

Printer commands are usually control-character sequences you want sent to your printer to enable special fonts, underlining, and so on. Some WP's, like AppleWriter, make no provision for these at all. Super Text II does allow you to enter these, although it is a little difficult to set up the first time.

File Structure: The issue here is whether a standard DOS text file is used, or some other format. Apple-PIE is one of the few that uses standard DOS text files. AppleWriter uses binary files, with a strange non-ASCII code and a special beginning-of-text and end-of-text code. Super Text II uses a modified DOS, so that the files are not accessible at all from your own programs. EasyWriter is coded in FORTH, and has its own way of formatting disks which is completely incompatible with DOS. Some WP's "lock" the data disks, so that they cannot be read or written from a normal DOS.

Some utility programs were on the February DOM to convert a standard text file to an AppleWriter file, and vice versa. There are also ways to get at the Super Text II files, but the technique has not been published.
Which Word Processor should I buy? It is entirely up to you. Weigh the various factors such as cost, ease of use, capability, documentation, and so on. Read reviews, such as the excellent one s published in Peelings II, Volume 1, No. 4 Nov-Dec 1980). Talk to owners. (Our club is full of them.) But above all, try several of them out before you buy!

[After this talk, several Apples were set up with a multitude of Word Processors on hand. For two hours Lee and others demonstrated and explained the features, advantages, and disadvantages of these to whoever was interested.]

==============================================================================
DOCUMENT :AAL-8105:Articles:Hires.Scrn.Func.txt
==============================================================================

Hi-Res SCRN Function for Applesoft

Apple's Lo-Res graphics capability includes a SCRN(X,Y) function, to determine the color currently on the screen at the given X,Y point. For some reason they did not provide the corresponding HSCRN(X,Y) function for Hi-Res graphics.

The following program implements the HSCRN function using the "&" character. If you write the statement "& HSCRN (A=X,Y)", this program will store either a 1 or a 0 into the variable A. The value 0 will be stored in A if there is not a spot plotted at X,Y; the value 1 will be stored if there is a spot.

Note that HPLOT(X,Y) may not result in a spot being plotted at X,Y; it depends on the HCOLOR you have set. If the HCOLOR is white, a spot will always be plotted; if it is black, a spot will always be erased; the other four colors may or may not plot a spot, depending on position and color.

The &HSCRN statement does not return the actual color, because that is MUCH more difficult to determine. The actual color depends on: whether the adjacent spots are on or off; whether X,Y is in an even or odd byte; whether X,Y is in an even or odd bit; and whether the sign bit of the byte is on or off. If you decide to add the capability to return a color value (0-7), send me a copy for this newsletter!

==============================================================================
DOCUMENT :AAL-8105:Articles:No.Pdl.Jitter.txt
==============================================================================

Conquering Paddle Jitter...........................Brooke Boering

A well-known problem with the paddles supplied with the Apple (at least they USED to be supplied!) concerns their tendency to rock back and forth between two adjacent values. "Jittering" like this can cause problems unless accuracy is unimportant, or unless the effect is somehow pleasing.

One solution to the jitter problem is to force the new paddle reading to move at least two increments from the prior reading. This words, but at the price of lower resolution. Also, it can have subtle side-effects.

A better solution is to keep track of the previous direction of movement, and enforcing the "rule of two" only if the direction is reversed.

The following program demonstrates my solution. It is set up to work with Applesoft, but it would be rather simple to make it directly callable from your own assembly language routines. To use from Applesoft, POKE the paddle number (0-3) at 768, CALL 770, and read the paddle value with PEEK(769).

I set up the following Applesoft program to test the routine, and to compare it with normal paddle readings:

```
     10 POKE 768,0:CALL 770:PRINT PEEK(769):GOTO10
     20 PRINT PDL(0):GOTO20
```

I typed RUN 20 and set the paddle to a jittery position. Then I typed control-C and RUN 10 to test the smoothing subroutine. The program really works!

```
==============================================================
DOCUMENT :AAL-8105:DOS3.3:DOS321.B800BCFF.txt
==============================================================

 1000  *       .LIF
 1010  *-----------------------------------
 1020  *       DOS 3.2.1 $B800 - $BCFF
 1030  *-----------------------------------
 1040          .OR $B800
 1050          .TA $0800
 1060  *-----------------------------------
 1070  BUF.PNTR   .EQ $3E,3F
 1080  CURRENT.TRACK       .EQ $0478
 1090  *-----------------------------------
 1100  *       DISK CONTROLLER ADDRESSES
 1110  *-----------------------------------
 1120  PHOFF  .EQ $C080    PHASE-OFF
 1130  PHON   .EQ $C081    PHASE-ON
 1140  MTROFF .EQ $C088    MOTOR OFF
 1150  MTRON  .EQ $C089    MOTOR ON
 1160  DRV0EN .EQ $C08A    DRIVE 0 ENABLE
 1170  DRV1EN .EQ $C08B    DRIVE 1 ENABLE
 1180  Q6L    .EQ $C08C    SET Q6 LOW
 1190  Q6H    .EQ $C08D    SET Q6 HIGH
 1200  Q7L    .EQ $C08E    SET Q7 LOW
 1210  Q7H    .EQ $C08F    SET Q7 HIGH
 1220  *
 1230  *       Q6    Q7     USE OF Q6 AND Q7 LINES
 1240  *      ----  ----    ---------------------
 1250  *      LOW   LOW     READ (DISK TO SHIFT REGISTER)
 1260  *      LOW   HIGH    WRITE (SHIFT REGISTER TO DISK)
 1270  *      HIGH  LOW     SENSE WRITE PROTECT
 1280  *      HIGH  HIGH    LOAD SHIFT REGISTER FROM DATA BUS
 1290          .PG
 1300  *-----------------------------------
 1310  *       CONVERT 256 BYTES TO 410 5-BIT NYBBLES
 1320  *-----------------------------------
 1330  PRE.NYBBLE
 1340          LDX #50      51 BYTES PER SECTION
 1350          LDY #0       INDEX INTO 256-BYTE BUFFER
 1360  *---BUFFER PART 1, SECTION 1-----
 1370  .1      LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
 1380          STA $26      SAVE HERE FOR LOWER 3 BITS
 1390          LSR          USE TOP 5 BITS
 1400          LSR
 1410          LSR
 1420          STA RWTS.BUFFER.1.1,X
 1430  *---BUFFER PART 1, SECTION 2-----
 1440          INY          NEXT REAL BYTE
 1450          LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
 1460          STA $27      SAVE HERE FOR LOWER 3 BITS
 1470          LSR          USE TOP 5 BITS
 1480          LSR
```

```
1490        LSR
1500        STA RWTS.BUFFER.1.2,X
1510 *---BUFFER PART 1, SECTION 3-----
1520        INY           NEXT REAL BYTE
1530        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
1540        STA $2A       SAVE FOR LOWER 3 BITS
1550        LSR
1560        LSR           USE TOP 5 BITS
1570        LSR
1580        STA RWTS.BUFFER.1.3,X
1590 *---BUFFER PART 1, SECTION 4-----
1600        INY           NEXT REAL BYTE
1610        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
1620        LSR           USE TOP 5 BITS
1630        ROL $2A       BIT 0 INTO $2A
1640        LSR
1650        ROL $27       BIT 1 INTO $27
1660        LSR
1670        ROL $26       BIT 2 INTO $26
1680        STA RWTS.BUFFER.1.4,X
1690 *---BUFFER PART 1, SECTION 5-----
1700        INY           NEXT REAL BYTE
1710        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
1720        LSR           USE TOP 5 BITS
1730        ROL $2A       BIT 0 INTO $2A
1740        LSR
1750        ROL $27       BIT 1 INTO $27
1760        LSR           HOLD BIT 2 IN CARRY-BIT
1770        STA RWTS.BUFFER.1.5,X
1780 *---BUFFER PART 2, SECTION 0-----
1790        LDA $26       APPEND BIT 2 TO $26
1800        ROL
1810        AND #$1F      5-BIT MASK
1820        STA RWTS.BUFFER.2.1,X
1830 *---BUFFER PART 2, SECTION 1-----
1840        LDA $27
1850        AND #$1F
1860        STA RWTS.BUFFER.2.2,X
1870 *---BUFFER PART 2, SECTION 2-----
1880        LDA $2A
1890        AND #$1F
1900        STA RWTS.BUFFER.2.3,X
1910 *--------------------------------
1920        INY           NEXT REAL BYTE
1930        DEX           NEXT BYTE IN EACH SECTION
1940        BPL .1        LOOP UNTIL EACH SECTION FULL
1950 *--------------------------------
1960        LDA (BUF.PNTR),Y  GET LAST REAL BYTE
1970        TAX
1980        AND #7        USE LOWER 3 BITS
1990        STA RWTS.BUFFER.2.4
2000        TXA           NOW GET 5 UPPER BITS
2010        LSR
2020        LSR
```

```
       LSR
       STA RWTS.BUFFER.1.6
       RTS
 .PG
*------------------------------
*      WRITE A SECTOR ON THE DISK FROM RWTS.BUFFER
*------------------------------
WRITE.SECTOR
       SEC          SET IN CASE OF ERROR RETURN
       LDA Q6H,X    Q6 HIGH, Q7 LOW,
       LDA Q7L,X    TO READ WRITE PROTECT STATUS
       BMI .5       DISK IS WRITE PROTECTED
       STX $27      SAVE SLOT #
       STX $0678    HERE, TOO
       LDA RWTS.BUFFER.2.1  FIRST NYBBLE OF DATA
       STA $26      SAVE IT
       LDA #$FF     SYNC BYTE
       STA Q7H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
       ORA Q6L,X    Q6L,Q7H: WRITE ON DISK
       PHA          TIME DELAYS
       PLA
       NOP
       LDY #10      WRITE TEN MORE SYNC BYTES
.1     ORA $26      WASTE TIME
       JSR WRT2     WRITE (A) ON DISK
       DEY
       BNE .1       UNTIL 10 OF THEM
       LDA #$D5     WRITE DATA HEADER
       JSR WRT1
       LDA #$AA
       JSR WRT1
       LDA #$AD
       JSR WRT1
       TYA          A=0
       LDY #154     WRITE 154 NYBBLES
       BNE .3       ...ALWAYS
.2     LDA RWTS.BUFFER.2.1,Y  GET CURRENT NYBBLE AND
.3     EOR RWTS.BUFFER.2.1-1,Y  EOR WITH PREVIOUS NYBBLE
       TAX          USE AS OFFSET INTO TABLE
       LDA NYBBLE.TABLE,X  MAP 5-BITS TO 8-BITS
       LDX $27      GET SLOT AGAIN
       STA Q6H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
       LDA Q6L,X    Q6L,Q7H: WRITE ON DISK
       DEY
       BNE .2       UNTIL ALL BYTES FROM THIS BLOCK DONE
       LDA $26      GET FIRST NYBBLE
       NOP
.4     EOR RWTS.BUFFER.1.1,Y  EOR WITH CURRENT NYBBLE
       TAX          INDEX INTO TABLE
       LDA NYBBLE.TABLE,X  MAP TO 8-BIT VALUE
       LDX $0678    SLOT # AGAIN
       STA Q6H,X    Q6H,Q7L: (A) TO SHIFT REGISTER
       LDA Q6L,X    Q6L,Q7H: WRITE ON DISK
       LDA RWTS.BUFFER.1.1,Y  GET NYBBLE
```

```
2570        INY
2580        BNE .4       MORE TO DO
2590        TAX          LAST NYBBLE
2600        LDA NYBBLE.TABLE,X  MAP TO 8 BITS
2610        LDX $27      SLOT # AGAIN
2620        JSR WRT3     WRITE CHECK SUM ON DISK
2630        LDA #$DE     WRITE TRAILER
2640        JSR WRT1
2650        LDA #$AA
2660        JSR WRT1
2670        LDA #$EB
2680        JSR WRT1
2690        LDA Q7L,X    Q7L
2700 .5     LDA Q6L,X    Q6L
2710        RTS
2720 *---------------------------------
2730 WRT1   CLC          WAIT 2 CYCLES
2740 WRT2   PHA          WAIT 3 CYCLES
2750        PLA          WAIT 4 CYCLES
2760 WRT3   STA Q6H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
2770        ORA Q6L,X    Q6L,Q7H: WRITE ON DISK
2780        RTS
2790  .PG
2800 *---------------------------------
2810 *      READ SECTOR INTO RWTS.BUFFER
2820 *---------------------------------
2830 READ.SECTOR
2840        LDY #32      MUST FIND $D5 WITHIN 32 BYTES
2850 .1     DEY
2860        BEQ ERROR.RETURN
2870 .2     LDA Q6L,X    READ SHIFT REGISTER
2880        BPL .2       WAIT FOR FULL BYTE
2890 .3     EOR #$D5     SEE IF FOUND $D5
2900        BNE .1       NOT YET
2910        NOP          DELAY BEFORE NEXT READ
2920 .4     LDA Q6L,X    READ SHIFT REGISTER
2930        BPL .4       WAIT FOR FULL BYTE
2940        CMP #$AA     SEE IF $AA
2950        BNE .3       NO
2960        LDY #154     BYTE COUNT FOR LATER
2970 .5     LDA Q6L,X    READ SHIFT REGISTER
2980        BPL .5       WAIT FOR FULL BYTE
2990        CMP #$AD     IS IT $AD?
3000        BNE .3       NO
3010 *---------------------------------
3020        LDA #0       BEGIN CHECKSUM
3030 .6     DEY
3040        STY $26
3050 .7     LDY Q6L,X    READ SHIFT REGISTER
3060        BPL .7       WAIT FOR FULL BYTE
3070        EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
3080        LDY $26      BUFFER INDEX
3090        STA RWTS.BUFFER.2.1,Y
3100        BNE .6
```

```
3110  .8      STY $26
3120  .9      LDY Q6L,X    READ SHIFT REGISTER
3130          BPL .9       WAIT FOR FULL BYTE
3140          EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
3150          LDY $26
3160          STA RWTS.BUFFER.1.1,Y
3170          INY
3180          BNE .8
3190  .10     LDY Q6L,X    READ CHECKSUM
3200          BPL .10
3210          CMP BYTE.TABLE,Y
3220          BNE ERROR.RETURN
3230  .11     LDA Q6L,X    READ TRAILER
3240          BPL .11
3250          CMP #$DE
3260          BNE ERROR.RETURN
3270          NOP
3280  .12     LDA Q6L,X
3290          BPL .12
3300          CMP #$AA
3310          BEQ GOOD.RETURN
3320  ERROR.RETURN
3330          SEC
3340          RTS
3350   .PG
3360  *---------------------------------
3370  *       READ ADDRESS
3380  *---------------------------------
3390  READ.ADDRESS
3400          LDY #$F8     TRY 1800 TIMES (FROM $F8F8 TO $10000)
3410          STY $26
3420  .1      INY
3430          BNE .2
3440          INC $26
3450          BEQ ERROR.RETURN
3460  .2      LDA Q6L,X    READ SHIFT REGISTER
3470          BPL .2       WAIT FOR FULL BYTE
3480  .3      CMP #$D5     SEE IF $D5
3490          BNE .1       NO
3500          NOP          DELAY
3510  .4      LDA Q6L,X    READ SHIFT REGISTER
3520          BPL .4       WAIT FOR FULL BYTE
3530          CMP #$AA     SEE IF $AA
3540          BNE .3       NO
3550          LDY #3       READ 3 BYTES LATER
3560  .5      LDA Q6L,X    READ SHIFT REGISTER
3570          BPL .5
3580          CMP #$B5     SEE IF $B5
3590          BNE .3       NO
3600          LDA #0       START CHECK SUM
3610  .6      STA $27
3620  .7      LDA Q6L,X    READ REGISTER
3630          BPL .7
3640          ROL
```

```
3650        STA $26
3660 .8     LDA Q6L,X    READ REGISTER
3670        BPL .8       WAIT FOR FULL BYTE
3680        AND $26      MERGE THE NYBBLES
3690        STA $2C,Y    $2C -- CHECK SUM
3700        EOR $27      $2D -- SECTOR
3710        DEY          $2E -- TRACK
3720        BPL .6       $2F -- VOLUME
3730        TAY          TEST CHECK SUM
3740        BNE ERROR.RETURN
3750 .9     LDA Q6L,X    READ REGISTER
3760        BPL .9       WAIT FOR FULL BYTE
3770        CMP #$DE     TEST FOR VALID TRAILER
3780        BNE ERROR.RETURN
3790        NOP
3800 .10    LDA Q6L,X    READ REGISTER
3810        BPL .10
3820        CMP #$AA
3830        BNE ERROR.RETURN
3840 GOOD.RETURN
3850        CLC
3860        RTS
3870        .PG
3880 *---------------------------------
3890 *      CONVERT 410 5-BIT NYBBLES TO 256 BYTES
3900 *      (THEY ARE NOW LEFT-JUSTIFIED IN RWTS.BUFFER)
3910 *---------------------------------
3920 POST.NYBBLE
3930        LDX #50      51 BYTES PER SECTION
3940        LDY #0
3950 *---BUFFER PART 1, SECTION 1-----
3960 .1     LDA RWTS.BUFFER.2.1,X
3970        LSR
3980        LSR          RIGHT-JUSTIFY THE NYBBLE
3990        LSR
4000        STA $27      SAVE BIT 0
4010        LSR
4020        STA $26      SAVE BIT 1
4030        LSR          BITS 2-4
4040        ORA RWTS.BUFFER.1.1,X
4050        STA (BUF.PNTR),Y  STORE IN BUFFER
4060 *---BUFFER PART 1, SECTION 2-----
4070        INY          NEXT BYTE
4080        LDA RWTS.BUFFER.2.2,X
4090        LSR          RIGHT-JUSTIFY THE NYBBLE
4100        LSR
4110        LSR
4120        LSR          BIT 0 INTO CARRY
4130        ROL $27      AND SAVE HERE
4140        LSR          BIT 1 INTO CARRY
4150        ROL $26      AND SAVE HERE
4160        ORA RWTS.BUFFER.1.2,X
4170        STA (BUF.PNTR),Y  STORE THE BYTE
4180 *---BUFFER PART 1, SECTION 3-----
```

```
       INY          NEXT BYTE
       LDA RWTS.BUFFER.2.3,X
       LSR          RIGHT-JUSTIFY THE NYBBLE
       LSR
       LSR
       LSR          BIT 0 INTO CARRY
       ROL $27      AND SAVE HERE
       LSR          BIT 1 INTO CARRY
       ROL $26      AND SAVE HERE
       ORA RWTS.BUFFER.1.3,X
       STA (BUF.PNTR),Y  STORE THE BYTE
*---BUFFER PART1, SECTION 4-----
       INY          NEXT BYTE
       LDA $26      USE THE 3 BITS SAVED HERE
       AND #7       MAKE SURE ONLY 3 BITS
       ORA RWTS.BUFFER.1.4,X
       STA (BUF.PNTR),Y  STORE THE BYTE
*---BUFFER PART1, SECTION 5-----
       INY          NEXT BYTE
       LDA $27      USE THE 3 BITS SAVED HERE
       AND #7       MAKE SURE ONLY 3 BITS
       ORA RWTS.BUFFER.1.5,X
       STA (BUF.PNTR),Y  STORE THE BYTE
*---------------------------------
       INY          NEXT BYTE
       DEX
       BPL .1
*---------------------------------
       LDA RWTS.BUFFER.2.4  GET THE LAST BYTE
       LSR          RIGHT JUSTIFY
       LSR
       LSR
       ORA RWTS.BUFFER.1.6
       STA (BUF.PNTR),Y  STORE THE LAST BYTE
       RTS
 .PG
*---------------------------------
*      TRACK SEEK
*---------------------------------
SEEK.TRACK.ABSOLUTE
       STX $2B      CURRENT SLOT*16
       STA $2A      SAVE TRACK #
       CMP CURRENT.TRACK  COMPARE TO CURRENT TRACK
       BEQ .9       ALREADY THERE
       LDA #0
       STA $26      # OF STEPS SO FAR
.1     LDA CURRENT.TRACK  CURRENT TRACK NUMBER
       STA $27
       SEC
       SBC $2A      DESIRED TRACK
       BEQ .6       WE HAVE ARRIVED
       BCS .2       CURRENT > DESIRED
       EOR #$FF     CURRENT < DESIRED
       INC CURRENT.TRACK  INCREMENT CURRENT
```

```
4730        BCC .3       ...ALWAYS
4740  .2     ADC #$FE     CARRY SET, SO A=A-1
4750         DEC CURRENT.TRACK  DECREMENT CURRENT TRACK
4760  .3     CMP $26      GET MINIMUM OF:
4770         BCC .4          1. # OF TRACKS TO MOVE LESS 1
4780         LDA $26         2. # OF ITERATIONS SO FAR
4790  .4     CMP #12         3. ELEVEN
4800         BCS .5
4810         TAY
4820  .5     SEC          TURN PHASE ON
4830         JSR .7
4840         LDA ONTBL,Y  GET DELAY TIME
4850         JSR DLY100   DELAY 100*A MICROSECONDS
4860         LDA $27      TRACK NUMBER
4870         CLC          TURN PHASE OFF
4880         JSR .8
4890         LDA OFFTBL,Y
4900         JSR DLY100
4910         INC $26      # OF STEPS SO FAR
4920         BNE .1       ...ALWAYS
4930  *--------------------------------
4940  .6     JSR DLY100
4950         CLC          TURN PHASE OFF
4960  .7     LDA CURRENT.TRACK
4970  .8     AND #3       ONLY KEEP LOW-ORDER 2 BITS
4980         ROL          (0000 0XX0)
4990         ORA $2B      (0SSS 0XX0) MERGE SLOT
5000         TAX          USE AS INDEX FOR PHASE-OFF
5010         LDA PHOFF,X  PHASE-OFF
5020         LDX $2B
5030  .9     RTS
5040  *--------------------------------
5050  *      SHORT DELAY SUBROUTINE
5060  *--------------------------------
5070  DLY100 LDX #17      100*A MICROSECONDS
5080  .1     DEX
5090         BNE .1
5100         INC $46
5110         BNE .2
5120         INC $47
5130  .2     SEC
5140         SBC #1
5150         BNE DLY100
5160         RTS
5170  *--------------------------------
5180  *      DELAY TIMES FOR STEPPING MOTOR
5190  *--------------------------------
5200  ONTBL  .HS 01302824201E1D1C1C1C1C1C
5210  OFFTBL .HS 702C26221F1E1D1C1C1C1C1C
5220         .HS 1C1C1C1C
5230         .PG
5240  *--------------------------------
5250  *      BYTE TABLE
5260  *--------------------------------
```

```
5270  BYTE.TABLE .EQ *-$A8
5280         .HS 0000000001081018020304050620283O
5290         .HS 070938400A4850580B0C0D0E0F111213
5300         .HS 14151617191A1B1C1D1E212223246068
5310         .HS 252670782780889O292A2B2C2D2E2F31
5320         .HS 323398A034A8B0B8353637393AC0C8D0
5330         .HS 3B3CD8E03EE8F0F8
5340  *---------------------------------
5350  *      410-BYTE BUFFER FOR NYBBLES
5360  *---------------------------------
5370  RWTS.BUFFER.1.1 .BS 51  $BB00 - BB32
5380  RWTS.BUFFER.1.2 .BS 51  $BB33 - BB65
5390  RWTS.BUFFER.1.3 .BS 51  $BB66 - BB98
5400  RWTS.BUFFER.1.4 .BS 51  $BB99 - BBCB
5410  RWTS.BUFFER.1.5 .BS 51  $BBCC - BBFE
5420  RWTS.BUFFER.1.6 .BS 1   $BBFF
5430  RWTS.BUFFER.2.1 .BS 51  $BC00 - BC32
5440  RWTS.BUFFER.2.2 .BS 51  $BC33 - BC65
5450  RWTS.BUFFER.2.3 .BS 51  $BC66 -BC98
5460  RWTS.BUFFER.2.4 .BS 1   $BC99
5470  *---------------------------------
5480  *      NYBBLE TABLE
5490  *---------------------------------
5500  NYBBLE.TABLE
5510         .HS ABADAEAFB5B6B7BABBBDBEBF
5520         .HS D6D7DADBDDDEDFEAEBEDEEEF
5530         .HS F5F6F7FAFBFDFEFF
5540  *---------------------------------
5550  *      $BCBA THRU $BCFF IS NOT USED BY DOS 3.2.1
5560  *---------------------------------
5570         .PG
```

```
==============================================================================
DOCUMENT :AAL-8105:DOS3.3:HIRES.SCRN.TEST.txt
==============================================================================

$∫Á(4)"BLOAD B.HIRES SCRN":å7682
ë:í3:ì0,0>ÅI–1¡10V-X–¤(1) 40:Y–¤(1) 40b(ì¡X,Y:Çv<ÅX–0¡39:ÅY–
0¡39äFØH◊A–X,Y):†15 AóPçX,Y:Ç:Ç•Zπ…16298,0ødæA$:π…16297,0:æA$:´90
```

```
==============================================================================
DOCUMENT :AAL-8105:DOS3.3:S.HIRES.SCRN.txt
==============================================================================

 1000  *--------------------------------
 1010  *      HI-RES SCRN FUNCTION
 1020  *
 1030  *      & HSCRN( A=X,Y )
 1040  *      X,Y DEFINES THE SPOT
 1050  *      A RECEIVES 0 OR 1
 1060  *--------------------------------
 1070         .OR $300
 1080         .TF B.HIRES SCRN
 1090  *--------------------------------
 1100  AMPERSAND.VECTOR    .EQ $3F5
 1110  *--------------------------------
 1120  CHRGET              .EQ $00B1
 1130  CHRGOT              .EQ $00B7
 1140  SYNCHR              .EQ $DEC0
 1150  SYNTAX.ERROR        .EQ $DEC9
 1160  PTRGET              .EQ $DFE3
 1170  SNGFLT              .EQ $E301
 1180  HPOSN               .EQ $F411
 1190  HFNS                .EQ $F6B9
 1200  *--------------------------------
 1210  VALUE.TYPE          .EQ $11
 1220  HPNTR               .EQ $26
 1230  HMASK               .EQ $30
 1240  FORMULA.PNTR        .EQ $85
 1250  *--------------------------------
 1260  TOKEN.EQUALS        .EQ $D0
 1270  TOKEN.SCRN          .EQ $D7
 1280  *--------------------------------
 1290  *      SETUP AMPERSAND VECTOR
 1300  *--------------------------------
 1310  SETUP  LDA #$4C     JMP OPCODE
 1320         STA AMPERSAND.VECTOR
 1330         LDA #HSCRN
 1340         STA AMPERSAND.VECTOR+1
 1350         LDA /HSCRN
 1360         STA AMPERSAND.VECTOR+2
 1370         RTS
 1380  *--------------------------------
 1390  *      HSCRN FUNCTION
 1400  *--------------------------------
 1410  HSCRN  LDA #'H      TEST FOR "HSCRN("
 1420         JSR SYNCHR        FIRST LETTER "H"
 1430         LDA #TOKEN.SCRN   AND THEN TOKEN "SCRN("
 1440         JSR SYNCHR
 1450         JSR PTRGET   SCAN THE VARIABLE NAME
 1460         STA FORMULA.PNTR  SAVE ITS POINTER ADDRESS
 1470         STY FORMULA.PNTR+1
 1480         LDA #TOKEN.EQUALS CHECK FOR "="
```

```
       JSR SYNCHR
       LDA VALUE.TYPE+1  SAVE VARIABLE TYPE ON STACK
       PHA
       LDA VALUE.TYPE
       PHA
       JSR HFNS     SCAN "X,Y" EXPRESSIONS
       JSR HPOSN    SET UP BASE, Y-REG, AND MASK
       JSR CHRGOT   CHECK FOR FINAL ")"
       CMP #')
       BNE .2       SYNTAX ERROR IF NOT THERE!
       JSR CHRGET   POSITION FOR NEXT STATEMENT
       LDA HMASK    ISOLATE SPOT AT X,Y
       AND (HPNTR),Y
       BEQ .1       SPOT IS OFF, RETURN ZERO
       LDA #1       SPOT IS ON, RETURN 1
.1     TAY
       JSR SNGFLT   CONVERT BYTE TO REAL VALUE
       JMP $DA5B    STORE IN VARIABLE, AND KEEP GOING!
*---------------------------------
.2     JMP SYNTAX.ERROR
```

```
==============================================================================
DOCUMENT :AAL-8105:DOS3.3:S.PADDLE.JITTER.txt
==============================================================================

 1000  *--------------------------------
 1010  *       PADDLE JITTER SMOOTHER
 1020  *
 1030  *       POKE 768,<PADDLE NUMBER>   0, 1, 2, OR 3
 1040  *       CALL 770
 1050  *       P=PEEK(769)   PADDLE VALUE 0-255
 1060  *--------------------------------
 1070  MON.PREAD  .EQ $FB1E SUBROUTINE TO READ PADDLE
 1080  *--------------------------------
 1090         .OR $300
 1100  *--------------------------------
 1110  PADDLE.NUMBER       .BS 1
 1120  PADDLE.VALUE        .BS 1
 1130  *--------------------------------
 1140  PADDLE.JITTER.SMOOTHER
 1150         LDA PADDLE.NUMBER
 1160         AND #3       BE CERTAIN 0>=PDL#>=3
 1170         TAX
 1180         JSR MON.PREAD READ PADDLE VALUE
 1190         TYA          SAVE IN A-REG TOO
 1200         CPY PADDLE.VALUE.1
 1210         BEQ .8       SAME, RETURN THIS VALUE
 1220         LDX PADDLE.VALUE.1  DETERMINE PREVIOUS DIRECTION
 1230         CPX PADDLE.VALUE.2
 1240         BCS .2       IT WAS INCREASING
 1250  *--------------------------------
 1260  *   IT WAS DECREASING...
 1270  *--------------------------------
 1280         CPY PADDLE.VALUE.1  WHAT IS CURRENT DIRECTION?
 1290         BCC .6       STILL DECREASING, SO ACCEPT IT
 1300         DEY          SEE IF ONLY 1 STEP
 1310         BCS .5       ...ALWAYS
 1320  *--------------------------------
 1330  *   IT WAS INCREASING...
 1340  .2     CPY PADDLE.VALUE.1  DETERMINE CURRENT DIRECTION
 1350         BCS .6       STILL INCREASING, SO ACCEPT IT
 1360         INY          SEE IF ONLY 1 STEP
 1370  *--------------------------------
 1380  *   REVERSED DIRECTION
 1390  *--------------------------------
 1400  .5     CPY PADDLE.VALUE.1  IF SAME NOW, IGNORE IT
 1410         BNE .6       USE NEW VALUE
 1420         TXA          USE PREVIOUS VALUE
 1430         BCS .8       ...ALWAYS
 1440  *--------------------------------
 1450  *   ACCEPT NEW READING
 1460  *--------------------------------
 1470  .6     STX PADDLE.VALUE.2  OLDEST READING
 1480         STA PADDLE.VALUE.1  PREVIOUS READING
```

```
*--------------------------------
.8     STA PADDLE.VALUE    CURRENT READING
       RTS
*--------------------------------
PADDLE.VALUE.1  .DA #0
PADDLE.VALUE.2  .DA #0
*--------------------------------
```

==================================================================
DOCUMENT :AAL-8106:Articles:DOS33.B800.List.txt
==================================================================

Commented Listing of DOS 3.3 $B800-BCFF

As I promised last month, here are the innermost routines of DOS 3.3. These are the ones which actually read and write the hardware, and are the most significantly different routines between DOS 3.2.1 and DOS 3.3.

The major difference between the two versions of DOS is the way in which data bytes are coded on the disk. DOS 3.2.1 maps 256 8-bit bytes into 410 5-bit "nybbles". DOS 3.3 maps 256 8-bit bytes into 342 6-bit "nybbles". (The term "nybble" usually means 4 bits, but Apple uses nybble to mean 5- and 6-bits also.)

The two routines PRE.NYBBLE and POST.NYBBLE convert between memory format and disk format. The DOS 3.3 versions are much shorter and simpler than those of DOS 3.2.1, but they are still hard to visualize and explain.

To write a sector on the disk, RWTS calls PRE.NYBBLE and WRITE.SECTOR. Here is what happens:

1. The most significant 6 bits of each byte in the buffer are copied into $BB00-BBFF and right-justified with two zero-bits on the left.

2. The least significant 2 bits of each buffer byte are mapped into $BC00-BC55.

3. Each 6-bit nybble is used as an index into the NYBBLE.TABLE to pick up a corresponding 8-bit disk code. (The codes in NYBBLE.TABLE always have the first bit = 1, and never have more than two zero-bits in a row.)

To read a sector from the disk, RWTS calls READ.SECTOR and POST.NYBBLE. Here is what happens:

1. Each disk byte is converted to a 6-bit nybble and copied into the buffer from $BB00 through $BC55.

2. The nybbles in $BB00-BBFF become the most significant 6-bits of the buffer bytes.

3. The nybbles in $BC00-BC55 supply the least significant 2-bits for each buffer byte. This is the reverse of the process above.

WRITE.ADDRESS is called from FORMAT, when you are initializing a 16-sector disk. This subroutine was embedded inside FORMAT in DOS 3.2.1. READ.ADDRESS, READ.SECTOR, and WRITE.SECTOR are almost identical to the DOS 3.2.1 versions.

Short as they are, I noticed that both PRE. and POST.NYBBLE can be written more efficiently.  Can you see how to save three bytes in PRE.NYBBLE, and two bytes in POST.NYBBLE?

```
 SECTOR BUFFER          RWTS.BUFFER.1
7 6 5 4 3 2 1 0        7 6 5 4 3 2 1 0
                00                      BB00
            A B

      G                      G


                55
                56
            C D

                AB
                AC
            E F

                FF                      BBFF

 RWTS.BUFFER.2
7 6 5 4 3 2 1 0
                BC00

    F E D C B A
                BC55
```

Another Way to Get 80-Columns

Those unpredictable Apple Parallel Interface ROMs!  I wonder if even Apple knows how many different versions they have made, and why!

Anyway, as you know if you have one, some of them make it very difficult to get 80-column printout when you are using the S-C Assembler II.  You should be able to type control-I and "80N", but the assembler sees control-I and does a tab.  Plus you get a syntax error, and the printer is un-hooked.

You can type "$I80N" (where "I" means control-I).  Or you can type "$579:50" (assuming slot 1).

Or, you can make the first line of your program do it.  Type in this line so it will be the first line in your program:

```
     0000 *I80N
```

Then type the "MEM" command.  It will tell you the memory address where your source program starts.  Using monitor commands, display about 8 bytes at the beginning of the source program.  Look for the pattern "49 38 30 4E".  Change the "49" to "09", which is ASCII for control-I.  When your program is LISTed or ASMed, the control-I will be caught by Apple's interface and put you into 80-column mode.

So, now you have at least three ways to make it work.  Don't you wish you had the ROM version which is in my Apple Parallel card?  It works right without ANY of the above!  Now if I could only make it work with my screen printing program....

=============================================================================
DOCUMENT :AAL-8106:Articles:FancyToneMakers.txt
=============================================================================

Two Fancy Tone Generators...................Mark Kriegsman

I was not quite satisfied with the sound from Bob Sander-Cederlof's "Touch-Tone Simulator" (AAL February 1981, page 5,6). His method for making two simulataneous tones was to play one tone for a while and then the other one for a while, letting your ear put it all together. I have written the following DUAL.TONES program which mixes the two tones together in a more realistic way. I also wrote SINGLE.TONE which plays a given tone at 16 different volume levels. All out of the standard Apple speaker! Really!

The programs are accessed from Applesoft with the "&". (See lines 1510 and 1830.) SINGLE.TONE is called with &T followed by three expressions separated by commas. The three expressions are for the tone, duration, and volume, respectively. Tone is a value from 0 to 255, duration a value from 0 to 65535, and volume a value from 0 to 15. Experiment with different settings and you will see how it works. By making loops which change both pitch and volume, you can simulate the sound of a falling bomb or a passing car.

DUAL.TONES also needs three parameters: tone#1, duration, and tone#2, respectively. The two tone values must be between 0 and 255; duration is again a value from 0 to 65535. It is interesting to try two tone values very close together, to hear the beating effect, and two tones at harmonic intervals to hear the chords. I think &D 254,28000,255 sounds a little like a light saber. Again, a loop which varies both tone values can make some exciting sound effects!

Lines 1340-1400 are executed when you BRUN B.AMPERTONE; they set up the ampersand vector for Applesoft. Once this is done, an ampersand in your program or typed in as a direct command will start executing the AMPERTONE subroutine.

Lines 1440-1490 determine which & routine you are calling. The character following the "&" is in the A-register. If it is "T", SINGLE.TONE is called; if "D", DUAL.TONE is called; if neither, you get SYNTAX ERR.

Subroutines in the Applesoft ROMs are used to read the parameter expressions (lines 2190-2230). GTBYTC advances to the next character, and then evaluates the expression that starts there. If the value is between 0 and 255 it is returned in the X-register. (If not, you get RANGE ERR.) CHKCOM makes sure the next character is a comma; if it isn't, you get SYNTAX ERR. GETNUM is used in executing the POKE statement. It looks for an expression giving a value between 0 and 65535, then a comma, and then another expression giving a value between 0 and 255. The first value is stored at $50 and $51, and the second is returned in the X-register.

[Mark Kriegsman is a 15-year-old Apple expert living in Summit, New Jersey.  I wrote the article above based on two letters and a program he sent.  (Bob Sander-Cederlof)]

===========================================================================
DOCUMENT :AAL-8106:Articles:Front.Page.txt
===========================================================================

Volume 1 -- Issue 9                                June, 1981

In This Issue...



New Quarterly Disk Ready

Remember that all the source programs which appear in the Apple Assembly Line are available on disk, ready to assembly with the S-C Assembler II Version 4.0.  Every three months I collect it all on a Quarterly Disk, and you can get it for only $15.

QD#1 covers AAL issues 1-3 (October thru December 1980), QD#2 covers AAL issues 4-6 (January thru March 1981), QD#3 covers issues 7-9 (April thru June 1981).  Copies of all back issues of the AAL newsletter are available for $1.20 each.

=============================================================================
DOCUMENT :AAL-8106:Articles:Multiplication.txt
=============================================================================

Correction

When I typed Rick Hatcher's code for "Hiding Things Under DOS", AAL April, 1981, page 10, I goofed. Change line 110 of the little Applesoft code from "110 POKE 40194,211" to "110 POKE 40192,211". Better yet, to reserve NP pages between the current bottom of DOS and DOS's buffers, use this code before any files are opened:

```
     100 POKE 40192,PEEK(40192)-NP
     110 CALL 42964
```

More About Multiplying on the 6502

You will remember Brooke Boering's article on this subject in MICRO last December; I mentioned it in AAL#5, and printed his 16x16 multiply subroutine. Now Leo J. Scanlon, author of 6502 Software Design, published an eight-page article "Multiplying by 1's and 0's" in Kilobaud Microcomputing, June 1981, pages 110-120.

If you are serious and really want to learn, this article gets down to the nuts and bolts level. Work your way through it, and you will have learned not only how to multiply, but also a lot about machine language in general. Subroutines are listed for 8x8, 16x16, and NxM multiplication, for both signed and unsigned operands.

Not to be outdone, I have written my own subroutine to multiply an M-byte multiplicand by a N-byte multiplier (both unsigned), producing a product of M+N bytes. It is written for clarity, not for size or speed (nevertheless, it is two bytes shorter than Scanlon's subroutine!).

The basic idea is to examine the bits of the multiplier one-by-one, starting on the right. If the multiplier bit = 1, the multiplicand is added in to the product, at the left end of the product register. In either case, the product register is then shifted right one bit position. The process is repeated until the multiplier is used up.

I wrote subroutines to shift the product register right one bit position, to shift the multiplier right one bit position returning the bit shifted out in the CARRY status bit, and to add the multiplicand to the product register. There is no reason these have to be subroutines; they could be coded in line, because they are only called from one place. I did it to make the overall program easier for you to follow.

The multiplication loop is coded as two loops: an outer loop for the number of bytes in the multiplier, and an inner loop for the number of bits in a byte. This allows me to have up to 255 bytes in the multiplier, just so the product (M+N bytes) is not more than 256 bytes.   (Of course, if you want variables that long, you will have to move them out of page zero.)

There is one little trick you might not notice.  After ACCUMULATE.PARTIAL.PRODUCT, carry will be set if the sum overflows. Then SHIFT.PRODUCT.RIGHT shifts the carry bit back into the product register, maintaining the right answer.

Specialized Multiplications

Sometimes you need a multiplication routine that is not general at all.  For example, when you are converting from decimal to binary, you need a routine that will multiply be ten.  When you are computing the memory address of a character at a particular position on a particular line on the Apple Screen, you need to be able to multiply by 40 and 128.  Other cases may come to your mind.

The subroutine BASCALC in the Apple Monitor computes the address in screen memory.  Here is what it is really doing, written in Integer BASIC:

```
100 ADDR = 1024 + (LINE MOD 8)*128 + (LINE/8)*40
```

To do all that using a generalized multiply routine would take hundreds of microseconds; BASCALC takes only 40 microseconds.  Here is Woz's code, with a few extra comments:

<bascalc subroutine here>

A subroutine to multiply by ten usually takes advantage of the fact that ten in binary is "1010".  That is, 10*X is the same as 8*X + 2*X, or 2*(4*X+X).  In fact, even in machines that have hardware multiply instructions, it is usually faster to multiply by ten using "shift-twice-and-add" than using the built in MPY opcode!

Here is a short piece of code which multiplies a two-byte value by ten, storing the result back in the same bytes.

<example here>

Another way, much less sophisticated, to multiply by ten is to simply add the number to itself nine times.  If you have the S-C ASSEMBLER II Version 4.0, disassemble from $114A through $117A.  You will find my subroutine for converting line numbers to binary.  It is not elegant, but it does the job reasonably fast in a small amount of memory.  A counter is initialized to 10; the next digit is read from the input line and converted from ASCII to binary; the number accumulator is added to the digit ten times, and the sum placed back into the number accumulator.  The counter is in $52, and the number accumulator is in $50,51.

When you are converting from binary to decimal, you need to divide by ten.  Or multiply by one-tenth.  One-tenth written as a binary fraction is ".00011001100110011100....".  Does the repetitive pattern here suggest to you a short-cut way to multiply by one-tenth? Maybe it would become even easier if we write one-tenth as 4/30 - 1/30. In decimal, to 8 places, that looks like .13333333 - .03333333 = .10000000. In binary, to 18 bits, it looks like .001000100010001000 - .000010001000100010 = .000110011001100110. See what you can come up with for a fast way to multiply a 16-bit number by one-tenth. I'll print the best version in AAL!

===============================================================================
DOCUMENT :AAL-8106:Articles:Rvw.Beneath.DOS.txt
===============================================================================

Beneath Apple DOS -- A Review

If you have any interest whatsoever in DOS, be sure to buy this book! It costs $19.95 (plus shipping), from Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335. Call them up at (213) 344-6599 and give them your Master Charge or VISA number. Do it now!

Or better yet, send your check for $18 to S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. I'll mail you a copy postpaid right away! Saves you both time and money!

The authors of Beneath Apple DOS are Don Worth and Pieter Lechner. You may know Don from his adventure-like program, "Beneath Apple Manor", or from his LINKER program (both available from Quality Software).

The book is published with a plastic comb binding, and is about the same dimensions as the "Apple Assembly Line". There are 156 pages, organized into 8 chapters and 3 appendices. A comprehensive Quick Reference Card for DOS 3.3 is included. There are cartoon sketches throughout which both amuse and aid comprehension, as well as more traditional diagrams and charts and tables. A four page index helps you find whatever you need to know.

Though the book focuses on DOS 3.3, it covers all the major differences found in earlier versions. Chapter 2 is called "The Evolution of DOS", and traces features and differences from Versions 3, 3.1, 3.2, 3.2.1, and 3.3. At other points throughout the book, wherever the various versions differ, the details for each version are explained.

Chapter 3 covers diskette formatting, in much more detail than the Apple DOS manual: how bits are recorded, how 256 bytes are converted to 410 or 342 shorter bytes, how those shorter bytes are converted to encoded bytes ready to be written, how the checksum is computed and tested, how the sectors are identified around a track, all about self-sync bytes, and how sectors are interleaved.

Chapter 4 covers diskette organization: the DOS image, the Volume Table of Contents, the catalog, track/sector lists, and the format of each type of file. Some guidelines for repairing damaged diskettes are given.

Chapter 5 outlines the overall structure of DOS. The booting process is explained in a fair amount of detail. If you need more information on DOS internals, chapter 8 is for you.

Chapter 6 gives clear instructions for using RWTS from machine language programs. You may already be quite familiar with this, because: 1) it is fairly well explained in the DOS manual; 2) many articles have been published in magazines and newsletters telling you how; and 3) you have gone ahead and tried it yourself. But there is another way to get into DOS which treats files as files, but without the normal DOS overhead. Apple's FID utility uses this way in, through the so-called File Manager. Chapter 6 goes into great detail describing the File Manager, and some examples showing how to use it are given. This information has never been published before, and is well worth the price of the entire book. Chapter 6 also shows you how to talk to the disk drive directly, without any DOS at all.

Chapter 7 explains how to customize DOS, and gives the patches for four nice custom features: avoiding the language card reload, making space between DOS and its buffers, removing the pause during a long CATALOG, and changing the HELLO file start-up from RUN to BRUN or EXEC.

Chapter 8, 42 pages long, describes EVERY routine in DOS. It starts with the disk controller ROM (at C600 of your controller is in slot 6), and goes from 9D00 through BFFF subroutine by subroutine. The descriptions are in text form: no disassembled code, and no flowcharts. If you put the book beside a disassembled section of DOS, it is easily understood. Data sections are outlined also, so that you can tell what every byte is there for. The last page of chapter 8 lists all the zero-page variables used by DOS, and explains each use.

Appendix A contains five sample programs which can be used to examine and repair diskettes. They also illustrate the use of RWTS and the File Manager.

Appendix B briefly explains the philosophy of disk protection schemes. Someday someone will write a whole book on this subject. This Appendix is only four pages, so you won't find out how to create the uncrackable disk, or even how to crack it if you did.

Appendix C is an excellent glossary of terms used in the book. I estimate that about 160 words are defined.

The authors list five good reasons why they wrote Beneath Apple DOS; no, six:

1. To show direct assembly language access to DOS.
2. To help you fix clobbered diskettes.
3. To correct errors and ommissions in the Apple manuals.
4. To provide complete infomation on diskette formatting and DOS internal operation.
5. To allow you to customize DOS to fit your needs.
6. To make the authors a lot of money.

They have done an excellent job with the first five objectives, and I think number 6 will be met as well.

```
==============================================================================
DOCUMENT :AAL-8106:DOS3.3:DOS33.B800.BCFF.txt
==============================================================================

 1000  *      .LIF
 1010  *---------------------------------
 1020  *      DOS 3.3 DISASSEMBLY        $B800 - $BCFF
 1030  *      COMMENTS BY BOB SANDER-CEDERLOF  5-25-81
 1040  *---------------------------------
 1050         .OR $B800
 1060         .TA $0800
 1070  *---------------------------------
 1080  BUF.PNTR            .EQ $3E,3F
 1090  CONST.AA            .EQ $3E
 1100  FMT.SECTOR          .EQ $3F
 1110  VOLUME              .EQ $41
 1120  TRACK.CNTR          .EQ $44
 1130  CURRENT.TRACK       .EQ $0478
 1140  *---------------------------------
 1150  *      DISK CONTROLLER ADDRESSES
 1160  *---------------------------------
 1170  PHOFF  .EQ $C080    PHASE-OFF
 1180  PHON   .EQ $C081    PHASE-ON
 1190  MTROFF .EQ $C088    MOTOR OFF
 1200  MTRON  .EQ $C089    MOTOR ON
 1210  DRV0EN .EQ $C08A    DRIVE 0 ENABLE
 1220  DRV1EN .EQ $C08B    DRIVE 1 ENABLE
 1230  Q6L    .EQ $C08C    SET Q6 LOW
 1240  Q6H    .EQ $C08D    SET Q6 HIGH
 1250  Q7L    .EQ $C08E    SET Q7 LOW
 1260  Q7H    .EQ $C08F    SET Q7 HIGH
 1270  *
 1280  *      Q6    Q7     USE OF Q6 AND Q7 LINES
 1290  *     ----  ----    ---------------------
 1300  *     LOW   LOW     READ (DISK TO SHIFT REGISTER)
 1310  *     LOW   HIGH    WRITE (SHIFT REGISTER TO DISK)
 1320  *     HIGH  LOW     SENSE WRITE PROTECT
 1330  *     HIGH  HIGH    LOAD SHIFT REGISTER FROM DATA BUS
 1340  *---------------------------------
 1350  *
 1360  *---------------------------------
 1370  *      CONVERT 256 BYTES TO 342 6-BIT NYBBLES
 1380  *---------------------------------
 1390  PRE.NYBBLE
 1400         LDX #0
 1410         LDY #2
 1420  .1     DEY
 1430         LDA (BUF.PNTR),Y  NEXT REAL BYTE FROM BUFFER
 1440         LSR
 1450         ROL RWTS.BUFFER.2,X
 1460         LSR
 1470         ROL RWTS.BUFFER.2,X
 1480         STA RWTS.BUFFER.1,Y
```

```
1490        INX
1500        CPX #86
1510        BCC .1
1520        LDX #0
1530        TYA
1540        BNE .1
1550        LDX #85      CLEAR TOP BITS OUT OF BUFFER
1560 .2     LDA RWTS.BUFFER.2,X
1570        AND #$3F
1580        STA RWTS.BUFFER.2,X
1590        DEX
1600        BPL .2
1610        RTS
1620  .PG
1630 *---------------------------------
1640 *      WRITE A SECTOR ON THE DISK FROM RWTS.BUFFER
1650 *---------------------------------
1660 WRITE.SECTOR
1670        SEC          SET IN CASE OF ERROR RETURN
1680        STX $27      SAVE SLOT #
1690        STX $0678    HERE, TOO
1700        LDA Q6H,X    Q6 HIGH, Q7 LOW,
1710        LDA Q7L,X    TO READ WRITE PROTECT STATUS
1720        BMI .5       DISK IS WRITE PROTECTED
1730        LDA RWTS.BUFFER.2  FIRST NYBBLE OF DATA
1740        STA $26      SAVE IT
1750        LDA #$FF     SYNC BYTE
1760        STA Q7H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
1770        ORA Q6L,X    Q6L,Q7H: WRITE ON DISK
1780        PHA          TIME DELAYS
1790        PLA
1800        NOP
1810        LDY #4       WRITE FOUR MORE SYNC BYTES
1820 .1     PHA          WASTE TIME
1830        PLA
1840        JSR WRT2     WRITE (A) ON DISK
1850        DEY
1860        BNE .1       UNTIL 4 OF THEM
1870        LDA #$D5     WRITE DATA HEADER
1880        JSR WRT1
1890        LDA #$AA
1900        JSR WRT1
1910        LDA #$AD
1920        JSR WRT1
1930        TYA          A=0
1940        LDY #86      WRITE 86 NYBBLES
1950        BNE .3       ...ALWAYS
1960 .2     LDA RWTS.BUFFER.2,Y  GET CURRENT NYBBLE AND
1970 .3     EOR RWTS.BUFFER.2-1,Y  EOR WITH PREVIOUS NYBBLE
1980        TAX          USE AS OFFSET INTO TABLE
1990        LDA NYBBLE.TABLE,X  MAP 6-BITS TO 8-BITS
2000        LDX $27      GET SLOT AGAIN
2010        STA Q6H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
2020        LDA Q6L,X    Q6L,Q7H: WRITE ON DISK
```

```
2030         DEY
2040         BNE .2       UNTIL ALL BYTES FROM THIS BLOCK DONE
2050         LDA $26      GET FIRST NYBBLE
2060         NOP
2070  .4     EOR RWTS.BUFFER.1,Y  EOR WITH CURRENT NYBBLE
2080         TAX          INDEX INTO TABLE
2090         LDA NYBBLE.TABLE,X  MAP TO 8-BIT VALUE
2100         LDX $0678    SLOT # AGAIN
2110         STA Q6H,X    Q6H,Q7L: (A) TO SHIFT REGISTER
2120         LDA Q6L,X    Q6L,Q7H: WRITE ON DISK
2130         LDA RWTS.BUFFER.1,Y  GET NYBBLE
2140         INY
2150         BNE .4       MORE TO DO
2160         TAX          LAST NYBBLE
2170         LDA NYBBLE.TABLE,X  MAP TO 8 BITS
2180         LDX $27      SLOT # AGAIN
2190         JSR WRT3     WRITE CHECK SUM ON DISK
2200         LDA #$DE     WRITE TRAILER
2210         JSR WRT1
2220         LDA #$AA
2230         JSR WRT1
2240         LDA #$EB
2250         JSR WRT1
2260         LDA #$FF
2270         JSR WRT1
2280         LDA Q7L,X    Q7L
2290  .5     LDA Q6L,X    Q6L
2300         RTS
2310  *---------------------------------
2320  WRT1   CLC          WAIT 2 CYCLES
2330  WRT2   PHA          WAIT 3 CYCLES
2340         PLA          WAIT 4 CYCLES
2350  WRT3   STA Q6H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
2360         ORA Q6L,X    Q6L,Q7H: WRITE ON DISK
2370         RTS
2380   .PG
2390  *---------------------------------
2400  *      CONVERT 342 6-BIT NYBBLES TO 256 BYTES
2410  *      (THEY ARE NOW RIGHT-JUSTIFIED IN RWTS.BUFFER)
2420  *---------------------------------
2430  POST.NYBBLE
2440         LDY #0
2450  .1     LDX #86
2460  .2     DEX
2470         BMI .1
2480         LDA RWTS.BUFFER.1,Y
2490         LSR RWTS.BUFFER.2,X
2500         ROL
2510         LSR RWTS.BUFFER.2,X
2520         ROL
2530         STA (BUF.PNTR),Y
2540         INY
2550         CPY $26      (RWTS PUT 0 IN $26)
2560         BNE .2
```

```
2570        RTS
2580 *--------------------------------
2590 *      READ SECTOR INTO RWTS.BUFFER
2600 *--------------------------------
2610 READ.SECTOR
2620        LDY #32       MUST FIND $D5 WITHIN 32 BYTES
2630 .1     DEY
2640        BEQ ERROR.RETURN
2650 .2     LDA Q6L,X     READ SHIFT REGISTER
2660        BPL .2        WAIT FOR FULL BYTE
2670 .3     EOR #$D5      SEE IF FOUND $D5
2680        BNE .1        NOT YET
2690        NOP           DELAY BEFORE NEXT READ
2700 .4     LDA Q6L,X     READ SHIFT REGISTER
2710        BPL .4        WAIT FOR FULL BYTE
2720        CMP #$AA      SEE IF $AA
2730        BNE .3        NO
2740        LDY #86       BYTE COUNT FOR LATER
2750 .5     LDA Q6L,X     READ SHIFT REGISTER
2760        BPL .5        WAIT FOR FULL BYTE
2770        CMP #$AD      IS IT $AD?
2780        BNE .3        NO
2790 *--------------------------------
2800        LDA #0        BEGIN CHECKSUM
2810 .6     DEY
2820        STY $26
2830 .7     LDY Q6L,X     READ SHIFT REGISTER
2840        BPL .7        WAIT FOR FULL BYTE
2850        EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
2860        LDY $26       BUFFER INDEX
2870        STA RWTS.BUFFER.2,Y
2880        BNE .6
2890 .8     STY $26
2900 .9     LDY Q6L,X     READ SHIFT REGISTER
2910        BPL .9        WAIT FOR FULL BYTE
2920        EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
2930        LDY $26
2940        STA RWTS.BUFFER.1,Y
2950        INY
2960        BNE .8
2970 .10    LDY Q6L,X     READ CHECKSUM
2980        BPL .10
2990        CMP BYTE.TABLE,Y
3000        BNE ERROR.RETURN
3010 .11    LDA Q6L,X     READ TRAILER
3020        BPL .11
3030        CMP #$DE
3040        BNE ERROR.RETURN
3050        NOP
3060 .12    LDA Q6L,X
3070        BPL .12
3080        CMP #$AA
3090        BEQ GOOD.RETURN
3100 ERROR.RETURN
```

```
3110        SEC
3120        RTS
3130   .PG
3140  *--------------------------------
3150  *      READ ADDRESS
3160  *--------------------------------
3170  READ.ADDRESS
3180         LDY #$FC     TRY 772 TIMES (FROM $FCFC TO $10000)
3190         STY $26
3200  .1     INY
3210         BNE .2
3220         INC $26
3230         BEQ ERROR.RETURN
3240  .2     LDA Q6L,X    READ SHIFT REGISTER
3250         BPL .2       WAIT FOR FULL BYTE
3260  .3     CMP #$D5     SEE IF $D5
3270         BNE .1       NO
3280         NOP          DELAY
3290  .4     LDA Q6L,X    READ SHIFT REGISTER
3300         BPL .4       WAIT FOR FULL BYTE
3310         CMP #$AA     SEE IF $AA
3320         BNE .3       NO
3330         LDY #3       READ 3 BYTES LATER
3340  .5     LDA Q6L,X    READ SHIFT REGISTER
3350         BPL .5
3360         CMP #$96     SEE IF $96
3370         BNE .3       NO
3380         LDA #0       START CHECK SUM
3390  .6     STA $27
3400  .7     LDA Q6L,X    READ REGISTER
3410         BPL .7
3420         ROL
3430         STA $26
3440  .8     LDA Q6L,X    READ REGISTER
3450         BPL .8       WAIT FOR FULL BYTE
3460         AND $26      MERGE THE NYBBLES
3470         STA $2C,Y    $2C -- CHECK SUM
3480         EOR $27      $2D -- SECTOR
3490         DEY          $2E -- TRACK
3500         BPL .6       $2F -- VOLUME
3510         TAY          TEST CHECK SUM
3520         BNE ERROR.RETURN
3530  .9     LDA Q6L,X    READ REGISTER
3540         BPL .9       WAIT FOR FULL BYTE
3550         CMP #$DE     TEST FOR VALID TRAILER
3560         BNE ERROR.RETURN
3570         NOP
3580  .10    LDA Q6L,X    READ REGISTER
3590         BPL .10
3600         CMP #$AA
3610         BNE ERROR.RETURN
3620  GOOD.RETURN
3630         CLC
3640         RTS
```

```
3650        .PG
3660 *--------------------------------
3670 *      TRACK SEEK
3680 *--------------------------------
3690 SEEK.TRACK.ABSOLUTE
3700        STX $2B      CURRENT SLOT*16
3710        STA $2A      SAVE TRACK #
3720        CMP CURRENT.TRACK  COMPARE TO CURRENT TRACK
3730        BEQ .9       ALREADY THERE
3740        LDA #0
3750        STA $26      # OF STEPS SO FAR
3760 .1     LDA CURRENT.TRACK  CURRENT TRACK NUMBER
3770        STA $27
3780        SEC
3790        SBC $2A      DESIRED TRACK
3800        BEQ .6       WE HAVE ARRIVED
3810        BCS .2       CURRENT > DESIRED
3820        EOR #$FF     CURRENT < DESIRED
3830        INC CURRENT.TRACK  INCREMENT CURRENT
3840        BCC .3       ...ALWAYS
3850 .2     ADC #$FE     CARRY SET, SO A=A-1
3860        DEC CURRENT.TRACK  DECREMENT CURRENT TRACK
3870 .3     CMP $26      GET MINIMUM OF:
3880        BCC .4          1. # OF TRACKS TO MOVE LESS 1
3890        LDA $26         2. # OF ITERATIONS SO FAR
3900 .4     CMP #12         3. ELEVEN
3910        BCS .5
3920        TAY
3930 .5     SEC          TURN PHASE ON
3940        JSR .7
3950        LDA ONTBL,Y  GET DELAY TIME
3960        JSR DLY100   DELAY 100*A MICROSECONDS
3970        LDA $27      TRACK NUMBER
3980        CLC          TURN PHASE OFF
3990        JSR .8
4000        LDA OFFTBL,Y
4010        JSR DLY100
4020        INC $26      # OF STEPS SO FAR
4030        BNE .1       ...ALWAYS
4040 *--------------------------------
4050 .6     JSR DLY100
4060        CLC          TURN PHASE OFF
4070 .7     LDA CURRENT.TRACK
4080 .8     AND #3       ONLY KEEP LOW-ORDER 2 BITS
4090        ROL          (0000 0XX0)
4100        ORA $2B      (0SSS 0XX0) MERGE SLOT
4110        TAX          USE AS INDEX FOR PHASE-OFF
4120        LDA PHOFF,X  PHASE-OFF
4130        LDX $2B
4140 .9     RTS
4150 *--------------------------------
4160        .HS AAA0A0   FILLER:  NOT USED IN DOS 3.3
4170 *--------------------------------
4180 *      SHORT DELAY SUBROUTINE
```

```
*--------------------------------
DLY100 LDX #17      100*A MICROSECONDS
.1     DEX
       BNE .1
       INC $46
       BNE .2
       INC $47
.2     SEC
       SBC #1
       BNE DLY100
       RTS
*--------------------------------
*      DELAY TIMES FOR STEPPING MOTOR
*--------------------------------
ONTBL  .HS 01302824201E1D1C1C1C1C1C
OFFTBL .HS 702C26221F1E1D1C1C1C1C1C
       .PG
*--------------------------------
*      NYBBLE TABLE
*--------------------------------
NYBBLE.TABLE
       .HS 96979A9B9D9E9FA6A7ABACADAEAF
       .HS B2B3B4B5B6B7B9BABBBCBDBEBFCB
       .HS CDCECFD3D6D7D9DADBDCDDDEDFE5
       .HS E6E7E9EAEBECEDEEEFF2F3F4F5F6
       .HS F7F9FAFBFCFDFEFF
*--------------------------------
*      FILLER:  $BA69 THRU $BA95 NOT USED BY DOS 3.3
*--------------------------------
       .BS 45
*--------------------------------
*      BYTE TABLE
*--------------------------------
BYTE.TABLE .EQ *-$96
       .HS 0001989902039C040506A0A1A2A3
       .HS A4A50708A8A9AA090A0B0C0DB0B1
       .HS 0E0F10111213B81415161718191A
       .HS C0C1C2C3C4C5C6C7C8C9CA1BCC1C
       .HS 1D1ED0D1D21FD4D52021D8222324
       .HS 25262728E0E1E2E3E4292A2BE82C
       .HS 2D2E2F303132F0F1333435363738F8393A3B3C3D3E3F
*--------------------------------
*      342-BYTE BUFFER FOR NYBBLES
*--------------------------------
RWTS.BUFFER.1 .BS 256  $BB00 - BBFF
RWTS.BUFFER.2 .BS 86   $BC00 - BC55
*--------------------------------
       .PG
*--------------------------------
*      WRITE ADDRESS HEADER (CALLED BY FORMAT)
*--------------------------------
WRITE.ADDRESS
       SEC          SET IN CASE OF ERROR RETURN
       LDA Q6H,X    Q6 HIGH, Q7 LOW,
```

```
4730        LDA Q7L,X    TO READ WRITE PROTECT STATUS
4740        BMI .2       DISK IS WRITE PROTECTED
4750        LDA #$FF     SYNC BYTE
4760        STA Q7H,X    Q6H,Q7H: (A) TO SHIFT REGISTER
4770        CMP Q6L,X    Q6L,Q7H: WRITE ON DISK
4780        PHA          TIME DELAYS
4790        PLA
4800 .1     JSR .3       12 CYCLE DELAY
4810        JSR .3       12 CYCLE DELAY
4820        STA Q6H,X    WRITE ON DISK
4830        CMP Q6L,X
4840        NOP
4850        DEY
4860        BNE .1
4870        LDA #$D5     WRITE D5 AA 96
4880        JSR WRITE.BYTE.3
4890        LDA #$AA
4900        JSR WRITE.BYTE.3
4910        LDA #$96
4920        JSR WRITE.BYTE.3
4930        LDA VOLUME   WRITE VOLUME, TRACK, AND SECTOR
4940        JSR WRITE.BYTE.1
4950        LDA TRACK.CNTR
4960        JSR WRITE.BYTE.1
4970        LDA FMT.SECTOR
4980        JSR WRITE.BYTE.1
4990        LDA VOLUME   COMPUTE CHECKSUM
5000        EOR TRACK.CNTR
5010        EOR FMT.SECTOR
5020        PHA          WRITE CHECKSUM
5030        LSR
5040        ORA CONST.AA   #$AA, FOR TIMING
5050        STA Q6H,X
5060        LDA Q6L,X
5070        PLA
5080        ORA #$AA
5090        JSR WRITE.BYTE.2
5100        LDA #$DE     WRITE DE AA EB
5110        JSR WRITE.BYTE.3
5120        LDA #$AA
5130        JSR WRITE.BYTE.3
5140        LDA #$EB
5150        JSR WRITE.BYTE.3
5160        CLC
5170 .2     LDA Q7L,X
5180        LDA Q6L,X
5190 .3     RTS
5200 *---------------------------------
5210 *      SUBROUTINES TO WRITE BYTE ON DISK
5220 *---------------------------------
5230 WRITE.BYTE.1
5240        PHA          ADDRESS BLOCK FORMAT
5250        LSR
5260        ORA CONST.AA
```

```
5270        STA Q6H,X
5280        CMP Q6L,X
5290        PLA
5300        NOP
5310        NOP
5320        NOP
5330        ORA #$AA
5340 WRITE.BYTE.2
5350        NOP
5360 WRITE.BYTE.3
5370        NOP
5380        PHA
5390        PLA
5400        STA Q6H,X
5410        CMP Q6L,X
5420        RTS
5430 *---------------------------------
5440 *      $BCDF THRU $BCFF IS NOT USED BY DOS 3.3
5450 *---------------------------------
5460        .PG
```

```
==================================================================
DOCUMENT :AAL-8106:DOS3.3:S.AMPERTONES.txt
==================================================================
```

```
1000  *---------------------------------
1010  *      DUAL TONE, AND TONE WITH VOLUME CONTROL
1020  *---------------------------------
1030  *      WRITTEN BY MARK KRIEGSMAN.......5-22-81
1040  *      REVISED BY BOB SANDER-CEDERLOF..5-29-81
1050  *---------------------------------
1060         .OR $300
1070         .TF B.AMPERTONES
1080  *---------------------------------
1090  *      ROM SUBROUTINES USED
1100  *---------------------------------
1110  CHKCOM .EQ $DEBE    MUST SEE COMMA
1120  SYNERR .EQ $DEC9    SYNTAX ERROR
1130  GTBYTC .EQ $E6F5    EAT CHAR, GET BYTE IN X
1140  GETNUM .EQ $E746    GET TWO-BYTE VALUE IN $50,51
1150  *                   THEN COMMA AND ONE-BYTE VALUE IN X
1160  *---------------------------------
1170  *      PAGE-ZERO VARIABLES
1180  *---------------------------------
1190  DURATION   .EQ $50 AND $51
1200  TONE1.CNT  .EQ $FB
1210  TONE2.CNT  .EQ $FC
1220  TONE1      .EQ $FD
1230  TONE2      .EQ $FE
1240  VOLUME     .EQ $FF
1250  *---------------------------------
1260  *      I/O ADDRESSES
1270  *---------------------------------
1280  SPKR       .EQ $C030
1290  *---------------------------------
1300  AMPERSAND.VECTOR    .EQ $3F5 THRU $3F7
1310  *---------------------------------
1320  *      INITIALIZE AMPERSAND VECTOR
1330  *---------------------------------
1340  INIT   LDA #$4C     JMP OPCODE
1350         STA AMPERSAND.VECTOR
1360         LDA #AMPERTONE
1370         STA AMPERSAND.VECTOR+1
1380         LDA /AMPERTONE
1390         STA AMPERSAND.VECTOR+2
1400         RTS
1410  *---------------------------------
1420  *      AMPERSAND ENTRY POINT
1430  *---------------------------------
1440  AMPERTONE
1450         CMP #'T      IS IT TONE?
1460         BEQ SINGLE.TONE
1470         CMP #'D      IS IT DUAL?
1480         BEQ DUAL.TONES
```

```
1490        JMP SYNERR   NEITHER, SO SYNTAX ERROR
1500 *--------------------------------
1510 *      &T <TONE>,<DURATION>,<VOLUME>
1520 *--------------------------------
1530 SINGLE.TONE
1540        JSR GET.PARAMS
1550        TXA          LIMIT VOLUME
1560        AND #15      TO 0-15
1570        STA VOLUME
1580        LDA TONE1
1590        STA TONE1.CNT
1600 .1     DEC TONE1.CNT
1610        BNE .5
1620        LDA SPKR     TOGGLE SPEAKER
1630        LDA TONE1    RESET COUNTER
1640        STA TONE1.CNT
1650        LDY VOLUME
1660 .3     NOP
1670        NOP
1680        DEY
1690        BPL .3
1700        LDA SPKR     TOGGLE SPEAKER AGAIN
1710        LDY VOLUME   EQUALIZE VOLUME DELAY
1720 .4     NOP
1730        INY
1740        CPY #16
1750        BCC .4
1760 .5     LDY #10      SHORT ADDITIONAL DELAY
1770 .6     DEY
1780        BNE .6
1790        JSR DECREMENT.DURATION
1800        BCC .1
1810        RTS
1820 *--------------------------------
1830 *      &D <TONE1>,<DURATION>,<TONE2>
1840 *--------------------------------
1850 DUAL.TONES
1860        JSR GET.PARAMS
1870        STX TONE2
1880        LDA TONE1
1890        STA TONE1.CNT
1900        LDA TONE2
1910        STA TONE2.CNT
1920 .1     DEC TONE1.CNT
1930        BEQ .2       TIME TO TOGGLE
1940        LSR VOLUME   TO EQUALIZE TIME
1950        LDA VOLUME   TO EQUALIZE TIME
1960        BPL .3       ...ALWAYS
1970 .2     LDA SPKR     TOGGLE SPEAKER
1980        LDA TONE1    RESET COUNTER
1990        STA TONE1.CNT
2000 .3     DEC TONE2.CNT
2010        BEQ .4
2020        LSR VOLUME   TO EQUALIZE TIME
```

```
2030        LDA VOLUME   TO EQUALIZE TIME
2040        BPL .5       ...ALWAYS
2050 .4     LDA SPKR     TOGGLE SPEAKER
2060        LDA TONE2    RESET COUNTER
2070        STA TONE2.CNT
2080 .5     JSR DECREMENT.DURATION
2090        BCC .1
2100        RTS
2110 *---------------------------------
2120 *      GET THREE PARAMETERS AFTER &T OR &D
2130 *      1. 8-BIT VALUE, STORE IN TONE1
2140 *      2. COMMA
2150 *      3. 16-BIT VALUE, STORE IN DURATION
2160 *      4. COMMA
2170 *      5. 8-BIT VALUE, RETURN IN X-REGISTER
2180 *---------------------------------
2190 GET.PARAMS
2200        JSR GTBYTC   GET TONE
2210        STX TONE1
2220        JSR CHKCOM
2230        JMP GETNUM   GET DURATION AND VOLUME
2240 *---------------------------------
2250 *      DECREMENT DURATION
2260 *      RETURN CARRY CLEAR IF NOT FINISHED
2270 *---------------------------------
2280 DECREMENT.DURATION
2290        LDA DURATION FINISHED YET?
2300        BNE .2
2310        LDA DURATION+1
2320        BNE .1
2330        SEC
2340        RTS          FINISHED
2350 .1     DEC DURATION+1
2360 .2     DEC DURATION
2370        CLC
2380        RTS
```

==================================================================
DOCUMENT :AAL-8106:DOS3.3:S.BASCALC.txt
==================================================================

```
 1000  *--------------------------------
 1010  *      BASCALC FROM APPLE MONITOR
 1020  *--------------------------------
 1030  BASL   .EQ $28
 1040  BASH   .EQ $29
 1050  *--------------------------------
 1060  BASCALC
 1070         PHA          ARG = 000ABCDE
 1080         LSR          (A) = 0000ABCD, E IN CARRY
 1090         AND #3       (A) = 000000CD
 1100         ORA #4       (A) = 000001CD
 1110         STA BASH     HI-BYTE OF ADDRESS
 1120         PLA          (A) = 000ABCDE
 1130         AND #$18     (A) = 000AB000
 1140         BCC .1       MERGE IN E FROM CARRY
 1150         ADC #$7F     (A) = E00AB000
 1160  .1     STA BASL     BASL = E00AB000
 1170         ASL          (A) = 00AB0000, E IN CARRY AGAIN
 1180         ASL          (A) = 0AB00000, CARRY CLEAR
 1190         ORA BASL     (A) = EABAB000
 1200         STA BASL     LO-BYTE OF ADDRESS
 1210         RTS
```

```
=============================================================================
DOCUMENT :AAL-8106:DOS3.3:S.BY.TEN.txt
=============================================================================

 1000  *---------------------------------
 1010  *      MULTIPLY TWO BYTES BY TEN
 1020  *---------------------------------
 1030  B0     .EQ $00
 1040  B1     .EQ $01
 1050  BY.TEN LDA B1       SAVE HI-BYTE ON STACK
 1060         PHA
 1070         LDA B0       GET LO-BYTE IN A
 1080         ASL B0       DOUBLE THE TWO-BYTE VALUE
 1090         ROL B1
 1100         ASL B0       DOUBLE IT AGAIN
 1110         ROL B1
 1120         CLC          ADD IN THE ORIGINAL VALUE
 1130         ADC B0
 1140         STA B0       LO-BYTE
 1150         PLA          HI-BYTE
 1160         ADC B1
 1170         STA B1
 1180         ASL B0       DOUBLE 5*B TO GET 10*B
 1190         ROL B1
 1200         RTS          RETURN TO CALLER
```

```
==============================================================================
DOCUMENT :AAL-8106:DOS3.3:S.MXN.MULTIPLY.txt
==============================================================================

 1000  *---------------------------------
 1010  *      M-BYTE BY N-BYTE MULTIPLY
 1020  *---------------------------------
 1030  M      .EQ $00      # BYTES IN MULTIPLICAND
 1040  N      .EQ $01      # BYTES IN MULTIPLIER
 1050  PSIZE  .EQ $02      # BYTES IN PRODUCT
 1060  I      .EQ $03      LOOP COUNTER
 1070  J      .EQ $04      LOOP COUNTER
 1080  MULTIPLICAND .EQ $90 THRU ...
 1090  MULTIPLIER   .EQ $A0 THRU ...
 1100  PRODUCT      .EQ $B0 THRU ...
 1110  *---------------------------------
 1120  MXN.MPY
 1130  *---------------------------------
 1140  *      CLEAR THE PRODUCT REGISTER
 1150  *---------------------------------
 1160         LDY M        # BYTES IN MULTIPLICAND
 1170         STY PSIZE
 1180         LDA #0
 1190  .1     STA PRODUCT,Y
 1200         DEY
 1210         BPL .1
 1220  *---------------------------------
 1230  *      FOR I=M TO 1 STEP -1
 1240  *      PSIZE = PSIZE + 1
 1250  *      FOR J=8 TO 1 STEP -1
 1260  *---------------------------------
 1270         LDA N        # BYTES IN MULTIPLIER
 1280         STA I
 1290  .2     INC PSIZE
 1300         LDA #8
 1310         STA J
 1320  *---------------------------------
 1330  *      ACCUMULATE PARTIAL PRODUCT FOR NEXT BIT
 1340  *---------------------------------
 1350  .3     JSR SHIFT.MULTIPLIER.RIGHT
 1360         BCC .4       ZERO-BIT
 1370         JSR ACCUMULATE.PARTIAL.PRODUCT
 1380  .4     JSR SHIFT.PRODUCT.RIGHT
 1390  *---------------------------------
 1400  *      NEXT J : NEXT I
 1410  *---------------------------------
 1420         DEC J
 1430         BNE .3
 1440         DEC I
 1450         BNE .2
 1460         RTS
 1470  *---------------------------------
 1480  *      SHIFT MULTIPLIER RIGHT
```

```
1490  *--------------------------------
1500  SHIFT.MULTIPLIER.RIGHT
1510         LDY N        # BYTES IN MULTIPLIER
1520         LDX #0
1530  .1     ROR MULTIPLIER,X
1540         INX
1550         DEY
1560         BNE .1
1570         RTS
1580  *--------------------------------
1590  *      SHIFT PRODUCT RIGHT
1600  *--------------------------------
1610  SHIFT.PRODUCT.RIGHT
1620         LDY PSIZE    # BYTES IN PRODUCT
1630         LDX #0
1640  .1     ROR PRODUCT,X
1650         INX
1660         DEY
1670         BPL .1
1680         RTS
1690  *--------------------------------
1700  *      ACCUMULATE PARTIAL PRODUCT
1710  *--------------------------------
1720  ACCUMULATE.PARTIAL.PRODUCT
1730         LDY M
1740         DEY
1750         CLC
1760  .1     LDA MULTIPLICAND,Y
1770         ADC PRODUCT,Y
1780         STA PRODUCT,Y
1790         DEY
1800         BPL .1
1810         RTS
```

================================================================
DOCUMENT :AAL-8107:Articles:Front.Page.txt
================================================================

Volume 1 -- Issue 10                              July, 1981

In This Issue...



Using Firmware Card in Slot 4

Are you tired of getting "LANGUAGE NOT AVAILABLE" errors?  Do you have a 16K RAM card, and also an old Firmware Card with one of the Basics on it?  You can patch DOS to allow the Firmware Card to be put in slot 4, and still keep your RAM card in slot 0 for Pascal or whatever. With DOS loaded, type CALL -151 to get to the monitor; then patch:

```
     *A5B8:C0
     *A5C0:C1
```

Get back into Basic (3D0G), and INIT a disk with the modified DOS.  If you have a disk utility program, you can patch the DOS image on an existing disk the same way.  (From Michael W. Sanders, Decatur, GA)

==============================================================================
DOCUMENT :AAL-8107:Articles:LowerCaseApple.txt
==============================================================================

The Lower Case Apple.........................Bob Matzinger

It occured to me that, since I have installed a Dan Paymar Lower Case Adapter, there ought to be a better way to generate lower case characters than by RAM-resident software.

The major problem is the F8 ROM. The CAPTST routine at $FD7E will not allow lower case characters to pass; if they get this far, they will be converted to upper case here. I cannot figure a reason for this routine, since the Apple will not generate lower case codes in the first place!

Anyway, there are only two ways I know of to avoid CAPTST: write my own line input subroutine (I want to avoid that!), or burn a new F8 ROM. All I would have to change is one lousy byte, at $FD83, from $DF to $FF. Seems like a waste of time...or is it? Maybe, since I am going to the trouble of burning the ROM, I can add some routines to extend the capabilities of my keyboard to access ALL of the ASCII characters.

That is what I decided to do. But! How do I make it transparent? It should not interfere with or be interfered by any program or language.

Within the monitor routines there are two that are not used; in fact, they were removed when the Autostart ROM came about. These are the 16-bit multiply and divide routines from $FB60 through $FBC0. I can insert my new code there.

I also need two RAM locations for shift lock and case flags. I must find two locations that would probably NOT be used by any other program. There are a number of location in zero page that are not normally used; the bottom of the stack and the top of the input buffer might not be used. Checking that out, however, I have found that most other people have thought of these locations already. Where can I go?

I found two bytes not used by anyone, inside the screen buffer area. They are reserved for the board plugged into slot 6, which in my case is the disk controller. The disk controller does not use locations $077E and $07FE ($0778+slot# and $07F8+slot#). More than likely, nobody would use these locations (at least that is what I am gambling on).

Now that I have room for flags, the next step is to write the routines to fit between $FB60 and $FBC0, and set up calls to them. I have to be careful not to change any other routines. Here is what I want:

1. Upon RESET, initialize to upper case.
2. Have a shift and shift-lock routine.
3. Be able to enter all ASCII characters.

When RESET is pressed, or when the Apple is turned on, the 6502 microprocessor executes a JMP indirect using the address at $FFFC and $FFFD. This effectively jumps to $FF59 in the monitor which is the reset routine. The reset routine calls INIT at $FB2F, which in turn ends with a JMP VTAB at $FB5D. If I change that last instruction, it can fall into the area formerly occupied by the multiply routine. How convenient! I'll put the code there to set upper case mode.

Most programs written for use with the Paymar Adapter have their own input routines. The monitor routines are not used. Therefore my changes should have no adverse effect on these programs.

The next thing I had to decide was which control-keys to use for shift, shift-lock, and the three characters not available from the standard Apple keyboard. I didn't want to use the escape key, since it is used by so many other programs. I finally chose these:

```
control-Z:  Shift and Shift-lock
control-K:  Left bracket and Left Brace
control-L:  Backslash and Vertical Bar
control-O:  Underline and Rubout
```

One final problem to overcome is passing the cursor over a lower case character. The cursor, in the normal monitor, makes the character under the cursor flash. A lower case character will flash in upper case, so you cannot tell whether it was lower or upper case without moving the cursor. I decided to make lower case characters under the cursor display as inverse upper case, rather than flashing. That way there is no doubt.

Now how do we get the patches into the ROM? First we need to get a copy of the standard ROM code into RAM. Then assemble the patches, and save the patched copy on disk. From inside the S-C Assembler II, type:

```
     :$6800<F800.FFFFM             (copy monitor into RAM)
     :ASM                          (assemble the patches)
     :BSAVE F8 EPROM,A$6800,L$800  (save patched monitor)
```

After the patches had been made, I used ROMWRITER, by Mountain Hardware, to burn a 2716 EPROM. This EPROM was then inserted, with appropriate adaptation, in the F8 socket on my Apple mother board.

[NOTE: A 2716 EPROM WILL NOT DIRECTLY REPLACE THE F8 ROM. EITHER THE MOTHER BOARD CIRCUITRY MUST BE MODIFIED OR AN APPROPRIATE SOCKET ADAPTER MUST BE USED.]

If you have a 16K RAM card, you can try the patched monitor without burning a ROM. After the patches have been assembled into the standard copy at $6800, type the following:

```
     :$C081 C081                  (write enable RAM card)
     :$F800<6800.6FFFM            (copy new monitor up)
```

```
     :$C080                      (turn on RAM version)
```

After putting the patched monitor into the RAM card, you have to patch the assembler to turn off its own CAPTST, if you want to see the lower case stuff work inside the assembler.  Type:

```
     :$139B:FF
```

This will make the assembler allow lower case characters to be typed in, but they are only legal in comments.

Some more words of caution.  These patches are for the "old" monitor ROM.  They will not work in the Autostart ROM.  My choice of control-K and control-L may upset some users.  Control-K is used as a monitor command equivalent for IN#slot, and control-L is used to generate a form-feed on some printers.  I can always go to BASIC for the IN#slot, and my printer has a button for form-feed.  I feel that the full upper-lower case ability is much more desirable.

WHEN ALL ELSE FAILS, READ THE INSTRUCTIONS AGAIN!

==============================================================================
DOCUMENT :AAL-8107:Articles:Miscellaneous.txt
==============================================================================

Renewing Subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for.

If your label says "8109", now is the time to renew to be sure of uninterrupted service.

Beneath Apple DOS

In the few weeks since I sent out last month's AAL, with the review of this book, I have sold 85 copies! My apologies if your shipment was delayed a little. Last Friday at 3:30 a shipment of 100 copies arrived; at 5:45 I took about 50 packages to the UPS station. Another 10 went out by mail this morning. A lot of work, but a lot of fun too.

I expect another shipment of 100 copies about the time you get this newsletter, so go ahead and order your copy if you have been waiting.

Restoring Clobbered Page 3 Pointers........Preston R. Black, M.D.

Here's a very short (14 byte) program which you might find useful. As you know, DOS writes the page 3 vectors (between $3D0 and $3FF) as the last step in the bootstrap process. This is done by copying a portion of DOS onto this area. The image remains in memory and can be used to rewrite the vectors if they are clobbered.

If you have a 48K Apple, the routine which copies the vector data starts at $9E25. My program temporarily patches DOS to isolate the vector-copier, by storing an RTS opcode at the end of the loop ($9E30). After calling the loop, the original value of $9E30 is restored.

I put the subroutine at $BCD0 inside DOS, abecause this area is not used by DOS. It can be placed on all slave diskettes you INIT after patching DOS. With this subroutine installed, you can use all of page 3 for your assembly language program. Once your program is finished, you can JMP $BCD0 to restore $3D0-$3FF to its normal state.

Here is the program, written to assemble into $0CD0-0CDD. After assembly is complete, you can move it into DOS with the monitor command

```
     :$BCD0<CD0.CDDM   (if issued from inside S-C Assembler II
     or
     *BCD0<CD0.CDD     (if you do it from the monitor.
```

<program.1>

On second thought, 12 bytes is enough. Rather than patching the DOS code to make a subroutine, I can just put a program up at $BCD0 which looks like the code at $9E25. Here is the shorter version:

<program.2>

=============================================================================
DOCUMENT :AAL-8107:Articles:Screen.Printer.txt
=============================================================================

Screen Printer

Last month I alluded to my trouble in getting a screen printing subroutine to work with the Apple Parallel Interface. I finally got it going, and now it doesn7t look hard at all.

The program is set up to be loaded and started with a BRUN command. This doesn't start any printing, however. The initial code just puts a hook address into location $38 and $39, and passes them to DOS. Henceforth, all character-input calls will have to go through my routine at lines 1260-1320.

The SCRN.PRNT subroutine looks at each input character to see if it is a control-P (ascii code = $90). If not, the character is passed on to whatever program tried to read a character. If it is a control-P, the current contents of the screen are printed.

(My printer is in slot 1; if you are using a different slot, change lines 1110 and 1120.)

The actual printing subroutine is really straightforward. It consists of four parts: 1) save current registers and cursor position; 2) initialize Apple Parallel Interface temporaries; 3) print each line of the screen on the printer; and 4) restore the cursor position and registers.

Lines 1350-1410 save the A-, X-, and Y-registers on the stack, followed by the cursor horizontal position. I pushed them on the stack rather than allocate temporaries, but either way will work. Using the stack saves a few bytes of code and 4 bytes of temporary memory, but it takes a few more cycles if you are worried about speed.

Lines 1420-1490 initialize the temporaries used by the code in Apple's Parallel Interface ROM. These temporaries are actually inside the screen buffer memory (between $0400 and $07FF), but they are in bytes that do not get displayed. (There are 64 bytes in the screen buffer that do not get displayed, and which are used by interface cards for temporary memory. These are $478-47F, $4F8-4FF, $578-57F, $5F8-5FF, $678-67F, $6F8-6FF, $778-77F, and $7F8-7FF.) For more information on how the Parallel Interface uses these temporaries, see your manual.

Lines 1500-1670 actually print the screen contents. The X-register is used as a line counter, and runs from 0 to 23. See lines 1500, 1510, and 1650-1670. This is quite analogous to a BASIC statement like FOR I=0 TO 23.

Inside the X-loop, line 1520 computes a new base address for the current line. Then the Y-register is used as a column counter. Lines 1530 and 1600-1620 control the Y-loop. Inside the Y-loop, each character of the line is picked up in turn. Lines 1550-1580 convert inverse or flashing characters to normal ASCII codes for printing. Line 1590 calls on the Parallel Interface program to print one character. (The entry at $Cx02 assumes all temporaries are already set up.) At the end of each line, lines 1630 and 1640 send a carriage return to the printer.

Lines 1680-1700 restore the cursor position and base address pointer, and lines 1710-1750 restore the 6502 registers.

I wrote this program, lines 1340-1760, as a subroutine even though it could have been in-line. I did it so that you can call it directly from your Applesoft or Integer BASIC program, with a "CALL 793". This feature makes the very-valuable screen printer even more useful.

==============================================================================
DOCUMENT :AAL-8107:Articles:StepTrace.Util.txt
==============================================================================

Step-Trace Utility

The Motive:

"Not that it was that good, mind you!  But we needed something, and they should not have yanked it out without providing some other way to debug machine language programs."

When Apple converted over to the Autostart ROM, they not only removed the hardly-ever-used 16-bit multiply and divide subroutines.  They also stripped the S and T commands, which left assembly language programmers naked.  How can you possibly debug complicated 6502 code without at least a single step capability?

Several programs are now on the market, in the $50 price range, which give you step, trace, breakpoints, stack display, et cetera.  "John's Debugger", from John Broderick & Associates, 8635 Shagrock, Dallas, TX 75238 is one.  Someone called me from Augusta, GA, yesterday to tell me about a similar package he has written and wants to market (I'll be reviewing this one; it may become an S-C SOFTWARE product).  I saw another ad this month somewhere, but I cannot find it now.

But I wanted to do something special this month for the Assembly Line, so here is a limited STEP-TRACE program...free!

The Manner:

It is set up as a BRUNnable file, to load at $0800.  If you want to load it somewhere else, you can put in an origin directive (.OR).  The code executed when you BRUN the file (lines 1390-1460) merely installs the "control-Y vector".  This enables the control-Y monitor command, which is a user-definable command.

Once the control-Y vector is loaded, you have two new commands.  If you type a memory address and a control-Y (and a carriage return), the instruction at that memory address will be disassembled and displayed on line 23.  The flashing cursor will be positioned at the end of the disassembled instruction.  Just above the cursor, on line 22, you will see the current register contents.  Line 24 is an inverse mode line which labels the registers, and reminds you of the options you have.

At this point you can type one of the five register names (A, X, Y, S, or P), or a space, or a carriage return.  If you type a carriage return, the trace is aborted and you are returned to the assembler. If you type a space, the disassembled instruction will be exectuted. The new register contents will be displayed, the screen will scroll up, and the next instruction will be disassembled on line 23.  If you type a register name, the cursor will be moved under that register.

You can type in a new value for the register, and then hit a space for the next register or a return to get ready to execute again.

If you want to step through a little faster, hold down the space bar and the repeat key.

Once you have terminated the trace (by typing a carriage return), you can restart where you stopped by typing a control-Y and a carriage return. Since there is no address given, STEP-TRACE will begin where you stopped the last time. You can stop the trace, do some monitor commands, and then start tracing again.

Two warnings: I wrote STEP-TRACE to be used from inside the S-C ASSEMBLER II. That means all monitor commands, including the control-Y, need to be preceded by a dollar sign ($). If you want to use STEP-TRACE directly from the monitor, and not return inside the assembler after stopping, you need to change line 3500. It now says JMP $3D0, which restarts DOS and the assembler. Change it to JMP $FF69, which restarts the monitor. Line 3470 requires the .DA modification published in the December 1980 issue of AAL. If you haven't installed that yet, then rewrite line 3470 as five separate lines; if you don't, it will assemble without error but it will be WRONG!

The Method:

Now let's look through the listing, and see how it works. When the monitor decodes the control-Y command, the address you typed (if any) is loaded into $3C,3D in page zero. Then the monitor branches to $3F8, where we have already loaded a JMP STEP.TRACE instruction. We step into the action at line 1510.

Lines 1520-1570: the X-register is zero if no address was typed. In this case, we skip around the code to copy the address into MON.PC. If there was an address, copy it into MON.PC.

Lines 1580-1630: Set the stack pointer to $FF, giving the whole stack to the program under test. Move the cursor to the bottom of the screen and print a carriage return.

Lines 1650-1680: Call on subroutines to display the current register values (from the SAVE.AREA at line 4350-4400), disassemble the instruction pointed to by MON.PC, and wait on you to type something on the keyboard. This last subroutine does not return unless you type a space, indicating you want to execute the disassembled instruction.

Lines 1690-1860: Clear the XQT.AREA to NOP instructions. Get the stack pointer from the SAVE.AREA. Pick up the opcode byte, and see if it is one we have to interpret rather than execute (BRK, JSR, RTI, JMP, RTS, or JMP indirect). If so, jump to the appropriate code for each opcode.

Lines 1870-2010: Get the instruction length (less one) in Y, so we can copy the instruction into XQT.AREA. See if the opcode is one of the relative branches; if so, change the displacement to $04, so that we can execute it inside XQT.AREA.  Copy the instruction bytes into XQT.AREA.  Restore the registers from the SAVE.AREA, restoring status (P-register last of all.

Lines 2030-2160:  Execute the instruction.  Unless it is a relative branch instruction which branches, jump to did.not.branch.  Relative branches which branch go to line 2100, where the effective address is computed and stored in MON.PC.

Lines 2180-2190:  A BRK instruction displays the registers and returns to the assembler (aborts STEP-TRACE).

Lines 2210-2250:  The RTI instruction checks the stack pointer; if there are not three bytes left on the stack, STEP-TRACE is aborted. If there are three left, the next byte is pulled off the stack and stored in the SAVE.AREA for the P-register.  The rest of the RTI instruction is the same as an RTS istruction.

Lines 2260-2350:  The RTS instruction checks the stack pointer; if there are not two bytes left on the stacke, STEP-TRACE is aborted.  If there are two left, they are pulled off and stored in MON.PC.

Lines 2370-2470:  The JSR instruction picks up the current MON.PC, adds two, and pushes the result on the stack.  The new stack ponter value is saved in SAVE.AREA.  Then a JMP instruction is simulated.

Lines 2480-2490:  Simulate a JMP instruction by copying the address into MON.PC.

Lines 2500-2530:  Simulate a JMP indirect instruction.  Copy the address contained in the two bytes pointed to by the instruction address into MON.PC.

Lines 2550-2640:  After a normal executed instruction, save all the registers in SAVE.AREA.  Be sure the processor is in binary mode (not decimal).

Lines 2650-2690:  Add the instruction length to MON.PC, and go back to get the next instruction.

Lines 2710-2800:  Using the current MON.PC as a pointer, pick up the two bytes pointed to and put them into MON.PC.  This is used by the JSR, JMP, and JMP indirect processors.

Lines 2820-2930:  Set cursor position to line 23, column 27, and wait for you to type a key.  If you type a carriage return, abort STEP-TRACE.  If you type a space, return to whoever called WAIT.ON.KEYBOARD.

Lines 2940-2990:  See if you typed a register name (letter A, X, Y, S, or P).  If not, go back and wait till you type something else.  If so, go on to line 3000.

Lines 3000-3100: Set inverse mode, position the cursor to the selected register column, and display the current contents of that register in inverse mode. Switch back to normal mode.

Lines 3110-3340: Wait again for you type a character on the keyboard. If you type a hexadecimal digit, shift the current register contents one digit position to the left, and add in the digit you just typed. (You can type as many digits as you want to; the last two you type will be the new contents.) If you type a space or a carriage return, branch to line 3350 or 3400.

Lines 3350-3390: You typed a space, so move over to the next register. If you just modified the S-register, move back to the A-register.

Lines 3400-3440: You typed a carriage return, so scroll up the screen and go back to the top of WAIT.ON.KEYBOARD.

Lines 3450-3470: REG.NAMES defines the register names. REG.INDEX is an index into REG.NAMES and REG.CH. REG.CH is a list of column positions for each of the registers. (If you have not installed the .DA modification from AAL Volume 1, Issue 3, you need to spread the data values out on five separate lines.)

Lines 3490-3500: Clear from the cursor to the end of screen, and return through DOS to the assembler. Change line 3500 if you want to go somewhere else after leaving the STEP-TRACE.

Lines 3540-3590: Adds the contents of the A-register to MON.PC.

Lines 3630-3740: Displays the register contents from SAVE.AREA.

Lines 3810-3840: Prints MON.PC and a dash. This is called by the disassembly subroutine.

Lines 3880-4330: Disassembles the instruction starting at MON.PC. This code is very similar to code in the Apple monitor ROM at $F882. It is modified slightly to change the spacing, so that there will be room for the register display on the same line.

Lines 4440-4480: A test program for you to try STEPping through. Another neat program to trace is at $FCA8 in the monitor (a delay loop).

==========================================================================
DOCUMENT :AAL-8107:Articles:Var.XRef.Correx.txt
==========================================================================

Corrections to Variable Cross Reference Program

The Variable Cross Reference program I printed in issue #2 (November, 1980) had at least three bugs. One of them was reported a long time ago, but I had no idea what the cause was until today. The other two were never reported by anyone, but I discovered their presence and cause today. Eventful day!

Bug #1: After using the VCR program, the first line number LISTed by a subsequent LIST command printed out with all sorts of extra fractional digits. Strange! I finally tracked it down to a page zero location which VCR used. Location $A4 is left with a non-zero value, but Applesoft expects and requires it to be zero. If it is not zero, the floating point multiply subroutine gives wrong answers. The multiplication failure ruins the first number printed after running VCR.

Solution to Bug #1: Add the following two lines to the VCR program.

```
1452          LDA #0       CLEAR $A4 FOR APPLESOFT
1454          STA $A4
```

Bug #2: The logic for terminating the main program loop (lines 1400-1460) was wrong, and resulted in sometimes adding a phony variable.

Solution to Bug #2: Delete line 1810, and change or add the following lines.

```
1650          LDY #3       CAPTURE POINTER AND LINE #
1692          LDA DATA+1   TEST FOR END
1694          BEQ .3       YES
1820 .3       RTS
```

Bug #3: If your program contained a PRINT statement with a quoted string not separated from a variable by a semi-colon or comma, the GET.NEXT.VARIABLE subroutine would invent new variable names from inside the quoted string! For example, the line PRINT D$"OPEN FILE" would add variables OP (for OPEN) and FI (for FILE).

Solution to Bug #3: Change or add the following lines.

```
2752          BEQ .6       YES
2754          CMP #'"      QUOTATION MARK?
2762          LDA PNTR     BACK UP PNTR OVER QUOTE MARK
2763          BNE .7
2764          DEC PNTR+1
2765 .7       DEC PNTR
2766          RTS
2770 .6       LDA VARNAM+2 SET HIGH BIT
```

If you have typed in the VCR program, or bought the Quarterly Disk #1 which contained the source, you should now go back and fix these three bugs.  (All the line numbers above fit in with the program as printed last November.)  Copies of the Quarterly Disk #1 with a serial number of 44 or higher already have been fixed.

```
==================================================================
DOCUMENT :AAL-8107:DOS3.3:S.F8EpromLC.txt
==================================================================

 1000  * LOWER CASE F8 ROM.1
 1010  *---------------------------------
 1020  * THESE PATCHES ARE FOR THE "OLD" F8 ROM.  THEY
 1030  * WILL NOT WORK INTO THE AUTOSTART ROM MONITOR
 1040  * ROUTINES.
 1050  *
 1060  * OPERATION: $6800<F800.FFFFM
 1070  *            ASM  (ASSEMBLE THIS CODE)
 1080  *            BSAVE F8 EPROM,A$6800,L$0800
 1090  *---------------------------------
 1100  CTRLK  .EQ $8B      LEFT BRACKET OR BRACE
 1110  CTRLL  .EQ $8C      BACKSLASH OR VERTICAL BAR
 1120  CTRLO  .EQ $8F      UNDERLINE OR RUBOUT
 1130  CTRLZ  .EQ $9A      SHIFT OR SHIFT LOCK
 1140  CASE   .EQ $77E     FOR DOS IN SLOT 6
 1150  LCKFLG .EQ $7FE     FOR DOS IN SLOT 6
 1160  KYSTRB .EQ $C010
 1170  VTAB   .EQ $FC22
 1180  RDKEY  .EQ $FD0C
 1190  *---------------------------------
 1200  PATCH1 .OR $FB5D
 1210         .TA $6B5D
 1220  *
 1230  SETCAS LDY #0       PART OF RESET ROUTINE TO INIT
 1240         STY CASE     UPPER CASE MODE
 1250         INY
 1260         STY LCKFLG
 1270         JMP VTAB
 1280  *---------------------------------
 1290  PATCH2 .OR $FD2B
 1300         .TA $6D2B
 1310  *
 1320         JMP LCADAP   FROM KEYIN ROUTINE TO LOWER
 1330         NOP          CASE "ADAPTER"
 1340  *---------------------------------
 1350  PATCH3 .OR $FD82
 1360         .TA $6D82
 1370  *
 1380         AND #$FF     ALLOW LOWER CASE TO PASS
 1390  *---------------------------------
 1400  PATCH4 .OR $FD11
 1410         .TA $6D11
 1420  *
 1430         JSR FORM     DISPLAY CHARACTERS UNDER THE
 1440         NOP          CURSOR CORRECTLY
 1450  *---------------------------------
 1460  * THE CTRL-Z KEY IS USED LIKE THE SHIFT KEY ON A
 1470  * TYPEWRITER:  ONE CTRL-Z WILL ENTER ONE UPPER
 1480  * CASE CHARACTER AND THEN RETURN TO LOWER CASE.
```

```
*
* TWO CTRL-Z'S IN SUCCESSION WILL PERFORM A
* "SHIFT-LOCK".  IF THE MODE WAS LOWER CASE,
* TWO CTRL-Z'S WILL LOCK IN UPPER CASE; IF THE
* MODE WAS UPPER CASE, TWO CTRL-Z'S WILL LOCK
* IN LOWER CASE.
*---------------------------------
PATCH5 .OR $FB69
       .TA $6B69
*
LCADAP BIT KYSTRB   CLEAR KEYBOARD
       CMP #CTRLZ   SEE IF "SHIFT"
       BNE .4       NO, TRY OTHER TESTS
       LDA LCKFLG
       EOR #$80     FLIP BIT 7 (CTRLZ FLAG)
       BMI .1       NEGATIVE IF FIRST CTRL-Z
       EOR #$01     FLIP BIT 0 (LOCK FLAG)
.1     STA LCKFLG
       BEQ .2       ...IF LOCK FLAG IS CLEAR
       LDA #0       SET UPPER CASE
       BEQ .3       ...ALWAYS
.2     LDA #$20     SET LOWER CASE
.3     STA CASE
       JMP RDKEY
.4     CMP #CTRLK
       BEQ .5
       CMP #CTRLL
       BEQ .5
       CMP #CTRLO
       BNE .6
.5     ORA #$50     CONVERT TO SPECIAL CHARS
.6     CMP #$C0     MERGE CASE IF ALPHA
       BCC .7       NOT ALPHA
       ORA CASE
.7     PHA          SAVE MODIFIED CHAR
       LDA LCKFLG
       BPL .8       ...IF Z-FLAG CLEAR
       LDA #0       CLEAR Z AND LOCK FLAGS
       STA LCKFLG
.8     BNE .9       ...IF LOCK FLAG IS SET
       LDA #$20     SET LOWER CASE
       STA CASE
.9     PLA          RETRIEVE MODIFIED CHAR
       RTS
       BRK
       BRK
*---------------------------------
* CURSOR DISPLAY FOR EDITING
*
FORM   CMP #$E0     IS IT LOWER CASE?
       BCS .1       YES, SO BRANCH
       AND #$3F     ALL CHARACTERS (EXCEPT LOWER
       ORA #$40     CASE) ARE FLASHED
       RTS
```

```
2030  .1     EOR #$E0     MAKE LOWER CASE INTO
2040         RTS          INVERSE UPPER CASE
2050  *---------------------------------
2055  * WRITTEN:  NOVEMBER 1, 1980
2060  * REVISED:  JUNE 25, 1981
2070  *  AUTHOR:  BOB MATZINGER
2080  *           P. O. BOX 13446
2090  *           ARLINGTON, TX 76013
2100  *           (817) 265-8122
2110  *---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8107:DOS3.3:S.RESTORE.1.txt
==============================================================================

 1000  *---------------------------------
 1010  *       RESTORE PAGE 3 VECTORS
 1020  *       ----------------------
 1030  *
 1040  *       PRESTON R. BLACK, M.D.
 1050  *       12 JUNE 1981
 1060  *---------------------------------
 1070         .OR $BCD0
 1080         .TA $0CD0
 1090  *---------------------------------
 1100  RESTORE.PAGE.3.VECTORS
 1110         LDA #$60     RTS OPCODE
 1120         STA $9E30
 1130         JSR $9E25
 1140         LDA #$AD     ORIGINAL DATA
 1150         STA $9E30
 1160         RTS
```

```
===========================================================================
DOCUMENT :AAL-8107:DOS3.3:S.RESTORE.2.txt
===========================================================================
```

```
 1000  *---------------------------------
 1010  *      RESTORE PAGE 3 VECTORS
 1020  *      ----------------------
 1030  *
 1040  *      PRESTON R. BLACK, M.D.
 1050  *      29 JUNE 1981
 1060  *---------------------------------
 1070         .OR $BCD0
 1080         .TA $0CD0
 1090  *---------------------------------
 1100  RESTORE.PAGE.3.VECTORS
 1110         LDX #$3FF-$3D0 # BYTES TO BE COPIED
 1120  .1     LDA $9E51,X    ADDRESS OF VECTORS INSIDE DOS
 1130         STA $3D0,X     VECTOR AREA
 1140         DEX
 1150         BPL .1
 1160         RTS
```

```
==============================================================================
DOCUMENT :AAL-8107:DOS3.3:S.ScrnPrinter.txt
==============================================================================

 1000  *---------------------------------
 1010  *      SCREEN PRINTER
 1020  *---------------------------------
 1030  MON.CH     .EQ $24
 1040  MON.BASL   .EQ $28,29
 1050  MON.BASCAL .EQ $FBC1
 1060  MON.VTAB   .EQ $FC22
 1070  MON.RDKEY  .EQ $FD0C
 1080  MON.KEYIN  .EQ $FD1B
 1090  DOS.REHOOK .EQ $3EA
 1100  *---------------------------------
 1110  SLOT   .EQ 1
 1120  PRINT  .EQ $C102    $C002+SLOT*256
 1130  MSTRT  .EQ $5F8+SLOT
 1140  MODE   .EQ $678+SLOT
 1150  ESCHAR .EQ $6F8+SLOT
 1160  FLAGS  .EQ $778+SLOT
 1170  *---------------------------------
 1180         .OR $300
 1190  *---------------------------------
 1200         LDA #SCRN.PRNT
 1210         STA $38
 1220         LDA /SCRN.PRNT
 1230         STA $39
 1240         JMP DOS.REHOOK
 1250  *---------------------------------
 1260  SCRN.PRNT
 1270         JSR MON.KEYIN  GET CHAR
 1280         CMP #$90     CONTROL-P?
 1290         BNE .1
 1300         JSR SCREEN.PRINTER
 1310         JMP MON.RDKEY
 1320  .1     RTS
 1330  *---------------------------------
 1340  SCREEN.PRINTER
 1350         PHA          SAVE REGS
 1360         TXA
 1370         PHA
 1380         TYA
 1390         PHA
 1400         LDA MON.CH   SAVE CH
 1410         PHA
 1420         LDA #40      SET UP APPLE CONTROLLER ROM
 1430         STA MSTRT    TEMPORARIES
 1440         LDA #0
 1450         STA MODE
 1460         LDA #$89
 1470         STA ESCHAR
 1480         LDA #1
```

```
1490          STA FLAGS
1500          LDX #0       START AT LINE 0
1510  .1      TXA
1520          JSR MON.BASCAL  COMPUTE BASE POINTER FOR LINE
1530          LDY #0       START AT CHAR 0
1540  .2      LDA (MON.BASL),Y
1550  .3      CMP #$A0     MAP FLASH AND INVERSE TO NORMAL
1560          BCS .4
1570          ADC #$40
1580          BNE .3       ...ALWAYS
1590  .4      JSR PRINT
1600          INY          NEXT CHARACTER
1610          CPY #40      END OF LINE?
1620          BCC .2       NO
1630          LDA #$8D     YES, PRINT CARRIAGE RETURN
1640          JSR PRINT
1650          INX          NEXT LINE
1660          CPX #24      END OF SCREEN
1670          BCC .1       NO
1680          PLA          YES, RESTORE CH
1690          STA MON.CH
1700          JSR MON.VTAB   RESTORE BASE POINTER
1710          PLA          RESTORE REGS
1720          TAY
1730          PLA
1740          TAX
1750          PLA
1760          RTS
```

```
==============================================================================
DOCUMENT :AAL-8107:DOS3.3:S.STEP.TRACE.txt
==============================================================================

 1000  *---------------------------------
 1010  *       STEP-TRACE UTILITY
 1020  *---------------------------------
 1030  MON.WNDBTM .EQ $23
 1040  MON.CH     .EQ $24
 1050  MON.CV     .EQ $25
 1060  LMNEM      .EQ $2C
 1070  RMNEM      .EQ $2D
 1080  MON.FORMAT .EQ $2E
 1090  MON.LENGTH .EQ $2F
 1100  MON.PC     .EQ $3A,3B
 1110  MON.A1     .EQ $3C,3D
 1120  MON.A2     .EQ $3E,3F
 1130  *---------------------------------
 1140  DOS.REENTRY .EQ $3D0
 1150  Y.VECTOR    .EQ $3F8
 1160  BASE.LINE24 .EQ $7D0
 1170  MON.INSDS2  .EQ $F88E
 1180  MON.INSTDSP .EQ $F8D0
 1190  MON.PRADDR  .EQ $F90C
 1200  MON.PRBLNK  .EQ $F948
 1210  MON.PRBL2   .EQ $F94A
 1220  MNEML       .EQ $F9C0
 1230  MNEMH       .EQ $FA00
 1240  MON.VTAB    .EQ $FC22
 1250  MON.CLREOP  .EQ $FC42
 1260  MON.SCROLL  .EQ $FC70
 1270  MON.CLREOL  .EQ $FC9C
 1280  MON.RDKEY   .EQ $FD0C
 1290  MON.CROUT   .EQ $FD8E
 1300  MON.PRYX3   .EQ $FD99
 1310  MON.PRBYTE  .EQ $FDDA
 1320  MON.COUT    .EQ $FDED
 1330  MON.SETINV  .EQ $FE80
 1340  MON.SETNORM .EQ $FE84
 1350  *---------------------------------
 1360  KEYBOARD   .EQ $C000
 1370  STROBE     .EQ $C010
 1380  *---------------------------------
 1390  STEP.TRACE.SETUP
 1400         LDA #$4C      'JMP' OPCODE
 1410         STA Y.VECTOR
 1420         LDA #STEP.TRACE
 1430         STA Y.VECTOR+1
 1440         LDA /STEP.TRACE
 1450         STA Y.VECTOR+2
 1451         LDA #0       CLEAR USER STATUS REGISTER
 1452         STA SAVE.P
 1460         RTS
```

```
1470  *--------------------------------
1480  *      (Y)          SINGLE STEP AT CURRENT PC
1490  *      ADR(Y)       SINGLE STEP AT ADR
1500  *--------------------------------
1510  STEP.TRACE
1520         TXA          X=0 IF NO ADDRESSES
1530         BEQ .1       NO ADDRESSES
1540         LDA MON.A1   ONE OR TWO ADDRESSES
1550         STA MON.PC
1560         LDA MON.A1+1
1570         STA MON.PC+1
1580  .1     LDX #$FF     USER GETS WHOLE STACK
1590         TXS
1600         STX SAVE.S
1610         LDA #23
1620         STA MON.CV
1630         JSR MON.CROUT
1640  *--------------------------------
1650  TRACE.LOOP
1660         JSR DISPLAY.REGISTERS
1670         JSR DISASSEMBLE ONE INSTRUCTION
1680         JSR WAIT.ON.KEYBOARD
1690         LDA #$EA     'NOP' OPCODE
1700         STA XQT.AREA+1
1710         STA XQT.AREA+2
1720         LDX SAVE.S
1730         TXS
1740         LDY #0
1750         LDA (MON.PC),Y  GET USER OPCODE
1760         BEQ X.BRK    'BRK' OPCODE
1770         CMP #$20     'JSR' OPCODE
1780         BEQ X.JSR
1790         CMP #$40     'RTI' OPCODE
1800         BEQ X.RTI
1810         CMP #$4C     'JMP' OPCODE
1820         BEQ X.JMP
1830         CMP #$60     'RTS' OPCODE
1840         BEQ X.RTS
1850         CMP #$6C     'JMP ()' OPCODE
1860         BEQ X.JMPI
1870         LDY MON.LENGTH  # BYTES IN INSTRUCTION
1880         AND #$1F     IF RELATIVE BRANCH, CHANGE
1890         EOR #$14     DISPLACEMENT TO $04
1900         CMP #$04     FOR XQT AREA
1910         BEQ .2
1920  .1     LDA (MON.PC),Y  COPY INSTRUCTION INTO XQT AREA
1930  .2     STA XQT.AREA,Y
1940         DEY
1950         BPL .1
1960         LDA SAVE.P   RESTORE ALL REGISTERS
1970         PHA
1980         LDA SAVE.A
1990         LDX SAVE.X
2000         LDY SAVE.Y
```

```
2010        PLP
2020 *--------------------------------
2030 XQT.AREA
2040        NOP         USER'S OPCODE GOES HERE
2050        NOP
2060        NOP
2070        JMP DID.NOT.BRANCH
2080 *--------------------------------
2090 *   RELATIVE BRANCHES THAT DO BRANCH COME HERE
2091        CLD
2100        CLC
2110        LDY #1      GET ORIGINAL DISPLACEMENT
2120        LDA (MON.PC),Y
2130        BPL .1      POSITIVE DISPLACEMENT
2140        DEC MON.PC+1 DECREMENT HI-BYTE IF NEGATIVE
2150 .1     JSR ADD.A.TO.PC
2160        JMP UPDATE.PC
2170 *--------------------------------
2180 X.BRK  JSR DISPLAY.REGISTERS
2190 RTRN.JMP  JMP RETURN
2200 *--------------------------------
2210 X.RTI  TSX
2220        CPX #$FD
2230        BCS RTRN.JMP
2240        PLA         SIMULATE RTI BY GETTING
2250        STA SAVE.P  STATUS FROM STACK
2260 X.RTS  TSX
2270        CPX #$FE
2280        BCS RTRN.JMP
2290        PLA         SIMULATE RTS BY GETTING
2300        STA MON.PC  PC FROM STACK
2310        PLA
2320        STA MON.PC+1
2330        TSX
2340        STX SAVE.S
2350        JMP UPDATE.PC
2360 *--------------------------------
2370 X.JSR  CLC         UPDATE PC AND PUSH ON STACK
2380        LDA MON.PC
2390        ADC #2
2400        TAY         SAVE LO-BYTE FOR NOW
2410        LDA MON.PC+1
2420        ADC #0
2430        PHA         PUSH HI-BYTE
2440        TYA
2450        PHA         PUSH LO-BYTE
2460        TSX
2470        STX SAVE.S
2480 X.JMP  JSR GET.NEW.PC
2490        JMP TRACE.LOOP
2500 X.JMPI JSR GET.NEW.PC
2510        LDY #0
2520        JSR GET.NEW.PC.0
2530        JMP TRACE.LOOP
```

```
2540 *--------------------------------
2550 DID.NOT.BRANCH
2560        STA SAVE.A   SAVE ALL REGISTERS
2570        STX SAVE.X
2580        STY SAVE.Y
2590        PHP
2600        PLA
2610        STA SAVE.P
2620        TSX
2630        STX SAVE.S
2640        CLD
2650 UPDATE.PC
2660        SEC          0=1, 1=2, 2=3
2670        LDA MON.LENGTH
2680        JSR ADD.A.TO.PC
2690        JMP TRACE.LOOP
2700 *--------------------------------
2710 GET.NEW.PC
2720        LDY #1       GET NEW PC FROM INSTRUCTION
2730 GET.NEW.PC.0
2740        LDA (MON.PC),Y
2750        TAX          SAVE LO-BYTE FOR NOW
2760        INY
2770        LDA (MON.PC),Y
2780        STA MON.PC+1 NEW HI-BYTE
2790        STX MON.PC   NEW LO-BYTE
2800        RTS
2810 *--------------------------------
2820 WAIT.ON.KEYBOARD
2830        LDA #22      LINE 23
2840        STA MON.CV
2850        LDA #26      COLUMN 27
2860        STA MON.CH
2870        JSR MON.VTAB
2880        JSR MON.RDKEY
2890        CMP #$8D
2900        BEQ RETURN
2910        CMP #$A0
2920        BNE .1       REGISTER NAME
2930        RTS
2940 .1     LDY #4
2950 .2     CMP REG.NAMES,Y
2960        BEQ .3
2970        DEY
2980        BPL .2
2990        BMI WAIT.ON.KEYBOARD
3000 .3     STY REG.INDEX
3010 .4     JSR MON.SETINV
3020        LDA #22
3030        STA MON.CV
3040        JSR MON.VTAB
3050        LDY REG.INDEX
3060        LDA REG.CH,Y
3070        STA MON.CH
```

```
3080        LDA SAVE.AREA,Y
3090        JSR MON.PRBYTE
3100        JSR MON.SETNORM
3110 .5     LDA KEYBOARD
3120        BPL .5
3130        STA STROBE
3140        CMP #$A0     BLANK?
3150        BEQ .8       YES
3160        CMP #$8D     RETURN?
3170        BEQ .9       YES
3180        EOR #$B0
3190        CMP #10
3200        BCC .6       DIGIT
3210        ADC #$88
3220        CMP #$FA
3230        BCC .5       NOT DIGIT, SO IGNORE
3240 .6     LDY #3
3250        ASL
3260        ASL
3270        ASL
3280        ASL
3290        LDX REG.INDEX
3300 .7     ASL
3310        ROL SAVE.AREA,X
3320        DEY
3330        BPL .7
3340        BMI .4       ...ALWAYS
3350 .8     LDY REG.INDEX
3360        DEY
3370        BPL .3
3380        LDY #4
3390        BNE .3       ...ALWAYS
3400 .9     LDA #23
3410        STA MON.WNDBTM
3420        JSR MON.SCROLL
3430        INC MON.WNDBTM
3440        JMP WAIT.ON.KEYBOARD
3450 REG.NAMES .AS -/SPYXA/
3460 REG.INDEX .BS 1
3470 REG.CH    .DA #38,#35,#32,#29,#26
3480 *--------------------------------
3490 RETURN JSR MON.CLREOP
3500        JMP DOS.REENTRY
3510 *--------------------------------
3520 *      ADD (A) TO MON.PC
3530 *--------------------------------
3540 ADD.A.TO.PC
3550        ADC MON.PC
3560        STA MON.PC
3570        BCC .1
3580        INC MON.PC+1
3590 .1     RTS
3600 *--------------------------------
3610 *      DISPLAY REGISTERS
```

```
3620  *--------------------------------
3630  DISPLAY.REGISTERS
3640         LDA #26
3650         STA MON.CH
3660         LDX #4
3670         BNE .2
3680  .1     LDA #$A0
3690         JSR MON.COUT
3700  .2     LDA SAVE.AREA,X
3710         JSR MON.PRBYTE
3720         DEX
3730         BPL .1
3740         RTS
3750  *--------------------------------
3760  BOTTOM.LINE .AS / <SPC>=NEXT  <RET>=QUIT   A  X  Y  P  S /
3770         .HS 00
3780  *--------------------------------
3790  *      PRINT PC AND DASH
3800  *--------------------------------
3810  PRINT.PC
3820         LDX MON.PC
3830         LDY MON.PC+1
3840         JMP MON.PRYX3
3850  *--------------------------------
3860  *      DISASSEMBLE NEXT OPCODE
3870  *--------------------------------
3880  DISASSEMBLE
3890         JSR PRINT.PC
3900         LDY #0
3910         LDA (MON.PC),Y GET OPCODE
3920         JSR MON.INSDS2
3930         PHA          SAVE MNEMONIC TABLE INDEX
3940  .1     LDA (MON.PC),Y
3950         JSR MON.PRBYTE
3960         LDX #1       PRINT ONE BLANK
3970  .2     JSR MON.PRBL2
3980         CPY MON.LENGTH
3990         INY
4000         BCC .1
4010         LDX #3
4020         CPY #3
4030         BCC .2
4040         PLA          GET MNEMONIC TABLE INDEX
4050         TAY
4060         LDA MNEML,Y
4070         STA LMNEM
4080         LDA MNEMH,Y
4090         STA RMNEM
4100  .3     LDA #0
4110         LDY #5
4120  .4     ASL RMNEM    SHIFT 5 BITS OF CHARACTER INTO A
4130         ROL LMNEM
4140         ROL
4150         DEY
```

```
4160        BNE .4
4170        ADC #$BF
4180        JSR MON.COUT
4190        DEX
4200        BNE .3
4210        LDA #$A0     PRINT BLANK
4220        JSR MON.COUT
4230        JSR MON.PRADDR
4240        JSR MON.CLREOL
4250        JSR MON.CROUT
4260        LDY #39
4270  .5    LDA BOTTOM.LINE,Y
4280        AND #$3F
4290        STA BASE.LINE24,Y
4300        DEY
4310        BPL .5
4320        DEC MON.CV
4330        RTS
4340  *---------------------------------
4350  SAVE.AREA
4360  SAVE.S .BS 1
4370  SAVE.P .BS 1
4380  SAVE.Y .BS 1
4390  SAVE.X .BS 1
4400  SAVE.A .BS 1
4410  *---------------------------------
4420  *      TEST PROGRAM
4430  *---------------------------------
4440  TEST   JSR TEST1
4450         BRK
4460  TEST1  JSR TEST2
4470  TEST2  JSR TEST3
4480  TEST3  RTS
```

=============================================================================
DOCUMENT :AAL-8108:Articles:Bin.Kbd.Input.txt
=============================================================================

## Binary Keyboard Input

David Holladay, from Madison, Wisconsin, wrote a recent article for the Adam & Eve Apple II Users Group about a technique he uses for turning the Apple keyboard into a Braille input device. He chose 6 keys which can be "simultaneously" depressed to give a composite code. The keys form a 2-by-3 rectangle, like the dots of Braille characters.

Because the Apple keyboard has N-key rollover, simultaneous depression of several keys results in each keycode being sent to the program one at a time. The order that the codes are produced appears random to the program. Some quirks in the way the Apple keyboard is wired up prevent the N-key rollover from working with every combination of keys. Some of them OR together to create a ghost code, different from the actual depressed keys. Apple has used many different keyboards, so the keys which can be used for David's program vary considerably from one Apple to another.

After playing around with his program for a while, I got interested in making a Binary Input Keyboard, rather than a Braille one. My keyboard, which is almost 4 years old (Apple serial # 219!), allows me to press any combination of the keys J, K, L, 1, 2, 3, and 4. I set up these keys with binary weights of hex 40, 20, 10, 08, 04, 02, and 01 respectively.

When you type a combination of these seven keys all at once, the time interval between keys is much shorter than the normal spacing between keystrokes. The program waits for one keyboard strobe, and then initiates a timeout loop. All keycodes received within the timeout window will be considered to have been struck "simultaneously". Each keycode is compared with the list of seven keys (JKL1234), and the appropriate binary weight ORed into the character. If a keycode is received which is not in the legal character list, the bell rings.

I made a test loop which calls the input routine, and displays the hex code on the screen.

The choice of keys (JKL1234) works fine on my Apple, but it may not work on yours. Experiment with various choices until you find seven keys which will work together on your keyboard. Then modify line 1420 with your list of keys, and it will be ready to go.

Possible applications? Maybe fast input of hexadecimal machine language programs. You would have to add one more key so that all eight bits could be specified. And you would have to train your mind and fingers to instantaneously translate from hex to binary finger-patterns. Or, maybe some sort of a game. The basic idea of reading simultaneous keystrokes could effectively create new keys. Or, maybe the basic idea of simultaneous keystrokes could be used for entering secret passwords.

==================================================================
DOCUMENT :AAL-8108:Articles:Compare.2Ways.txt
==================================================================

Two Ways to Compare a Byte.........................Lee Meador

I have noticed two ways to compare a byte used inside DOS and other Apple software.  In the cases I am thinking of, the following code required the Y-register to be zero.  The first way I have seen is straightforward:

```
LDA ...        BYTE TO BE TESTED
CMP #$19       VALUE WE WANT TO TEST FOR
BNE .1         ALSO AFFECTS CARRY STATUS
LDY #0         IF =, CARRY SET
...
```

The other way is a little trickier, but it saves one byte:

```
LDA ...        BYTE TO BE TESTED
EOR #$19       VALUE WE WANT TO TEST FOR
BNE .1         DOESN'T AFFECT CARRY STATUS
TAY            A AND Y BOTH ZERO
...
```

This may help you understand some of those disassemblies you are making, or help you save a byte here and there.

==============================================================================
DOCUMENT :AAL-8108:Articles:DOS33BootROMLst.txt
==============================================================================

Commented Listing of DOS 3.3 Boot ROM

The P5A ROM on your Apple Disk II Controller has a 256-byte program in it which reads track 0 sector 0 into memory and starts executing it.

The data in track 0 sector 0 is read into memory from $0800-08FF. Location $0800 contains a value indicating how many sectors to boot in. This is usually zero, meaning to read only sector zero. However, it could be as high as $0F, meaning to read all 16 sectors of track 0 into memory from $0800-17FF. (The BASICS diskette uses this feature.) Once the selected number of sectors has been read, the boot ROM jumps to $0801 to start execution. At this point (in a normal DOS boot) the rest of DOS is loaded.

My listing starts at $C600, which is where it will be if your controller is in slot 6. The code is all independent of position, so that it can be plugged into any slot. In fact, you can move the code into RAM if you like, just so the second digit of the address is the same as the controller card slot number. I do this some times when I am trying to crack locked disks. I go to the monitor, type 8600<C600.C6FFM, and then patch a BRK opcode on top of the JMP $0801 at $86F8. Then 8600G will read in track 0 sector 0 and BRK back to the monitor, and I can analyze the code to see how the rest is read in.

Enough of that, let's get into the code! Lines 1510-1690 are an esoteric loop which generate the nybble conversion table. The table is built in page 3, from $36C through $3D5. I tried out the loop after storing FF bytes throughout page 3, and got this:

```
0368- FF FF FF FF 00 01 FF FF   03A0- FF 1B FF 1C 1D 1E FF FF
0370- 02 03 FF 04 05 06 FF FF   03A8- FF 1F FF FF 20 21 FF 22
0378- FF FF FF FF 07 08 FF FF   03B0- 23 24 25 26 27 28 FF FF
0380- FF 09 0A 0B 0C 0D FF FF   03B8- FF FF FF 29 2A 2B FF 2C
0388- 0E 0F 10 11 12 13 FF 14   03C0- 2D 2E 2F 30 31 32 FF FF
0390- 15 16 17 18 19 1A FF FF   03C8- 33 34 35 36 37 38 FF 39
0398- FF FF FF FF FF FF FF FF   03D0- 3A 3B 3C 3D 3E 3F FF FF
```

These bytes are referred to at lines 2670 and 2740, indexed from a base of $02D6. This makes a disk code of $96 give a $00 value, and a code of $FF give a value of $3F.

Lines 1710-1790 determine the slot number and multiply it by 16. The JSR MON.RTS is to an RTS instruction in the monitor ROM. The only purpose of this JSR is to put its own address on the stack. Then lines 1720 and 1730 lift up the high byte of the address from the stack. The second digit of this address is the slot number, and 4 ASL's will isolate it and multiply it by 16. Lines 1800-1830 select drive 0 and turn on the motor. (If you want to boot from drive 2, you

**can copy this code into RAM at $8600 and change the byte at $8636 from $8A to $8B.)**

**Lines 1880-1990 move the head to track 0 from wherever it was. If you were already at track 0, it just sits there making a racket as it bangs against the stop. Lines 2030-2070 initialize the track and sector numbers and the memory address to read into.**

**Lines 2090-2480 read a sector into the input area. Lines 2110-2290 are used two different ways, depending on the CARRY status upon entry. The first time CARRY is clear, and we look for an address header (D5 AA 96). After finding an address header the sector and track are check in lines 2300-2480; if they are the ones we want, CARRY is set and we do lines 2110-2290 over again. This time they look for a data header. If one is found, it's time to read the data.**

**Lines 2530-2880 read in the sector. First 86 bytes are read into a little buffer at the bottom of page 3 ($0300-0355). Then 256 bytes are read into the target memory area (normally $0800-08FF). A checksum is computed and checked; if it doesn't match, we start all over. Lines 2770-2880 put the bits from $0300-0355 together with those in the main buffer, in the same way discussed two months ago in the listing of DOS 3.3 B800-BCFF.**

**Lines 2900-2950 check whether we have read all the sectors specified by the first byte of track 0 sector 0. If not, loop back to read the next sector one page higher in memory. When they have all been read, control branches to $0801. The normal DOS boot only reads one sector before branching to $0801.**

```
==========================================================================
DOCUMENT :AAL-8108:Articles:FID.Select.Cat.txt
==========================================================================
```

A Selective Catalog from FID.........................Lee Meador

If you have DOS 3.3, you have no doubt enjoyed using the FID program to copy files from one disk to another.  The wildcard feature in filenames is especially nice, because it lets you set up a semi-automatic copy of a whole set of files, or even the whole disk.

Sometimes I am reluctant to let the wildcard name go through without prompting, because there might be a file or two I don't want copied which matches the specified name.  However, there are so many files involved that I really don't want to sit there and type "Y" for every one of them.  What we need is a "selective catalog" command -- a FID command to list all files names which match the wildcarded-name.

Here are some easy patches which you can apply to FID which will convert the VERIFY command to just what we want.

```
]BLOAD FID                           load FID
]CALL -151                           get to Apple's monitor
*DBE:60                              return before verifying
*C10:EA EA EA                        no double spacing
*3D0G                                return to BASIC
]BSAVE FID/CATALOG,A$803,L$124E      save the new version
```

Now if you BRUN FID/CATALOG you will see the normal FID menu.  Select option 8 (VERIFY), specify a slot and drive, and type a file name (preferably with the "=" wildcard in it).  Specify NO prompting.  When you "PRESS ANY OTHER KEY TO BEGIN" you will see a list of all files whose names match the filename you typed.

Someone else will have to figure out how to get the file type and size to print.

=============================================================================
DOCUMENT :AAL-8108:Articles:FindASLineNums.txt
=============================================================================

Finding Applesoft Line Numbers........................Bob Potts

Sometimes I have needed to know where in memory a certain Applesoft line is located. Maybe I want to patch in a code which cannot be typed from the keyboard. Or maybe the program has been "compressed and optimized", so that the lines are too long to edit. Or maybe I am just curious.

It is simple enough, because the line number is stored in binary at the beginning of each line. I would looke at locations $67,68 to get the address of the first line. Then look at that location to get the address of the next line, and so on. Each line is stored in memory with the first two bytes telling where to find the next line. and the third and fourth bytes giving the line number. Of course, the line number is in binary, and the bytes are backward, and the whole screen is full of hex numbers making it very hard to keep everything straight....

There has to be an easier way! Working with Bob Sander-Cederlof last week, I came up with this simple little program which will print the address of any line in hex. It uses the ampersand (&) statement of Applesoft. You simply BRUN this program, which I call AMPERFIND, and then type an ampersand and the line number. BRUNning sets up the ampersand vector at $3F5-3F7 and returns.

Here is the program. Note that it takes more code to set up the ampersand vector than it takes to do the line number search! Lines 1210-1260 could be put anywhere in memory, just so $3F6 and $3F7 are made to point to that place.

[Bob Potts is an Assistant Vice President at the Bank of Louisville in Kentucky. this bank has 115 Apple IIs in use doing a variety of banking functions.]

================================================================================
DOCUMENT :AAL-8108:Articles:Front.Page.txt
================================================================================

$1.20

Volume 1 -- Issue 10                                    August, 1981

In This Issue...

```
==================================================================
DOCUMENT :AAL-8108:Articles:Miscellaneous.txt
==================================================================
```

Renewing Subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for. If your label says "8109" or "8110", now is the time to renew to be sure of uninterrupted service.

We now have about 500 subscribers, and are shooting for 1000 by the end of the year. (Look for my full page ad in the next NIBBLE.) I am printing 1000 copies of each issue so there will be plenty of back issues for latecomers.

Notice that I have a new address. The old one will still work for a while, but you should start using the new one: Bob Sander-Cederlof, S-C Software, P. O. Box 280300, Dallas, TX 75228.

About Advertising

Do you have a new product you want to test market, which would appeal to the Apple Assembly Line readers? You ought to try an ad in these pages. The current price is $20 for a full page, $10 for a half page. Send it to me just as you want it printed (I can do the reduction to make it fit on the page).

Things For Sale

Here is an up-to-date list of some of the things which I have that you might need:

```
Quarterly Disk #1 (source code from Oct 80 - Dec 80)...$15.00
Quarterly Disk #2 (source code from Jan 81 - Mar 81)...$15.00
Quarterly Disk #3 (source code from Apr 81 - Jun 81)...$15.00
S-C ASSEMBLER II Version 4.0...........................$55.00
Beneath Apple DOS (book)...............................$18.00
Apple Machine Language (book)..........................$11.65
Blank Diskettes (Verbatim, with hub rings, no labels,
                 plain white jackets, in cellophane
                 wrapper)................20 disks for $50.00
Zip-Lock Bags (2-mil, 6"x9")..............100 bags for $8.50
```

If you are interested in getting a regular monthly shipment of 100 or more disks, we can work out an even lower price.

If you are in Texas, remember to send 5% sales tax on books, disks, or bags.

================================================================================
DOCUMENT :AAL-8108:Articles:Rand.Nums.IntBA.txt
================================================================================

Random Number Generator from Integer BASIC

When you are writing games or other simulation exercises, you frequently need a source of random numbers. In Basic it's easy, but how about assembly language?

The WozPak from Call A.P.P.L.E. has directions for calling the RND(X) function in the Integer BASIC ROMs. Remember that this function returns a random integer between 0 and X-1 for an argument X. Linda Egan, from Maywood, California, wrote that she had trouble making the WozPak method work. I don't know what that method was, but I looked up the code in the ROM and came up with some working code.

<random code here>

Lines 1190-1260 are all you need. They set up a call to the ROM code, and pick up the returned value.

Line 1190 sets the X-register to $20. The ROM code uses X for a stack index, and $20 means an empty stack. This is not the hardware stack ($100-1FF), but a software-implemented stack. The stack is in three parts. The part I call IB.LOSTACK runs from $50 thru $6F. IB.HISTACK runs from $A0 thru $BF. A third part runs from $78 thru $97. The ROM code pushes our argument on these stacks like this: the low byte goes on LOSTACK, the high byte on HISTACK, and a zero (from the Y-register) on the FLAGSTACK. (If the value pushed on FLAGSTACK was not zero, it would be used as the high-byte of an address along with the low-byte from LOSTACK to indirectly address the data value.)

Lines 1200 and 1210 store our argument where the ROM code expects it to be, in $CE and $CF. Lines 1240 and 1250 retrieve the resulting random number from the stack.

Lines 1280 through 1420 are a test loop to demonstrate the random function. Twenty lines of eight random numbers each are printed on the screen in hexadecimal. I used an argument of 1000, so all the numbers are between 0 and 999.

What if you don't have the Integer BASIC ROMs in your Apple? Since the code is not very long, you could make your own copy of Woz's routines. I did that, and came up with the following program. I used the same test loop, but this time it is in lines 1760 thru 1900.

Lines 1160 and 1170 save the argument for later use. Lines 1180-1260 get the current random seed from the Apple Monitor and store it in VALUE. However, if the seed was 0000 it is converted to 0100. This is because a seed of 0000 replicates itself forever. Furthermore, the sign bit is stripped off; in other words, VALUE is set to the seed value modulo 32768. This is supposed to force the VALUE to be between 1 and 7FFF.

The random seed is also modified by the monitor whenever you are in KEYIN waiting for an input from the keyboard. This code is at $FD1B thru $FD24 in the monitor ROM. This means the seed might have any (truly random) value between 0000 and FFFF. If by chance it is $8000 when the RND function is called, VALUE will be set to 0000.

Lines 1270-1290 clear two more bytes of VALUE, which will be used later, in the division loop.

Lines 1300-1400 are Woz's algorithm for generating a sequence of random integers. It is a binary polynomial technique, but there seems to be a bug in it. If you run it 32768 times, you should generate each and every value between 0 and $7FFF exactly one time, but in random order. I tested it, and it really generates the values between $6000 and $60FF twice, and never generates $2000-20FF at all! You can play with it and see if there are some seed values which will produce numbers between $2000 and $20FF.

Lines 1420-1440 check the argument. If it is zero, I return the value zero for the function. Integer BASIC would give you "*** >32767 ERR" with a zero argument.

Lines 1490-1650 are a division program, to divide the random VALUE by the LIMIT. After it is finished, the quotient is in VALUE and VALUE+1, and the remainder is in VALUE+2 and VALUE+3. We don't need the quotient; the remainder is the random value we want.

Lines 1690-1710 pick up the result in registers A and Y, and return to the calling program.

==============================================================================
DOCUMENT :AAL-8108:Articles:Re.AsmSrc.Text.txt
==============================================================================

Correction to "Assembly Source on Text Files"

Volume 1, Issue 2 of Apple Assembly Line contained a program for writing assembly source programs for the S-C Assembler II Version 4.0 on DOS text files. Peter Bartlett of Chicago was trying to use it with a Corvus Hard Disk, and found a problem with the program.

The Corvus system will not accept a CLOSE command unless there is a file name on it (unlike regular DOS). One solution is to delete the two calls to CLOSE.FILE at lines 1410 and 1570.

While talking with Peter I discovered a bug in my program, in the subroutine named ISSUE.DOS.COMMAND. It is supposed to allow slot and drive parameters on the file name. This was described in the write-up on page 11. Two errors made it not work.

First, line 1910 says:

```
     1910        CMP #',      COMMA?
```

but the character in the A-register has the high bit set to one.
Cvhange line 1910 to:

```
     1910        CMP #$AC     COMMA?
```

Second, line 1940 says:

```
     1940        STA DOS.BUFFER,Y
```

Change it to:

```
     1940        STA DOS.BUFFER-1,Y
```

The line numbers above correspond to the printed listing in the AAL article. They may not be exactly the same as the source code on Quarterly Disk #1. If you have Quarterly Disk #1 with a serial number of 45 or higher, your copy is already fixed.

==============================================================================
DOCUMENT :AAL-8108:Articles:Rvw.Apple.ML.txt
==============================================================================

Apple Machine Language -- A Review

Many of you have asked me, "What book will help me, an absolute beginner, learn 6502 machine language? I don't know what these other books are talking about!"

If these are your words, then the book "Apple Machine Language", by Don and Kurt Inman, is for you. It is published by Reston Publishing Company, in both hardback ($17.95) and paperback ($12.95). The book has 296 pages, is set in clear, easy-to-read type, and has lots of good diagrams and illustrations.

The authors assume that you are at least familiar with Applesoft Basic. Chapter 1 gives a brief review of Applesoft, with special emphasis on the PEEK, POKE, and CALL statements. (These are the statements you will be using to communicate between Basic and machine language programs.) The authors also assume that you have your own Apple, and that you will not just READ the book. They expect you to follow along every example with your own Apple, so you can EXPERIENCE the material. You will not only learn a lot faster, but it will stick with you and you will UNDERSTAND what is going on.

Chapter 2 takes you across the bridge from Basic to machine language, very gently. You develop, with the authors, a little Applesoft program which helps you enter and test machine language programs.

Chapter 3 finally introduces the ideas of binary numbers, hexadecimal, the A-register in the 6502, and a few instruction codes. You will learn how to load a value into the A-register, modify that value, and store the result back into memory.

There are exercises at the end of each chapter which review the material covered. Don't let that worry you, though...they also printed the answers!

Chapter 4 starts to get interesting and useful. You learn how to use machine language to put some simple color graphics on the Apple screen. You can plot individual points, draw rectangles, and color them in. All the while, you are learning more machine instructions, more registers, more about memory addressing, and so forth.

Chapter 5 introduces you to writing text on the screen. You learn how to call some of the monitor subroutines for text output, how to print characters at particular screen locations, and how to write messages of your choice. Some new instructions are covered, and you learn some new address modes. In particular, you learn all about relative branching.

Chapter 6 is one of my favorites. I have always enjoyed twiddling Apple's little built-in speaker, and this chapter shows you how. You build and play with a tone generator program, even to the point of tuning it up to make a simulated piano keyboard.

Chapter 7 takes you deeper into sound and graphics, helping you code a routine to display the notes as you play them from the keyboard. By the time you finish this chapter you will understand how to use 28 of the 6502's 56 instructions, and 8 of its 13 addressing modes. You will also have used 9 of the subroutines found inside the Apple Monitor ROM.

Chapter 8 takes you inside Apple's Monitor...just a little. Until now, you have been using the Applesoft program developed in chapter 2 to enter and test all your machine language programs. In chapter 8 you learn how to do it from the monitor. You will also learn how to do addition and subtraction.

Chapter 9 show you how to add numbers too big to fit in one byte. Since one byte will only hold numbers between 0 and 255, or between -128 and +127, you can see that most numbers ARE too big to fit in one byte. You will also learn all about the way negative numbers are handled in the 6502.

Chapter 10 delves deeper into the Apple Monitor, and explores 6502 decimal mode arithmetic.

Chapter 11 is only for those fortunate readers who have Integer BASIC in their Apples. It doesn't matter whether Integer BASIC is on the Apple Monitor board, on a firmware card in ROM, or in a 16K RAM card...just so you have it. Why? Because there is another program in there you might not even be aware of: the Apple Mini-Assembler. If you are lucky enough to have it, chapter 11 will tell you how to use it. If not, skipover this chapter and use your S-C ASSEMBLER II instead! On second thought, don't skip chapter 11 entirely. It is here that indirect addressing is covered, and you need to know this material.

Chapter 12, "Putting It All Together", puts it all together. The programming experience you work through is a multiplication subroutine.

There are four appendices which summarize the information about the Apple hardware found throughout the book. Several of the charts in Appendix-A list page number references. (Early editions of the book had blank columns where the page numbers were supposed to be, but that has been corrected.) And finally, there is a regular alphabetic index.

By the time you finish this book, you have a solid foundation for learning to use an assembler like the S-C ASSEMBLER II. I would like to think that my assembler is easy enough to learn that books like this one would not be needed, but there are a lot of concepts that are completely foreign to new computer owners.

I want to do all I can to help every one of you become proficient in assembly language, so I am making "Apple Machine Language" available to you at a discount.  You can buy the $12.95 paperback edition from me for $11.65 (plus 58 cents tax if you are in Texas).  Include a dollar for shipping, so I don't go broke.

================================================================================
DOCUMENT :AAL-8108:Articles:Whaduzzit.Do.txt
================================================================================

What Does This Code Do?.........................John Broderick

What does it do?  Why would you want to use it?  Those who send in correct answers will get their names published here in a few months with the solution.

```
SUBROUTINE:  BRK
             PLA
             PLA
             PLA
             RTS
```

OK, I'll give you a little hint.  One of the five instructions is not used by the 6502 processor.  Can you tell which one?

As far as I know, this routine has never before been published; however, I use it in almost every program I write.  It's a jewel of a routine, worth many times its weight in gold!

Send your answers to John Broderick, 8635 Shagrock, Dallas, TX 75238.  If you have any similar neat code segments, send them with explanation.  I'll try to make this a regular column in the AAL.

```
=========================================================================
DOCUMENT :AAL-8108:DOS3.3:DOS33.Boot.ROM.txt
=========================================================================

 1000  *---------------------------------
 1010  *      DOS 3.3 BOOT ROM $C600.C6FF
 1020  *
 1030  *      COMMENTS BY BOB SANDER-CEDERLOF
 1040  *                  JULY, 4, 1981
 1050  *---------------------------------
 1060  *      DISK CONTROLLER ADDRESSES
 1070  *---------------------------------
 1080  PHOFF  .EQ $C080    PHASE-OFF
 1090  PHON   .EQ $C081    PHASE-ON
 1100  MTROFF .EQ $C088    MOTOR OFF
 1110  MTRON  .EQ $C089    MOTOR ON
 1120  DRV0EN .EQ $C08A    DRIVE 0 ENABLE
 1130  DRV1EN .EQ $C08B    DRIVE 1 ENABLE
 1140  Q6L    .EQ $C08C    SET Q6 LOW
 1150  Q6H    .EQ $C08D    SET Q6 HIGH
 1160  Q7L    .EQ $C08E    SET Q7 LOW
 1170  Q7H    .EQ $C08F    SET Q7 HIGH
 1180  *
 1190  *      Q6    Q7     USE OF Q6 AND Q7 LINES
 1200  *     ----  ----    ----------------------
 1210  *     LOW   LOW     READ (DISK TO SHIFT REGISTER)
 1220  *     LOW   HIGH    WRITE (SHIFT REGISTER TO DISK)
 1230  *     HIGH  LOW     SENSE WRITE PROTECT
 1240  *     HIGH  HIGH    LOAD SHIFT REGISTER FROM DATA BUS
 1250  *---------------------------------
 1260  BUFFER.PNTR .EQ $26,27
 1270  SLOT16 .EQ $2B      SLOT NUMBER TIMES 16
 1280  SECTOR .EQ $3D
 1290  TRACK  .EQ $41
 1300  STACK      .EQ $0100
 1310  POST.NYBBLE.CODES .EQ $02D6
 1320  LITTLE.BUFFER     .EQ $0300
 1330  MON.RTS    .EQ $FF58
 1340  MON.WAIT   .EQ $FCA8
 1350  *---------------------------------
 1360         .OR $C600
 1370         .TA $0800
 1380  *---------------------------------
 1390  BOOT.3.3
 1400         LDX #$20     REDUNDANT INSTRUCTION, USED
 1410  *                   TO IDENTIFY CONTROLLER CARD
 1420  *---------------------------------
 1430  *      GENERATE POST-NYBBLE CONVERSION TABLE
 1440  *          FILLS IN THOSE SLOTS WHOSE INDEX
 1450  *          RELATIVE TO POST.NYBBLE.CODES IS
 1460  *          A VALID NYBBLE CODE.  (VALID CODES
 1470  *          HAVE AT MOST ONE PAIR OF ADJACENT
 1480  *          0-BITS, AND AT LEAST ONE PAIR OF
```

```
1490  *          ADJACENT 1-BITS IN BITS 0-6.)
1500  *---------------------------------
1510         LDY #0
1520         LDX #3       COULD BE ANY VALUE FROM 0 TO $16
1530  *                   3 USED FOR CONTROLLER ID
1540  .1     STX $3C      CHECK CODE FOR VALID NYBBLE
1550         TXA
1560         ASL
1570         BIT $3C      TEST (X .AND. 2*X)
1580         BEQ .3       NO ADJACENT 1-BITS, NO GOOD
1590         ORA $3C      TEST ADJACENT 0-BITS
1600         EOR #$FF     CHANGE TO 1'S FOR TEST
1610         AND #$7E     DON'T CARE ABOUT BIT 7
1620  .2     BCS .3       NOT VALID NYBBLE CODE
1630         LSR
1640         BNE .2
1650         TYA
1660         STA POST.NYBBLE.CODES+$80,X
1670         INY
1680  .3     INX
1690         BPL .1
1700  *---------------------------------
1710         JSR MON.RTS  GET THIS LOCATION ON STACK
1720         TSX          FIND THE PAGE BYTE ON STACK
1730         LDA STACK,X
1740         ASL          ISOLATE SLOT NUMBER
1750         ASL          AND MULTIPLY BY 16
1760         ASL
1770         ASL
1780         STA SLOT16   SLOT NUMBER TIMES 16
1790         TAX
1800         LDA Q7L,X    SET UP TO READ DRIVE
1810         LDA Q6L,X
1820         LDA DRV0EN,X ENABLE DRIVE 0
1830         LDA MTRON,X  TURN ON MOTOR
1840  *---------------------------------
1850  *      MOVE TO TRACK 0 (ASSUME WORST CASE
1860  *      INITIAL POSITION OF TRACK 40).
1870  *---------------------------------
1880         LDY #80      80 HALF-TRACKS
1890  .4     LDA PHOFF,X  STEPPER MOTOR PHASE OFF
1900         TYA          COMPUTE NEXT PHASE
1910         AND #3       YIELDS 3,2,1,0
1920         ASL          YIELDS 6,4,2,0
1930         ORA SLOT16   MERGE WITH SLOT*16
1940         TAX
1950         LDA PHON,X   STEPPER MOTOR PHASE ON
1960         LDA #86      WAIT 19.2 MILLISECONDS
1970         JSR MON.WAIT NO CHANGE TO X OR Y, A=0
1980         DEY          NEXT HALF-TRACK
1990         BPL .4
2000  *---------------------------------
2010  *      A=0, X=SLOT*16
2020  *---------------------------------
```

```
2030        STA BUFFER.PNTR   ($00 --> LOW BYTE OF PNTR)
2040        STA SECTOR 0
2050        STA TRACK 0
2060        LDA #8       BUFFER AT $0800
2070        STA BUFFER.PNTR+1  ($08 --> HI-BYTE OF PNTR)
2080  *--------------------------------
2090  READ.SECTOR
2100  .1    CLC          FLAG CLEAR, LOOK FOR $D5 AA 96
2110  .2    PHP          SAVE FLAG ON STACK
2120  .3    LDA Q6L,X    READ DISK
2130        BPL .3
2140  .4    EOR #$D5
2150        BNE .3       NO
2160  .5    LDA Q6L,X    READ DISK
2170        BPL .5
2180        CMP #$AA
2190        BNE .4
2200        NOP
2210  .6    LDA Q6L,X    READ DISK
2220        BPL .6
2230        CMP #$96
2240        BEQ .7       FOUND ADDRESS MARK: $D5 AA 96
2250        PLP          RETRIEVE FLAG
2260        BCC .1       LOOKING FOR ADDRESS HEADER
2270        EOR #$AD     LOOKING FOR DATA HEADER
2280        BEQ FILL.BUFFER
2290        BNE .1       START ALL OVER
2300  *--------------------------------
2310  .7    LDY #3       READ VOLUME, TRACK, SECTOR
2320  .8    STA $40
2330  .9    LDA Q6L,X    READ DISK
2340        BPL .9
2350        ROL          SAVE UPPER SLICE
2360        STA $3C
2370  .10   LDA Q6L,X    READ DISK
2380        BPL .10
2390        AND $3C      MERGE SLICES
2400        DEY          3RD BYTE YET?
2410        BNE .8       NO, GET ANOTHER
2420        PLP          THROW AWAY FLAG
2430        CMP SECTOR   CORRECT SECTOR?
2440        BNE .1       NO
2450        LDA $40      CORRECT TRACK?
2460        CMP TRACK
2470        BNE .1       NO
2480        BCS .2       YES, SET FLAG FOR DATA HEADER
2490  *                  AND BRANCH BACK ALWAYS
2500  *--------------------------------
2510  *     A=0 ON ENTRY
2520  *--------------------------------
2530  FILL.BUFFER
2540        LDY #86      READ 86 BYTES
2550  .1    STY $3C
2560  .2    LDY Q6L,X    READ BYTE
```

```
2570        BPL .2
2580        EOR POST.NYBBLE.CODES,Y  DECODE BYTE
2590        LDY $3C
2600        DEY
2610        STA LITTLE.BUFFER,Y
2620        BNE .1
2630  *---------------------------------
2640  .3    STY $3C      Y=0
2650  .4    LDY Q6L,X    READ BYTE
2660        BPL .4
2670        EOR POST.NYBBLE.CODES,Y DECODE BYTE
2680        LDY $3C
2690        STA (BUFFER.PNTR),Y
2700        INY
2710        BNE .3
2720  .5    LDY Q6L,X    READ CHECKSUM BYTE
2730        BPL .5
2740        EOR POST.NYBBLE.CODES,Y
2750  .6    BNE READ.SECTOR  BAD CHECKSUM, START OVER
2760  *---------------------------------
2770        LDY #0
2780  .7    LDX #86      PATCH THE 6+2 BACK TOGETHER
2790  .8    DEX
2800        BMI .7       FINISHED A TRIP
2810        LDA (BUFFER.PNTR),Y
2820        LSR LITTLE.BUFFER,X
2830        ROL
2840        LSR LITTLE.BUFFER,X
2850        ROL
2860        STA (BUFFER.PNTR),Y
2870        INY
2880        BNE .8
2890  *---------------------------------
2900        INC BUFFER.PNTR+1  POINT AT NEXT PAGE
2910        INC SECTOR    POINT AT NEXT SECOTR
2920        LDA SECTOR
2930        CMP $0800    SEE IF HAVE READ ENUF SECTORS
2940        LDX SLOT16
2950        BCC .6       NOT ENUF SECTORS YET
2960        JMP $0801    GO TO REST OF BOOT
2970  *---------------------------------
2980        .HS 0000000000  UNUSED BYTES IN ROM
```

```
================================================================================
DOCUMENT :AAL-8108:DOS3.3:Hello.FW.Slot4.txt
================================================================================


∑I»768∑817:∑A:∑I,A:∑:†768Ã∑173,192,192,162,2,189,0,224,221,44,3,208,1
6,202,16,245,162,192,142,184,165,232,142,192,165,173,193,192,96,162,2,
189,0,224,221,47,3,208,242,202,16,245,48,228,32,0,240,76,40,241Ë-∑:®:∑
‰(4)"CATALOG"
```

```
==============================================================================
DOCUMENT :AAL-8108:DOS3.3:S.AMPERFIND.txt
==============================================================================

 1000  *---------------------------------
 1010  *      FIND AN APPLESOFT LINE NUMBER
 1020  *      AND PRINT ADDRESS IN HEX
 1030  *---------------------------------
 1040         .OR $300
 1050         .TF AMPERFIND
 1060  *---------------------------------
 1070  MON.PRNTAX .EQ $F941   PRINT TWO BYTES IN HEX
 1080  AS.LINGET  .EQ $DA0C   CONVERT LINE NUMBER TO BINARY
 1090  AS.FNDLIN  .EQ $D61A   FIND LINE IN APPLESOFT PROGRAM
 1100  *---------------------------------
 1110  *      SET UP AMPERSAND VECTOR
 1120  *---------------------------------
 1130         LDA #$4C     "JMP" OPCODE
 1140         STA $3F5
 1150         LDA #AMPERFIND
 1160         STA $3F6
 1170         LDA /AMPERFIND
 1180         STA $3F7
 1190         RTS
 1200  *---------------------------------
 1210  AMPERFIND
 1220         JSR AS.LINGET  CONVERT LINE NUMBER TO BINARY
 1230         JSR AS.FNDLIN  FIND THE LINE
 1240         LDX $9B
 1250         LDA $9C        GET THE LINE'S ADDRESS
 1260         JMP MON.PRNTAX PRINT THE ADDRESS IN HEX
```

```
=============================================================================
DOCUMENT :AAL-8108:DOS3.3:S.Bin.Keyboard.txt
=============================================================================

 1000  *-----------------------------------
 1010  *      BINARY KEYBOARD
 1020  *-----------------------------------
 1030  MON.CH   .EQ $24
 1040  MON.CV   .EQ $25
 1050  KEYBOARD .EQ $C000
 1060  STROBE   .EQ $C010
 1070  MON.VTAB .EQ $FC24
 1080  MON.HOME .EQ $FC58
 1090  MON.BELL .EQ $FBE2
 1100  MON.PRBYTE .EQ $FDDA
 1110  *-----------------------------------
 1120  GETCHR LDA #0
 1130  .1     STA CHARCODE
 1140         LDA #-16
 1150         STA CNTR
 1160         STA CNTR+1
 1170  .2     LDA KEYBOARD
 1180         BMI .4       SOMETHING TYPED
 1190         INC CNTR
 1200         BNE .2
 1210         INC CNTR+1
 1220         BNE .2
 1230         LDA CHARCODE GET COMPOSITE CODE
 1240         BEQ GETCHR   NO KEYS HIT YET
 1250  .3     RTS
 1260  *-----------------------------------
 1270  .4     STA STROBE   CLEAR KEYBOARD STROBE
 1280         AND #$7F
 1290         CMP #$20     HANDLE BLANK SEPARATELY
 1300         BEQ .3
 1310         LDY #6       SEARCH LIST OF LEGAL KEYS
 1320  .5     CMP LEGAL.KEYS,Y
 1330         BEQ .6
 1340         DEY
 1350         BPL .5
 1360         JSR MON.BELL
 1370         JMP GETCHR
 1380  .6     LDA KEY.BITS,Y
 1390         ORA CHARCODE
 1400         BNE .1       ...ALWAYS
 1410  *-----------------------------------
 1420  LEGAL.KEYS .AS /JKL1234/
 1430  KEY.BITS   .HS 40201008040201
 1440  *-----------------------------------
 1450  CHARCODE   .BS 1
 1460  CNTR       .BS 2
 1470  *-----------------------------------
 1480  *      TEST BINARY KEYBOARD
```

```
1490  *--------------------------------
1500  TEST   JSR MON.HOME
1510  .1     JSR GETCHR
1520         STA $403     LINE 1, COLUMN 4 OF SCREEN
1530         LDA #0
1540         STA MON.CH
1550         STA MON.CV
1560         JSR MON.VTAB
1570         LDA $403
1580         JSR MON.PRBYTE
1590         JMP .1
```

```
================================================================
DOCUMENT :AAL-8108:DOS3.3:S.CallIB.Random.txt
================================================================

 1000  *---------------------------------
 1010  *       RANDOM FUNCTION
 1020  *       ---------------
 1030  *       CALLS SUBROUTINE IN INTEGER BASIC ROM TO GET
 1040  *       A RANDOM NUMBER BETWEEN 0 ANT X-1
 1050  *
 1060  *       CALL:  VALUE X IN Y- AND A-REGISTERS
 1070  *              JSR RANDOM
 1080  *     RETURN:  RANDOM NUMBER IN Y- AND A-REGISTERS
 1090  *              LO-BYTE IN Y, HI-BYTE IN A
 1100  *---------------------------------
 1110  IB.ARG     .EQ $CE,CF
 1120  IB.LOSTACK .EQ $50 THRU $6F
 1130  IB.HISTACK .EQ $A0 THRU $BF
 1140  *---------------------------------
 1150  IB.RANDOM  .EQ $EF51
 1160  MON.PRBYTE .EQ $FDDA
 1170  MON.COUT   .EQ $FDED
 1180  *---------------------------------
 1190  RANDOM LDX #$20     I/B NOUN-STACK POINTER
 1200         STA IB.ARG+1
 1210         STY IB.ARG
 1220         LDY #0       FLAG VALUE ON STACK
 1230         JSR IB.RANDOM
 1240         LDA IB.HISTACK,X
 1250         LDY IB.LOSTACK,X
 1260         RTS
 1270  *---------------------------------
 1280  TEST.RANDOM
 1290         LDA #160
 1300         STA COUNT
 1310  .1     LDY #1000
 1320         LDA /1000
 1330         JSR RANDOM   RND(1000)
 1340         JSR MON.PRBYTE
 1350         TYA
 1360         JSR MON.PRBYTE
 1370         LDA #$A0     PRINT BLANK
 1380         JSR MON.COUT
 1390         DEC COUNT
 1400         BNE .1
 1410         RTS
 1420  COUNT  .BS 1
```

==============================================================
DOCUMENT :AAL-8108:DOS3.3:S.RANDOM.TEST.txt
==============================================================

```
1000  *---------------------------------
1010  *      STAND-ALONE RANDOM FUNCTION
1020  *      ---------------------------
1030  *
1040  *      GET A RANDOM NUMBER BETWEEN 0 AND X-1
1050  *
1060  *      CALL:  VALUE X IN Y- AND A-REGISTERS
1070  *             JSR RANDOM
1080  *    RETURN:  RANDOM NUMBER IN Y- AND A-REGISTERS
1090  *             LO-BYTE IN Y, HI-BYTE IN A
1100  *---------------------------------
1110  MON.RNDL   .EQ $4E
1120  MON.RNDH   .EQ $4F
1130  MON.PRBYTE .EQ $FDDA
1140  MON.COUT   .EQ $FDED
1150  *---------------------------------
1160  RANDOM LDA MON.RNDH GET SEED HI-BYTE
1170         BNE .1       BE SURE SEED BTWN 1 AND 7FFF
1180         CMP MON.RNDL SET CARRY IF BOTH BYTES ZERO
1190         ADC #0       CHANGE 0000 TO 0100
1200  .1     AND #$7F     MAKE SURE NOT LARGER THAN 7FFF
1210         STA MON.RNDH
1220         STA VALUE+1
1230         LDA MON.RNDL
1240         STA VALUE
1250  *---------------------------------
1260         LDY #17      LOOP TO MAKE NEXT RANDOM VALUE
1270  .2     LDA MON.RNDH (WOZNIAK'S ALGORITHM)
1280         ASL
1290         CLC
1300         ADC #$40
1310         ASL
1320         ROL MON.RNDL
1330         ROL MON.RNDH
1340         DEY
1350         BNE .2
1360  *---------------------------------
1370         RTS
1380  *---------------------------------
1390  LIMIT  .BS 2
1400  VALUE  .BS 2
1410  *---------------------------------
1420  TEST.RANDOM
1430         LDA #0
1440         STA COUNT
1450         STA COUNT+1
1460         STA MON.RNDL
1465         INC MON.RNDL
1470         STA MON.RNDH
```

```
1480  TSTLP  JSR RANDOM
1490         LDA VALUE
1500         STA LOBYAD
1510         LDA VALUE+1
1520         ADC #$10
1530         STA HIBYAD
1550         INC $FFFF
1551  LOBYAD .EQ *-2
1552  HIBYAD .EQ *-1
1560         INC COUNT
1570         BNE TSTLP
1580         INC COUNT+1
1590         BPL TSTLP
1600         RTS
1610  *--------------------------------
1620  COUNT  .BS 2
```

```
==============================================================================
DOCUMENT :AAL-8108:DOS3.3:S.Rnd.Function.txt
==============================================================================

 1000  *---------------------------------
 1010  *      STAND-ALONE RANDOM FUNCTION
 1020  *      ---------------------------
 1030  *
 1040  *      GET A RANDOM NUMBER BETWEEN 0 AND X-1
 1050  *
 1060  *      CALL:  VALUE X IN Y- AND A-REGISTERS
 1070  *             JSR RANDOM
 1080  *    RETURN:  RANDOM NUMBER IN Y- AND A-REGISTERS
 1090  *             LO-BYTE IN Y, HI-BYTE IN A
 1100  *---------------------------------
 1110  MON.RNDL   .EQ $4E
 1120  MON.RNDH   .EQ $4F
 1130  MON.PRBYTE .EQ $FDDA
 1140  MON.COUT   .EQ $FDED
 1150  *---------------------------------
 1160  RANDOM STY LIMIT    SAVE LIMIT VALUE
 1170         STA LIMIT+1
 1180         LDA MON.RNDH GET SEED HI-BYTE
 1190         BNE .1       BE SURE SEED BTWN 1 AND 7FFF
 1200         CMP MON.RNDL SET CARRY IF BOTH BYTES ZERO
 1210         ADC #0       CHANGE 0000 TO 0100
 1220  .1     AND #$7F     MAKE SURE NOT LARGER THAN 7FFF
 1230         STA MON.RNDH
 1240         STA VALUE+1
 1250         LDA MON.RNDL
 1260         STA VALUE
 1270         LDA #0
 1280         STA VALUE+2
 1290         STA VALUE+3
 1300  *---------------------------------
 1310         LDY #17      LOOP TO MAKE NEXT RANDOM VALUE
 1320  .2     LDA MON.RNDH (WOZNIAK'S ALGORITHM)
 1330         ASL
 1340         CLC
 1350         ADC #$40
 1360         ASL
 1370         ROL MON.RNDL
 1380         ROL MON.RNDH
 1390         DEY
 1400         BNE .2
 1410  *---------------------------------
 1420         LDA LIMIT
 1430         ORA LIMIT+1
 1440         BEQ .5       RETURN ZERO
 1450  *---------------------------------
 1460  *      DIVIDE RANDOM VALUE (1-7FFF) BY LIMIT
 1470  *      AND USE REMAINDER (0<=REMAINDER<LIMIT)
 1480  *---------------------------------
```

```
1490         LDY #16      LOOP FOR 16-BITS
1500  .3     ASL VALUE    DOUBLE DIVIDEND
1510         ROL VALUE+1
1520         ROL VALUE+2
1530         ROL VALUE+3
1540         LDA VALUE+2
1550         CMP LIMIT
1560         LDA VALUE+3
1570         SBC LIMIT+1
1580         BCC .4       PARTIAL DIVIDEND < LIMIT
1590         STA VALUE+3
1600         LDA VALUE+2  CARRY IS SET, SUBTRACT
1610         SBC LIMIT    LO-BYTE OF LIMIT
1620         STA VALUE+2
1630         INC VALUE    SET BIT IN QUOTIENT
1640  .4     DEY
1650         BNE .3
1660  *---------------------------------
1670  *      RETURN RANDOM VALUE MOD LIMIT
1680  *---------------------------------
1690  .5     LDA VALUE+3  PICK UP REMAINDER FROM DIVISION
1700         LDY VALUE+2
1710         RTS
1720  *---------------------------------
1730  LIMIT  .BS 2
1740  VALUE  .BS 4
1750  *---------------------------------
1760  TEST.RANDOM
1770         LDA #160
1780         STA COUNT
1790  .1     LDY #1000
1800         LDA /1000
1810         JSR RANDOM   RND(1000)
1820         JSR MON.PRBYTE
1830         TYA
1840         JSR MON.PRBYTE
1850         LDA #$A0     PRINT BLANK
1860         JSR MON.COUT
1870         DEC COUNT
1880         BNE .1
1890         RTS
1900  COUNT  .BS 1
```

```
================================================================================
DOCUMENT :AAL-8109:Articles:CHRGET.CHRGOT.txt
================================================================================
```

CHRGET and CHRGOT in Applesoft

On pages 13 and 14 of the September 1981 Kilobaud Microcomputing (Robert Baker's Pet-Pourri column) there is a good description of the CHRGET/CHRGOT duo. These two subroutines (really two entry points into one routine) seem to be common to the Microsoft Basics, at least the 6502 versions.

What are they? When Applesoft initializes itself one of the tasks is to copy a short subroutine into page zero, from $00B1 through $00C8. There is no difference between the PET and the Apple versions, except that the PET version is copied into $0070-0087. Here is the code:

<chrget/chrgot routines here>

Almost every time Applesoft wants to look at a character from your program or even from the input buffer, it does so by calling this subroutine. The CHRGET entry increments the address used to pick up the next character, and then falls into CHRGOT. In either case, the character is picked up and several tests are performed. Blanks are passed over, ignored. Colon (end of statement) and $00 (end of line) set the Z status bit. Digits clear CARRY, non-digits set CARRY. The calling program can use these status bits. For example:

```
JSR CHRGET
BEQ END        BRANCH IF COLON OR END-OF-LINE
BCC DIGIT      BRANCH IF CHAR IS DIGIT (0-9)
```

The article in Kilobaud suggests patching this routine at $00BA to jump to your own code. Your program can trap certain characters for special functions, in much the same way as the "&" is now handled by Applesoft. You just have to be sure that you execute the instructions your JMP overlayed before returning to the remainder of CHRGET. It appears that many of the enhancement packages available for PET Basic use this scheme.

Why use this patching scheme instead of the "&" for special functions? Because your special functions can be made to appear an integral part of the language, without the telltale ampersand. Because even special codes inside expressions or other statements can be trapped. Because you want to encode or otherwise obfuscate your program for security. Because you just want to be different. Of course, the disadvantage is that the entire operation of Applesoft is slowed down by the amount of time your extra testing takes, since every character retrieved by the interpreter will go through your routine as well as the standard CHRGET.

Here is a sample patch program, just show how it is done. Any time the patch discovers a "#" character, it will ring the Apple's bell.

The sample Applesoft lines show what I mean.  If you want to try out the patch, assemble it and then call Applesoft.  Then get to the monitor and patch CHRGET like this:

```
]CALL -151
*BA:4C 00 03
*3D0G
```

Then enter some Applesoft lines with embedded "#" characters, and RUN.

If you think of some really practical ways to use patches like this, let me know about them.

==============================================================================
DOCUMENT :AAL-8109:Articles:DOS3.3.RWTS.Src.txt
==============================================================================

Commented Listing of DOS 3.3 RWTS

Last March I started out this series of DOS listings with the RWTS portion of DOS 3.2.1. Since then I have printed all of DOS 3.2.1 and DOS 3.3 from $B800 thru $BFFF, except for DOS 3.3 RWTS. Somehow it almost was overlooked, but here it is now.

There are minor differences between the two versions of RWTS, which you can find by comparing the listing from the March 1981 issue of AAL and this one. The differences start at line 1810. I suppose the changes are meant to be improvements, but most of them seem to make very little difference.

One critical major difference: DOS 3.2.1 and previous versions use sector numbers which are actually written in the headers. DOS 3.3 uses two different sets of sector numbers: physical and logical. The physical sector numbers are recorded in the sector header blocks; logical sector numbers are used in RWTS calls and File Manager calls. The translation is performed using the table at line 4280, which I have called the PHYSICAL.SECTOR.VECTOR. This table is accessed at line 3310: the logical sector number is in the Y-register, and indexes into the physical sector vector to pick up a physical sector number.

==================================================================
DOCUMENT :AAL-8109:Articles:Fancy.AS.Direct.txt
==================================================================

A New, Fancier .AS Directive

Many times I write text printing loops that depend on the sign bit of each byte to indicate the end of text. I might set up the text this way:

```
     .AS /THIS IS THE TEXT I WANT TO PRIN
     .AS -/T/
```

This assembles with the sign bits off (0) on all the characters of the text except the last one. I can terminate my printing loop by testing that bit. A little later, I will show you an example of just such a loop.

But when there are many messages, I get tired of using separate lines for the last character of each message! Why not have an assembler directive which automatically sets the sign bit of the last character the opposite of the sign bits of the rest of the message? Since Version 4.0 of the S-C Assembler II has a .US directive for me, the user, to program....

The only problem is that how to program for the .US directive has never been revealed. Until now.

The following little program will implement just the directive I want, and install it as the .US directive. It uses five programs inside the assembler (see lines 1100-1140). The code is patterned directly after the code for the .AS directive, which starts at $203C in most copies of Version 4.0.

NOTE: You should check your assembler to make sure that the four bytes starting at $203C are "A0 00 84 04"; if they are, you can use the same addresses for the five routines as I have shown here. (If not, send me your original Version 4.0 disk for a free update. Be sure to adequately protect the disk for shipping, because your new copy will come back on the same disk.)

Line 1000 sets the origin of the code to $0F00. You could use some other origin, like $0300, if you wish. Just be sure it is an area of memory that you will not be using for some other purpose wile you are assembling. Line 1010 directs the object code to a BRUNnable file named B.US.DIRECTIVE.

The code from 1160 to 1210 is executed when you BRUN B.US.DIRECTIVE. It stores the address of DIR.US in the .US vector at the beginning of the assembler. You can read a little about this on page 15 of the Version 4.0 update manual.

Lines 1030-1050 define a few variables. WBUF is the line buffer the assembler uses, starting at $0200. The assembler unpacks a line from the source code into this buffer, and then proceeds to analyze it. DLIM and HIBIT are temporary locations in page zero where I will save the delimiter character and the high-bit setting.

The meat of the directive is in lines 1230-1510. If you disassemble the code at $203C in the S-C Assembler II, you will see a marked similarity here. You might also try disassembling the code for the GNNB and GNC subroutines.

GNC retrieves the next character from WBUF and increments the pointer. The character is tested. Carry status is set if the end-of-line token was picked up. Equal status is set if a blank or end-of-line token was picked up. GNNB calls on GNC until a non-blank character is found. GNC returns with the character in the A-register, and the pointer to the next character in the Y-register.

Lines 1240-1310 scan from the end of the opcode field to try to find the delimiter. If no non-blank character is found after the opcode field, you will get the "BAD ADDRESS ERROR". If a minus sign is found, $80 is stored in HIBIT instead of $00. This value will be merged with every character between the delimiters, to set or clear the high-bit of each byte. When the delimiter is found, it is stored in DLIM.

Lines 1320-1350 check to make sure that there are some characters after the delimiter before the next occurrence of the delimiter. For example, if you write ".US //", I want to assemble no bytes and go on. If I find the end-of-line token, you will get the error message.

Lines 1360-1430 are a loop to output the bytes one by one. I have to look ahead to see if the next character is the delimiter again. If not, then I will output the current character (by now accessed with "LDA WBUF-2,Y", because Y has been advanced). If the next one is the delimiter, then the current one is the last character of the string; I will have to go to ".3", to handle the last character.

Lines 1450-1490 handle the last character of the string between the delimiters. The high-bit is first set just like all the rest of the bytes at line 1460, and then reversed with the EOR #$80 at line 1470.

There is no end to the detail we could get into by describing how EMIT, CMNT, and ERBA work. I will leave them for you to puzzle over at your leisure. (Can't give away the whole plot in chapter 1!)

<code for dir.us>

The following program shows how I might use the new .US directive I have just built. It prints the line of text from line 1230 ten times on the screen. The .US directive assures that I can tell when I am at the end of the text string by looking at the sign bit. That is just what the BMI opcode at line 1110 is doing. Lines 1070, 1080, 1190, and 1200 are the looping code to make ten copies of the line. Lines

1090-1150 print the message except for the last character; lines 1170-1180 print that last character and a carriage return.

```
==============================================================================
DOCUMENT :AAL-8109:Articles:FieldInputRtn.txt
==============================================================================
```

Field Input Routine for Applesoft....................Bob Potts

Inputting strings to an Applesoft program is normally a simple task. What could be easier than "INPUT A$"? But, that method will not allow commas or colons.

Another easy way is to use GET C$ for each character, and append them to a string using A$=A$+C$. But, by the time you add the testing for each input character to find the end of input and other possible control characters, the routine can be terribly slow. Furthermore, it eats up string space like crazy; eventually Applesoft garbage collection starts, and the program dies for a while. Here is the kind of loop I am talking about:

```
10 A$=""
20 GET C$
30 <perform various tests on C$>
40 A$=A$+C$:PRINT C$;
50 GO TO 20
```

As the string increases in length, the speed decreases dramatically. In fact, some characters may be lost if you are a fast typist.

One way to correct this is to use a machine language routine to input each keystroke, test it, and build a string for the Applesoft program. Such a routine was printed in "Apple Assembly Line" issue #7 (April, 1981), pages 6-8. But that routine used the monitor's RDLINE subroutine to input the string. I needed a routine more adapted to inputting a series of fields, using the screen in a "fill-in-the-blanks" mode.

The following program was designed for use in the various branches of the Bank of Louisville. The Apple is used to calculate loans, print the installment notes, and to enter loan applications. A loan application involves filling in the blanks on several screens full of prompts.

To use the input routine, you first position the cursor to the start of field using VTAB and HTAB; then set a field length using the SCALE= statement, and a field code using the ROT= statement. The actual call to the input routine is done with "&INPUT" and the name of your string. Here is an example for inputting a 5-character field starting in column 10 of line 7:

```
10 VTAB 7 : HTAB 10 : SCALE=5 : ROT = 0 : &INPUT A$
```

The input routine allows skipping from field to field, either forward or backward through a form. Backspace and copy (right arrow) are supported. Filling up a field, or hitting RETURN within a field, finish that field and return the value to Applesoft. An EXIT CODE tells the Applesoft program whether a value was returned in the string or some other exit was chosen. You access the exit code with a PEEK(224). Here are the four exit codes and their meanings:

```
EXIT CODE = 0     Field was filled or RETURN typed.
          = 1     ESCAPE was typed at beginning of field.
          = 2     CTRL-F was typed at beginning of field.
          = 3     Left Arrow (backspace) was typed
                  at beginning of field.
```

If the exit code is zero, then the field data you typed is in your string. Otherwise, the string's value is not changed. Finishing a field by either filling it up or hitting RETURN puts the field data into your string, and I then advance to the next field on the form. I use an exit code of 3 (backspace at beginning of field) to mean that the Applesoft program should go back to the previous field on the current form.

How you use the exit codes of 1 and 2 is up to you. You might use an ESCAPE (exit code = 1) to abort the form-filling and return to a main menu. The ESCAPE is now only recognized if you are at the beginning of the field and the field code is non-zero. Of course, you could change that. You might use the control-F to mean you are finished with the current form.

How Does It Work?

Line 1110 sets the origin to $0300. If you already have something else in page 3, you can change the origin to whatever suits your fancy. Just remember to set the correct values for HIMEM and LOMEM to protect it from Applesoft, and vice versa.

Lines 1380-1440 install the ampersand vector. If you BRUN the program, this code is executed. If you BLOAD it, then CALL 768 will execute it. You only have to execute this once in your program. Once done, any occurrence of an ampersand statement in your program will branch to INPUT.FIELD, at line 1460.

Lines 1460-1500 check for the keyword "INPUT", and a string variable name. The three routines (and others used in this program) starting with "AS." are in the Applesoft ROMs. AS.SYNCHR compares the current character with what is in the A-register; if different you get SYNTAX ERROR, and if the same the character pointer is advanced. AS.PTRGET scans a variable name and finds its descriptor in memory. AS.CHKSTR makes sure that the variable is a string (if not you get TYPE MISMATCH). At this point the address of the string descriptor is in $83,84. The address in $83,84 points to 3 bytes which tell the length and address of the string's contents.

Lines 1520-1690 test the input character and branch accordingly. I use MON.RDKEY to read the character, which means that the data could come from any I/O slot as well as the normal Apple Keyboard. You could add more tests here, or remove some. If it is a printing character, we fall into lines 1730-1810 to store the character in the input buffer and on the screen. If the filed is now full, line 1810 jumps to the routine which passes the data to Applesoft. Note that characters stored in the input buffer have the high-bit equal to zero (Applesoft likes them that way). Characters written on the screen have the high-bit set to one, so that they print in NORMAL video.

Lines 1920-1990 handle the backspace character. If you are at the beginning of a field, the routine will return with an exit code of 3. Otherwise, the current character will be replace on the screen with an underline character, and the cursor will be backed up.

Lines 2030-2050 handle the right arrow. Normally this just copies over a character on the screen. Characters are picked up from the screen image, and the treated just as though they came from the keyboard. Note that the right arrow will not advance over an underline character.

Lines 2090-2140 handle ESCAPE. As I mentioned earlier, ESCAPE is ignored unless it is typed when the cursor is at the beginning of the field, and the field code is non-zero. This is the only use for the field code in the input routine presented here, but you might think of many more uses and make your own modifications.

Lines 2180-2190 make Applesoft allocate some space for the string in the normal string data space. Then lines 2200-2270 set up the string variable's descriptor to point to this space. Lines 2280-2310 move the string data from the input buffer up to the new place. This code was copied from the "Fast String Input Routine" in AAL #7.

The input routine is presented here in a very simple form; I leave it up to you to modify it to suit your most demanding applications.

Here is a brief sample showing how you might use the input routine to fill in five fields:

<sample program>

```
==============================================================================
DOCUMENT :AAL-8109:Articles:Front.Page.txt
==============================================================================
```

$1.20

Volume 1 -- Issue 12                              September, 1981

In This Issue...



Quarterly Disk #4

The fourth Quarterly Disk is now ready, containing all the source code from issues 10 through 12. The cost is only $15, and it will save you a lot of typing and possible searching for typos. All previous Quarterly Disks are still available, at the same price.

Renewing subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for. If it says "8109", this is your last issue. Unless, of course, I receive your renewal check for $12. If your label says 8111 or less, now is the time to renew!

More about the Firmware Card in Slot 4

Michael Sanders' DOS patch for using the Firmware card in slot 4 is really nice. A lot of you have written or called about it, and I use it myself now. In fact, I have changed my HELLO programs to do the patch. All it takes is two POKEs:

```
     10 POKE 42424,192 : POKE 42432,193
```

I like doing it this way a lot better than INITting a disk with a modified DOS. If you want to test for the presence of a card before patching, you can do it like this:

<<<Applesoft listing>>>

=================================================================
DOCUMENT :AAL-8109:Articles:LeaveVers4.0.txt
=================================================================

Leaving the S-C Assembler II

How do you get out of the assembler? I suppose I could have made a QUIT or EXIT command, but I didn't. If you want to go to Applesoft or Integer BASIC, type FP or INT. You will then be instantly in the version of Basic you wanted. However, you will still be hooked into the Assembler's output subroutine. If you load a small program and LIST it, you will find that tapping the space bar will stop the listing and restart it, just as inside the assembler. Notice I said a "small" program; a large program might over-write part of the assembler, causing the computer to hang up.

What you must do is type FP or INT, and then PR#0. The PR#0 unhooks the assembler output routine, and you are free.

Now, if you are sure that you have not over-written the assembler with your Applesoft or Integer BASIC program, and you want to return to the assembler, you can do so by typing CALL 4096. I use this for going back and forth rapidly when I am testing &-routines and the like.

What if you want to leave the assembler to go to the monitor? First of all, remember that you can use all of the monitor commands without ever leaving the assembler, by typing a dollar sign and then the monitor command. But if you really want out, how do you get there? If you have an old monitor ROM (not AUTOSTART), hitting RESET will get you to the monitor. With the Autostart ROM, you can type $FF59G or $FF69G. The first will unhook DOS, while the second will leave DOS hooked in. (The second is the same as the Basic command CALL-151.) Still another way is to patch the Autostart ROM RESET vector at $3F2 (type "$3F2:69 FF 5A"), so that RESET enters the monitor.

And how do you get back to the assembler from the monitor, without disturbing or losing your source code? Simply type "1003G" and you will be there. If you type "1000G" you will also get to the assembler, but all your source code will be gone, just as though you had typed the "NEW" command.

=============================================================================
DOCUMENT :AAL-8109:DOS3.3:Demo.US.Direct.txt
=============================================================================

```
 1000  *----------------------------------
 1010  *       DEMONSTRATE USE OF .US DIRECTIVE
 1020  *----------------------------------
 1030  MON.COUT    .EQ $FDED
 1040  MON.CROUT   .EQ $FD8E
 1050  *----------------------------------
 1060  DEMO.US
 1070         LDA #10      DO 10 LINES
 1080         STA LINE.COUNT
 1090  .3     LDY #0
 1100  .1     LDA TEXT,Y   GET CHAR FROM TEXT STRING
 1110         BMI .2
 1120         ORA #$80     MAKE NORMAL VIDEO
 1130         JSR MON.COUT
 1140         INY          NEXT CHARACTER
 1150         BNE .1       ...ALWAYS
 1160  *----------------------------------
 1170  .2     JSR MON.COUT
 1180         JSR MON.CROUT
 1190         DEC LINE.COUNT
 1200         BNE .3
 1210         RTS
 1220  *----------------------------------
 1230  TEXT   .US /THIS IS MY MESSAGE/
 1240  LINE.COUNT .BS 1
 1250  *----------------------------------
```

==============================================================================
DOCUMENT :AAL-8109:DOS3.3:S.CHRGET.PATCH.txt
==============================================================================

```
 1000  *---------------------------------
 1010  *      SAMPLE APPLESOFT FILTER PROGRAM
 1020  *---------------------------------
 1030         .OR $BA
 1040         JMP FILTER
 1050  *---------------------------------
 1060         .OR $300
 1070  FILTER CMP #'#      CHECK FOR "#" CHARACTER
 1080         BNE .1       NO, PASS UNMOLESTED
 1090         JSR WHATEVER.YOU.WANT
 1100         JMP $B1
 1110  .1     CMP #$3A     CHECK FOR COLON
 1120         BCS .2       YES, RETURN JUST CHRGET WOULD
 1130         JMP $BE      NO, RECONNECT WITH CHRGET
 1140  .2     RTS
 1150  *---------------------------------
 1160  WHATEVER.YOU.WANT
 1170         JSR $FBE2    RING BELL
 1180         RTS
```

```
=============================================================================
DOCUMENT :AAL-8109:DOS3.3:S.CHRGET.txt
=============================================================================

 1000  *--------------------------------
 1010  *      APPLESOFT CHRGET/CHRGOT SUBROUTINES
 1020  *--------------------------------
 1030         .OR $00B1
 1040  *--------------------------------
 1050  TXTPTR .EQ $B8      INSIDE 'LDA' INSTRUCTION
 1060  *--------------------------------
 1070  CHRGET INC TXTPTR   INCREMENT ADDRESS OF NEXT CHARACTER
 1080         BNE CHRGOT
 1090         INC TXTPTR+1
 1095  *--------------------------------
 1100  CHRGOT LDA $8888    PICK UP THE NEXT CHARACTER
 1110         CMP #$3A     TEST IF COLON
 1120         BCS .1       YES, Z AND C SET, RETURN
 1130         CMP #$20     TEST IF BLANK
 1140         BEQ CHRGET   YES, IGNORE IT
 1150         SEC          DO DIGIT TEST
 1160         SBC #$30
 1170         SEC          SET Z IF VALUE WAS $00 (EOL TOKEN)
 1180         SBC #$D0     AND CLEAR CARRY IF DIGIT ($30-39)
 1190  .1     RTS
```

```
=====================================================================
DOCUMENT :AAL-8109:DOS3.3:S.D33.BD00BEAE.txt
=====================================================================

 1000  *---------------------------------
 1010  *      DOS 3.3 DISASSEMBLY   $BD00-BEAE
 1020  *      BOB SANDER-CEDERLOF       3-3-81
 1030  *---------------------------------
 1040  CURRENT.TRACK       .EQ $478
 1050  DRIVE.1.TRACK       .EQ $478 THRU 47F (INDEX BY SLOT)
 1060  DRIVE.2.TRACK       .EQ $4F8 THRU 4FF (INDEX BY SLOT)
 1070  SEARCH.COUNT        .EQ $4F8
 1080  RETRY.COUNT         .EQ $578
 1090  SLOT                .EQ $5F8
 1100  SEEK.COUNT          .EQ $6F8
 1110  *---------------------------------
 1120  PHASE.OFF           .EQ $C080
 1130  PHASE.ON            .EQ $C081
 1140  MOTOR.OFF           .EQ $C088
 1150  MOTOR.ON            .EQ $C089
 1160  ENABLE.DRIVE.1      .EQ $C08A
 1170  ENABLE.DRIVE.2      .EQ $C08B
 1180  Q6L                 .EQ $C08C
 1190  Q6H                 .EQ $C08D
 1200  Q7L                 .EQ $C08E
 1210  Q7H                 .EQ $C08F
 1220  *---------------------------------
 1230  SECTOR     .EQ $2D
 1240  TRACK      .EQ $2A
 1250  VOLUME     .EQ $2F
 1260  DRIVE.NO   .EQ $35
 1270  DCT.PNTR   .EQ $3C,3D
 1280  BUF.PNTR   .EQ $3E,3F
 1290  MOTOR.TIME .EQ $46,47
 1300  IOB.PNTR   .EQ $48,49
 1310  *---------------------------------
 1320  PRE.NYBBLE          .EQ $B800
 1330  WRITE.SECTOR        .EQ $B82A
 1340  READ.SECTOR         .EQ $B8DC
 1350  READ.ADDRESS        .EQ $B944
 1360  POST.NYBBLE         .EQ $B8C2
 1370  SEEK.TRACK.ABSOLUTE .EQ $B9A0
 1380  DELAY.LOOP          .EQ $BA00
 1390  *---------------------------------
 1400  ERR.WRITE.PROTECT   .EQ $10
 1410  ERR.WRONG.VOLUME    .EQ $20
 1420  ERR.BAD.DRIVE       .EQ $40
 1430  *---------------------------------
 1440         .OR $BD00
 1450         .TA $800
 1460  *---------------------------------
 1470  RWTS   STY IOB.PNTR SAVE ADDRESS OF IOB
 1480         STA IOB.PNTR+1
```

```
1490          LDY #2
1500          STY SEEK.COUNT  UP TO 2 RE-CALIBRATIONS
1510          LDY #4
1520          STY SEARCH.COUNT
1530          LDY #1       POINT AT SLOT# IN IOB
1540          LDA (IOB.PNTR),Y  SLOT# FOR THIS OPERATION
1550          TAX
1560          LDY #15      POINT AT PREVIOUS SLOT#
1570          CMP (IOB.PNTR),Y  SAME SLOT?
1580          BEQ .3       YES
1590          TXA          SAVE NEW SLOT ON STACK
1600          PHA
1610          LDA (IOB.PNTR),Y  GET OLD SLOT#
1620          TAX
1630          PLA          STORE NEW SLOT #
1640          PHA          INTO OLD SLOT# SPOT
1650          STA (IOB.PNTR),Y
1660   *---------------------------------
1670   *      SEE IF OLD MOTOR STILL SPINNING
1680   *---------------------------------
1690          LDA Q7L,X    GO INTO READ MODE
1700   .1     LDY #8       IF DATA DOES NOT CHANGE
1710          LDA Q6L,X       FOR 96 MICROSECONDS,
1720   .2     CMP Q6L,X       THEN THE DRIVE IS STOPPED
1730          BNE .1       WOOPS! IT CHANGED!
1740          DEY          TIME UP YET?
1750          BNE .2       NO, KEEP CHECKING
1760          PLA          GET NEW SLOT # AGAIN
1770          TAX
1780   *---------------------------------
1790   .3     LDA Q7L,X    SET UP TO READ
1800          LDA Q6L,X
1810          LDY #8
1820   .31    LDA Q6L,X    GET CURRENT DATA
1830          PHA          7 CYCLE DELAY
1840          PLA
1850          PHA          7 CYCLE DELAY
1860          PLA
1870          STX SLOT
1880          CMP Q6L,X    SEE IF DATA CHANGED
1890          BNE .32      YES, IT CHANGED
1900          DEY
1910          BNE .31      KEEP WAITING
1920   .32    PHP          SAVE ANSWER ON STACK
1930          LDA MOTOR.ON,X   TURN ON MOTOR
1940          LDY #6       COPY POINTERS INTO PAGE ZERO
1950   .4     LDA (IOB.PNTR),Y
1960          STA DCT.PNTR-6,Y
1970          INY          DCT.PNTR .EQ $3C,3D
1980          CPY #10      BUF.PNTR .EQ $3E,3F
1990          BNE .4
2000          LDY #3       GET MOTOR ON TIME FROM DCT
2010          LDA (DCT.PNTR),Y
2020          STA MOTOR.TIME+1  HIGH BYTE ONLY
```

```
2030        LDY #2       GET DRIVE #
2040        LDA (IOB.PNTR),Y
2050        LDY #16      SEE IF SAME AS OLD DRIVE#
2060        CMP (IOB.PNTR),Y
2070        BEQ .5       YES
2080        STA (IOB.PNTR),Y  UPDATE OLD DRIVE #
2090        PLP          SET Z STATUS
2100        LDY #0       TO FLAG MOTOR OFF
2110        PHP
2120  .5    ROR          CHECK LSB OF DRIVE #
2130        BCC .6       DRIVE 2
2140        LDA ENABLE.DRIVE.1,X
2150        BCS .7       ...ALWAYS
2160  .6    LDA ENABLE.DRIVE.2,X
2170  .7    ROR DRIVE.NO SET SIGN BIT IF DRIVE 1
2180        PLP          WAS MOTOR PROBABLY OFF?
2190        PHP
2200        BNE .9       NO, DEFINITELY ON
2210  *--------------------------------
2220  *     DELAY FROM 150 TO 180 MILLISECONDS,
2230  *     DEPENDING ON WHAT GARBAGE IS IN A-REG
2240  *--------------------------------
2250        LDY #7       YES, WAIT A WHILE
2260  .8    JSR DELAY.LOOP
2270        DEY          BUT IT WORKS ANYWAY....
2280        BNE .8
2290        LDX SLOT     RESTORE SLOT#
2300  *--------------------------------
2310  .9    LDY #4       GET TRACK #
2320        LDA (IOB.PNTR),Y
2330        JSR SEEK.TRACK
2340        PLP          WAS MOTOR DEFINITELY ON?
2350        BNE PROCESS.COMMAND  YES, MOTOR ON
2360        LDY MOTOR.TIME+1   SEE IF NEED TO WAIT
2370        BPL PROCESS.COMMAND   NO
2380  *--------------------------------
2390  *     MOTOR WAS OFF, SO WAIT REST OF MOTOR ON TIME
2400  *     FOR APPLE DISK II, MOTOR ON TIME IS 1 SECOND.
2410  *     PART OF THIS TIME IS COUNTED DOWN WHILE SEEKING
2420  *     FOR THE TRACK.
2430  *--------------------------------
2440  .10   LDY #18      ABOUT 100 MICROSECONDS PER TRIP
2450  .11   DEY
2460        BNE .11
2470        INC MOTOR.TIME
2480        BNE .10
2490        INC MOTOR.TIME+1
2500        BNE .10
2510  *--------------------------------
2520  *     MOTOR ON AND UP TO SPEED, SO LET'S
2530  *     FIND OUT WHAT THE COMMAND IS AND DO IT!
2540  *--------------------------------
2550  PROCESS.COMMAND
2560        LDY #12      GET COMMAND
```

```
2570        LDA (IOB.PNTR),Y
2580        BEQ .8       NULL COMMAND, LET'S LEAVE
2590        CMP #4       FORMAT?
2600        BEQ .9       YES
2610        ROR          SET CARRY=1 IF READ, =0 IF WRITE
2620        PHP          SAVE ON STACK
2630        BCS .1       READ
2640        JSR PRE.NYBBLE  WRITE
2650  .1    LDY #48      UP TO 48 RETRIES
2660        STY RETRY.COUNT
2670  .2    LDX SLOT     GET SLOT NUMBER AGAIN
2680        JSR READ.ADDRESS
2690        BCC .5       GOOD ADDRESS READ
2700  .21   DEC RETRY.COUNT
2710        BPL .2       KEEP TRYING
2720  .3    LDA CURRENT.TRACK  GET TRACK WE WANTED
2730        PHA          SAVE IT
2740        LDA #96      PRETEND TO BE ON TRACK 96
2750        JSR SETUP.TRACK
2760        DEC SEEK.COUNT
2770        BEQ .6       NO MORE RE-CALIBRATES
2780        LDA #4
2790        STA SEARCH.COUNT
2800        LDA #0       LOOK FOR TRACK 0
2810        JSR SEEK.TRACK
2820        PLA          GET TRACK WE REALLY WANT
2830  .4    JSR SEEK.TRACK
2840        JMP .1
2850  *---------------------------------
2860  .5    LDY $2E      TRACK# IN ADDRESS HEADER
2870        CPY CURRENT.TRACK
2880        BEQ .10      FOUND RIGHT TRACK
2890        LDA CURRENT.TRACK
2900        PHA          SAVE TRACK WE REALLY WANT
2910        TYA          SET UP TRACK WE ACTUALLY FOUNG
2920        JSR SETUP.TRACK
2930        PLA          TRACK WE WANT
2940        DEC SEARCH.COUNT
2950        BNE .4       TRY AGAIN
2960        BEQ .3       TRY TO RE-CALIBRATE AGAIN
2970  *---------------------------------
2980  *      DRIVE ERROR, CANNOT FIND TRACK
2990  *---------------------------------
3000  .6    PLA          REMOVE CURRENT.TRACK
3010        LDA #ERR.BAD.DRIVE
3020  .7    PLP
3030        JMP ERROR.HANDLER
3040  *---------------------------------
3050  *      NULL COMMAND, ON THE WAY OUT....
3060  *---------------------------------
3070  .8    BEQ RWTS.EXIT
3080  *---------------------------------
3090  *      FORMAT COMMAND
3100  *---------------------------------
```

```
.9     JMP FORMAT
*--------------------------------
*      READ OR WRITE COMMAND
*--------------------------------
.10    LDY #3       GET VOLUME# WANTED
       LDA (IOB.PNTR),Y
       PHA          SAVE DESIRED VOLUME# ON STACK
       LDA VOLUME
       LDY #14      STORE ACTUAL VOLUME NUMBER FOUND
       STA (IOB.PNTR),Y
       PLA          GET DESIRED VOLUME# AGAIN
       BEQ .11      IF =0, DON'T CARE
       CMP VOLUME   SEE IF RIGHT VOLUME
       BEQ .11      YES
       LDA #ERR.WRONG.VOLUME
       BNE .7       UH OH!
*--------------------------------
.11    LDY #5       GET SECTOR# WANTED
       LDA (IOB.PNTR),Y   (LOGICAL SECTOR NUMBER)
       TAY          INDEX INTO PHYSICAL SECTOR VECTOR
       LDA PHYSICAL.SECTOR.VECTOR,Y
       CMP SECTOR
       BNE .21      NOT THE RIGHT SECTOR
       PLP          GET COMMAND FLAG AGAIN
       BCC WRITE
       JSR READ.SECTOR
       PHP          SAVE RESULT; IF BAD, WILL BE COMMAND
       BCS .21      BAD READ
       PLP          THROW AWAY
       LDX #0
       STX $26
       JSR POST.NYBBLE
       LDX SLOT
RWTS.EXIT
       CLC
       .HS 24       "BIT" TO SKIP NEXT INSTRUCTION
*--------------------------------
ERROR.HANDLER
       SEC          INDICATE AN ERROR
       LDY #13      STORE ERROR CODE
       STA (IOB.PNTR),Y
       LDA MOTOR.OFF,X
       RTS
*--------------------------------
WRITE  JSR WRITE.SECTOR
       BCC RWTS.EXIT
       LDA #ERR.WRITE.PROTECT
       BCS ERROR.HANDLER   ...ALWAYS
*--------------------------------
*      SEEK TRACK SUBROUTINE
*      (A) = TRACK# TO SEEK
*      (DRIVE.NO) IS NEGATIVE IF DRIVE 1
*                AND POSITIVE IF DRIVE 2
*--------------------------------
```

```
3650  SEEK.TRACK
3660        PHA          SAVE TRACK#
3670        LDY #1       CHECK DEVICE CHARACTERISTICS TABLE
3680        LDA (DCT.PNTR),Y   FOR TYPE OF DISK
3690        ROR          SET CARRY IF TWO PHASES PER TRACK
3700        PLA          GET TRACK# AGAIN
3710        BCC .1       ONE PHASE PER TRACK
3720        ASL          TWO PHASES PER TRACK, SO DOUBLE IT
3730        JSR .1       FIND THE TRACK
3740        LSR CURRENT.TRACK  DIVIDE IT BACK DOWN
3750        RTS
3760  *--------------------------------
3770  .1    STA TRACK
3780        JSR GET.SLOT.IN.Y
3790        LDA DRIVE.1.TRACK,Y
3800        BIT DRIVE.NO   WHICH DRIVE?
3810        BMI .2       DRIVE 1
3820        LDA DRIVE.2.TRACK,Y
3830  .2    STA CURRENT.TRACK   WHERE WE ARE RIGHT NOW
3840        LDA TRACK    WHERE WE WANT TO BE
3850        BIT DRIVE.NO WHICH DRIVE?
3860        BMI .3       DRIVE 1
3870        STA DRIVE.2.TRACK,Y   DRIVE 2
3880        BPL .4       ...ALWAYS
3890  .3    STA DRIVE.1.TRACK,Y
3900  .4    JMP SEEK.TRACK.ABSOLUTE
3910  *--------------------------------
3920  *     CONVERT SLOT*16 TO SLOT IN Y-REG
3930  *--------------------------------
3940  GET.SLOT.IN.Y
3950        TXA          SLOT*16 FROM X-REG
3960        LSR
3970        LSR
3980        LSR
3990        LSR
4000        TAY          SLOT INTO Y
4010        RTS
4020  *--------------------------------
4030  *     SET UP CURRENT TRACK LOCATION
4040  *     IN DRIVE.1.TRACK OR DRIVE.2.TRACK VECTORS,
4050  *     INDEXED BY SLOT NUMBER.
4060  *
4070  *     (A) = TRACK# TO BE SET UP
4080  *--------------------------------
4090  SETUP.TRACK
4100        PHA          SAVE TRACK # WE WANT TO SET UP
4110        LDY #2       GET DRIVE NUMBER FROM IOB
4120        LDA (IOB.PNTR),Y
4130        ROR          SET CARRY IF DRIVE 1, CLEAR IF 2
4140        ROR DRIVE.NO MAKE NEGATIVE IF 1, POSITIVE IF 2
4150        JSR GET.SLOT.IN.Y
4160        PLA          GET TRACK #
4170        ASL          DOUBLE IT
4180        BIT DRIVE.NO  WHICH DRIVE?
```

```
4190        BMI .1       DRIVE 1
4200        STA DRIVE.2.TRACK,Y
4210        BPL .2       ...ALWAYS
4220 .1     STA DRIVE.1.TRACK,Y
4230 .2     RTS
4240 *---------------------------------
4250 FORMAT
4260 *---------------------------------
4270        .BS $BFB8-*
4280 PHYSICAL.SECTOR.VECTOR
4290        .HS 000D0B09070503010E0C0A080604020F
```

```
=====================================================================
DOCUMENT :AAL-8109:DOS3.3:S.FldInputRtn.txt
=====================================================================

 1000  *--------------------------------
 1010  *      FIELD INPUT SUBROUTINE
 1020  *      --------------------
 1030  *      BY ROBERT W. POTTS
 1040  *      BANK OF LOUISVILLE
 1050  *      P. O. BOX 1101
 1060  *      LOUISVILLE, KY 40201
 1070  *
 1080  *      MODIFIED BY BOB SANDER-CEDERLOF
 1090  *      FOR THE "APPLE ASSEMBLY LINE"
 1100  *---------------------------------
 1110         .OR $300
 1120  *--------------------------------
 1130  MON.CH      .EQ $24      MONITOR HORIZONTAL
 1140  MON.BASL    .EQ $28
 1150  SPC.PNTR    .EQ $71,72
 1160  STR.PNTR    .EQ $83,84
 1170  *---------------------------------
 1180  CT          .EQ $E1      CHARACTER COUNT
 1190  FL          .EQ $E7      FIELD LENGTH  (SET BY "SCALE=FL")
 1200  FLDCOD      .EQ $F9      FIELD CODE    (SET BY "ROT=FC")
 1210  EXITCODE    .EQ $E0      PEEK (224) TO SEE EXIT CODE
 1220  *---------------------------------
 1230  INPUT.BUFFER .EQ $0200
 1240  AMPER.VECTOR .EQ $03F5
 1250  *---------------------------------
 1260  MON.RDKEY   .EQ $FD0C    MONITOR CHAR INPUT
 1270  MON.COUT    .EQ $FDED
 1280  MON.BS      .EQ $FC10    MONITOR BACKSPACE
 1290  *---------------------------------
 1300  AS.CHKSTR   .EQ $DD6C
 1310  AS.SYNCHR   .EQ $DEC0
 1320  AS.PTRGET   .EQ $DFE3
 1330  AS.GETSPA   .EQ $E452
 1340  AS.MOVSTR   .EQ $E5E2
 1350  *---------------------------------
 1360  *  SET UP AMPERSAND VECTOR
 1370  *--------------------------------
 1380  SETUP  LDA #$4C     JMP OPCODE
 1390         STA AMPER.VECTOR
 1400         LDA #INPUT.FIELD
 1410         STA AMPER.VECTOR+1
 1420         LDA /INPUT.FIELD
 1430         STA AMPER.VECTOR+2
 1440         RTS
 1450  *---------------------------------
 1460  INPUT.FIELD
 1470         LDA #$84     "INPUT" TOKEN
 1480         JSR AS.SYNCHR  REQUIRE "INPUT" OR SYNTAX ERROR
```

```
       JSR AS.PTRGET  GET STRING VARIABLE
       JSR AS.CHKSTR  REQUIRE STRING OR MISMATCH
*---------------------------------
       LDA #0       ZERO OUT CHARACTER COUNT
       STA CT
.1     JSR MON.RDKEY    GET CHARACTER
.2     AND #$7F     APPLESOFT STYLE
       CMP #$06     CONTROL-F?
       BEQ .3       YES
       CMP #$08     BACKSPACE?
       BEQ .4       YES
       CMP #$0D     RETURN?
       BEQ .7       YES, END OF FIELD
       CMP #$15     RIGHT ARROW?
       BEQ .5       YES
       CMP #$1B     ESCAPE?
       BEQ .6       YES
       CMP #$20     SOME OTHER CONTROL CHARACTER?
       BCC .1       YES, IGNORE IT
       CMP #$5B     ACCEPTABLE PRINTING CHARACTER?
       BCS .1       NO, IGNORE IT
*--------------------------------
*  GOT PRINTING CHARACTER - STORE IT
*--------------------------------
       LDY CT       CHARACTER COUNTER
       STA INPUT.BUFFER,Y  STORE IN STRING
       ORA #$80     TURN ON HIGH BIT
       JSR MON.COUT PRINT CHARACTER
       INC CT       INCREMENT CHARACTER COUNT
       LDA CT
       CMP FL       IS FIELD FILLED UP?
       BNE .1       NO, GET ANOTHER CHARACTER
       BEQ .7       ...ALWAYS
*---------------------------------
*  HANDLE CONTROL-F
*---------------------------------
.3     LDA CT       ON FIRST CHARACTER?
       BNE .1       NO, GET ANOTHER CHARACTER
       LDA #2       EXIT CODE = 2
       BNE .8       ...ALWAYS
*---------------------------------
*  HANDLE BACKSPACE
*---------------------------------
.4     LDA #3       EXIT CODE = 3 IF IN 1ST CHAR
       DEC CT       DECREMENT CHARACTER COUNTER
       BMI .8       ON FIRST POSITION
       JSR MON.BS   BACKSPACE
       LDA #$DF     UNDERLINE
       JSR MON.COUT PRINT IT
       JSR MON.BS   BACKSPACE AGAIN
       JMP .1       DO AGAIN
*---------------------------------
*  HANDLE RIGHT ARROW
*---------------------------------
```

```
2030 .5     LDY MON.CH   YES, GET NEXT CHARACTER FROM SCREEN
2040        LDA (MON.BASL),Y
2050        JMP .2
2060 *--------------------------------
2070 *  HANDLE ESCAPE
2080 *--------------------------------
2090 .6     LDA FLDCOD   FIELD CODE = 0?
2100        BEQ .1       YES, GET ANOTHER CHARACTER
2110        LDA CT
2120        BNE .1       NO, GET ANOTHER CHARACTER
2130        LDA #1       EXIT CODE = 1
2140        BNE .8       ...ALWAYS
2150 *--------------------------------
2160 *  STORE THE INPUT DATA IN THE STRING
2170 *--------------------------------
2180 .7     LDA CT       STRING LENGTH
2190        JSR AS.GETSPA  GET SPACE IN STRING AREA
2200        LDY #0       MOVE DATA INTO VARIABLE
2210        STA (STR.PNTR),Y  LENGTH
2220        LDA SPC.PNTR
2230        INY
2240        STA (STR.PNTR),Y  LO-BYTE OF ADDRESS
2250        LDA SPC.PNTR+1
2260        INY
2270        STA (STR.PNTR),Y  HI-BYTE OF ADDRESS
2280        LDX #INPUT.BUFFER
2290        LDY /INPUT.BUFFER
2300        LDA CT       LENGTH
2310        JSR AS.MOVSTR
2320        LDA #0       EXIT CODE = 0
2330 .8     STA EXITCODE
2340        RTS
```

```
==============================================================================
DOCUMENT :AAL-8109:DOS3.3:S.US.DIRECTIVE.txt
==============================================================================

 1000        .OR $F00
 1010        .TF B.US.DIRECTIVE
 1020  *--------------------------------
 1030  WBUF   .EQ $0200
 1040  DLIM   .EQ $DA
 1050  HIBIT  .EQ $04
 1060  *--------------------------------
 1070  *  THE FOLLOWING VALUES ARE FOR VERSION 4.0
 1080  *  OF S-C ASSEMBLER II (DISK)
 1090  *--------------------------------
 1100  GNNB   .EQ $1283    GET NEXT NON-BLANK CHAR
 1110  GNC    .EQ $128B    GET NEXT CHAR
 1120  CMNT   .EQ $188E    FINISH THE LINE
 1130  ERBA   .EQ $1932    ERROR: BAD ADDRESS
 1140  EMIT   .EQ $19FA    EMIT A BYTE OF OBJECT CODE
 1150  *--------------------------------
 1160  ACTIVATE.US
 1170         LDA #DIR.US  STORE ADDRESS IN .US VECTOR
 1180         STA $100D    INSIDE S-C ASSEMBLER II VER
 1190         LDA /DIR.US  DISK VERSION 4.0
 1200         STA $100E
 1210         RTS
 1220  *--------------------------------
 1230  DIR.US
 1240         LDY #0       START WITH HI-BIT EQUAL TO ZERO
 1250  .1     STY HIBIT    SET HI-BIT ZERO OR ONE
 1260         JSR GNNB     GET NEXT NON-BLANK AFTER OPCODE
 1270         BCS ERBA2    END OF LINE IS BAD NEWS
 1280         LDY #$80     IN CASE WE NEED HI-BIT OF ONE
 1290         CMP #$2D     CHECK FOR MINUS SIGN
 1300         BEQ .1       YES, WE NEED HI-BIT OF ONE
 1310         STA DLIM     NOT MINUS, MUST BE DELIMITER
 1320         JSR GNC      GET NEXT CHARACTER
 1330         BCS ERBA2    END OF LINE IS BAD NEWS
 1340         CMP DLIM     SEE IF DELIMITER ALREADY
 1350         BEQ .4       YES, NO STRING IN BETWEEN
 1360  .2     JSR GNC      GET NEXT CHARACTER
 1370         BCS ERBA2    END OF LINE IS BAD NEWS
 1380         CMP DLIM     SEE IF DELIMITER YET
 1390         BEQ .3       YES, FINISH UP AND RETURN
 1400         LDA WBUF-2,Y NO, GET PREVIOUS CHAR
 1410         ORA HIBIT    MERGE WITH SELECTED HI-BIT
 1420         JSR EMIT     EMIT THE OBJECT CODE BYTE
 1430         JMP .2       GO FOR ANOTHER ONE
 1440  *--------------------------------
 1450  .3     LDA WBUF-2,Y GET PREVIOUS CHAR
 1460         ORA HIBIT    MERGE WITH SELECTED HI-BIT
 1470         EOR #$80     TOGGLE HI-BIT SINCE LAST CHAR
 1480         JSR EMIT     EMIT THE OBJECT CODE BYTE
```

```
1490 .4     JMP CMNT     FINISH PROCESSING THE LINE
1500 *--------------------------------
1510 ERBA2  JMP ERBA     BAD ADDRESS ERROR
```

================================================================================
DOCUMENT :AAL-8109:DOS3.3:Tst.Fld.Inp.Rtn.txt
================================================================================

```
(4)"BRUN B.INPUT ROUTINE"AÜV(5),H(5),L(5),T$(5),A$(5)d-ÅI–
1¡5:áV(I),H(I),L(I),T$(I):Çu(É5,7,30,NAMEÑ2É7,7,3,AGEó<É7,27,3,WEIGHT©
FÉ9,7,12,STATEπPÉ9,27,5,ZIPÌZâ:ó:¢23:û:∫" TYPE CTRL-F WHEN FORM
FINISHED ":ù0  dÅI–1¡5:¢V(I):ñH(I)…„(T$(I))…1:∫T$(I)" ";:ÅJ –
1¡L(I):∫Á(95);:Ç:Ç8  nI–1Q     x¢V(I):ñH(I):ôL(I):ò0g    ÇØÑA$(I):XC–
,(224)Å   å¥XC»1´200,300,400,500î   »I–I»1:≠Iœ5ƒ110ù     “´120´
     ,Ä:≤ESCAPE∫     ê≤CONTROL-F“    öó:ÅI–1¡5:∫A$(I):Ç:Ä·
     Ù≤BACKSPACEÙ    ¸I–I…1:≠I–0ƒI–5˝     ´120
```

```
==============================================================================
DOCUMENT :AAL-8110:Articles:DOS3.3Disasm.txt
==============================================================================
```

DOS Disassembly:  $B052-B0B5 AND $B35F-B7FF

Everything from $B800 through $BFFF has now been covered in previous issues of AAL.  Also, the 3.3 boot ROM was covered in the August issue.  In this issue I present the rest of the boot code and part of the File Manager (FM).

Lines 1000-1570 are a subroutine inside FM which calls RWTS.  The main entry at line 1170 assumes (A)=opcode, (X)=track, and (Y)=sector.  A subsidiary entry at line 1200 assumes (A)=opcode, and track and sector were already set up.  The valid opcodes are SEEK=0, READ=1, WRITE=2, and FORMAT=4.

Lines 1580-1970 are the various exits from FM.  Upon exit, (A)=error code and CARRY status is set if there was an error, clear if not.

Lines 1980-2560 are various buffers, constants, and variables for FM. Notice there are some apparently unused bytes in this area.

Lines 2570-3690 are what is written on track 0 sector 0.  It loads and executes BOOT.STAGE1 at $0800 (execution starts at $0801).  This code reads in RWTS and BOOT.STAGE2.  Since most of this area was unused, patches to solve the APPEND problem are here (lines 3020-3640).

Lines 3700-4080 are BOOT.STAGE2, which read in the rest of DOS and jump to $9D84.

Routines to write the DOS image on tracks 0-2, to enter RWTS with interrupts disabled, and to clear a 256-byte buffer are in lines 4090-4990.

Lines 5100-5300 are the IOB and DCT used by FM for all calls to RWTS. The contents of these are described in the DOS Reference Manual pages 95-98.

=============================================================================
DOCUMENT :AAL-8110:Articles:Errata.CHRGET.txt
=============================================================================

Errata

Volume 1, Issue 12 (Sep 1981) page 8:  Line 1120 in the CHRGET/CHRGOT subroutine should be BCS instead of BEQ.

Volume 1, Issue 7 (Apr 1981) page 8:  Insert the following lines:

```
1331        TXA          LINE LENGTH
1332        TAY          IN Y-REG FOR LOOP COUNT
1333 .2     LDA $200,Y   STRIP SIGN-BITS FROM EACH BYTE
1334        AND #$7F
1335        STA $200,Y
1336        DEY
1337        BPL .2
```

This patch is necessary because characters Applesoft strings are supposed to have the sign-bit clear.  Everything is fine unless you try compare input strings with constant strings.

Another Way Out of the Assembler

James Church, from Trumbull, CT, writes that he has found a way to get from the assembler into Applesoft, without wiping out an Applesoft program.

The normal way to leave is by typing FP, and then PR#0.  This of course clears any Applesoft program from memory.  But by typing $AAB6:40, $E003G, and PR#0 you can enter Applesoft softly.

```
===============================================================================
DOCUMENT :AAL-8110:Articles:Front.Page.txt
===============================================================================
```

$1.20

Volume 2 -- Issue 1                                 October, 1981

In This Issue...

```
==============================================================================
DOCUMENT :AAL-8110:Articles:GRAM.1lineprint.txt
==============================================================================
```

Screen Printers in One Line....................Bob Sander-Cederlof

When you are writing a fancy program to help you in your business, you spend a lot of time formatting the screen output. You want it to look perfect!

But how do you get it from the screen to your printer? You might end up re-writing the whole output routine, or incorporating a machine language screen dump. You don't have to go to such extremes, because you can copy the contents of the screen to your printer with as little as one line of Applesoft code!

Here is one such line, printed with one statement per physical line for easy reading by humans. (But it is still only one line to Applesoft.)

```
100 PR# 1
    : FOR V = 1 TO 24
    :   FOR H = 1 TO 40
    :     VTAB V
    :     HTAB H
    :     VH = PEEK(40)+PEEK(41)*256+H-1
    :     CH = PEEK (CH)
    :     PRINT CHR$(CH+32*(CH<32));
    :   NEXT
    :   PRINT
    : NEXT
    : PR# 0
    : CALL 1002
    : RETURN
```

Note that the RETURN on the end means I am expecting you to call this with a GOSUB 100. If you want to, you can put this line right where you need it as in-line code, and leave off the RETURN.

How does it work? First, PR#1 turns on your printer. It also unhooks DOS, but we don't need DOS right now anyway. We will rehook it at the end. Look to the last few lines now: PR#0 turns off your printer, and CALL 1002 re-hooks DOS.

The first FOR loop covers the 24 lines of the screen. The second FOR loop covers the 40 characters of each line. VTAB V and HTAB H position the cursor over the next character on the screen to be copied to your printer. VTAB V also sets up locations 40 and 41 with the actual memory address of the first character on line V. The "VH =" statement computes the memory address of character H on line V.

CH=PEEK(VH) will retrieve the character under the cursor. PRINT CHR$(CH+32*(CH<32)) prints that character on your printer. It also prints it on the screen, but so what? Since we are printing the same character on top of itself, nothing changes. That is, unless the character was INVERSE mode. Inverse mode characters are converted to FLASH mode on the screen, and both of those modes are printed as NORMAL mode on the printer. (Note that the expression 32*(CH<32) equals 0 if CH is greater than 31, and equals 32 if CH is less than 32.)

NEXT : PRINT : NEXT moves us to the next character until the end of line, then prints a carriage return on the printer, and then moves us to the next line on the screen. After all the lines have been printed, the printer is unhooked, DOS rehooked, and the subroutine returns.

Here is another version, which computes its own screen addresses in a series of three nested FOR loops:

```
100 PR# 1
    : PRINT CHR$ (9)"80N"
    : FOR I = 0 TO 80 STEP 40
    :   FOR J = I+1024 TO I+1920 STEP 128
    :     FOR K = J TO J+39
    :       CH = PEEK(K)
    :       PRINT CHR$(CH+32*(CH<32));
    :     NEXT
    :     PRINT
    :   NEXT
    : NEXT
    : PR# 0
    : CALL 1002
    : RETURN
```

Notice I had to print a control-I and "80N" to the printer interface to turn off the screen echo in this version.

It seemed to me that this second version ran a little faster than the first one, although I didn't use a stopwatch. Both versions keep ahead of the printer anyway.

=====================================================================
DOCUMENT :AAL-8110:Articles:Gram.Book.Revws.txt
=====================================================================

Mini-Review: "Real Time Programming"........Bob Sander-Cederlof

The other night at B. Dalton's (the book store you find in almost every shopping mall in America), I came across a new book you might like: "Real Time Programming--Neglected Topics", by Caxton C. Foster, Addison-Wesley Publishing Company, 1981, 190 pages, $8.95.

Are you serious about learning to program in assembly language? Even to the point of learning how to interface your Apple to other devices? Foster introduces such topics as interrupt processing, switch debouncing, timers, synchronizing processes, digital filtering, adaptive control loops, and network communication.

If you are still here, after all those frightening buzzwords, good! Fear not! The book was written for ordinary mortals like you and me, not mathematical wizards. Each topic is amply illustrated with working programs, and actual hardware experiments you can set up with your own computer. I found that I could actually read the book, without stumbling and going over and over the same passage to understand it. Foster has made some very complex techniques comprehensible. There are lots of interesting analogies and drawings to aid in understanding.

The examples are written in the assembly language of a mythical machine called FOSSOL. A simple chart on page 3 shows the correspondence between these opcodes and those of the 6502 in your Apple. Most of them are identical to the 6502 opcodes. The same chart shows how to translate FOSSOL into Z-80 assembly language. (Why in the world would anyone want to do that?!!)

If you are not quite ready to sink your teeth into this one, you might look over his previous book, "Programming a Microcomputer: 6502". It was published about 3 1/2 years ago, by the same publishers, and is still available.

More New Publications........................Bob Sander-Cederlof

MICRO Magazine has collected together another series of their best Apple-related articles, called "MICRO-Apple 2". If you missed "MICRO-Apple 1", it is still available too. Each book comes complete with a disk containing all the programs printed inside, is 224 pages long, and costs $24.95 (plus $2.00 shipping charges).

MICRO has also recently published an atlas to all the interesting locations inside your Apple, called "What's Where in the APPLE?". It is 128 pages long, 8 1/2 by 11 inches, wire-circle bound to lie flat on your desk without fighting. It retails for $14.95, plus $2.00 shipping charges. Call 1-800-227-1617, extension 564, if you can't wait. I will have some at the next Apple Corps meeting at a slightly lower price.

If you have been reading your SOFTALK magazine each month, you probably have noticed Roger Wagner's very helpful column. "Assembly Lines" guides you each month through the a-MAZE-ing and mystifying world of assembly language programming. The articles have been so popular that SOFTALK has collected the first 12 into a book called "Everyone's Guide to Assembly Language". They have added some new material not yet printed in the magazine. It costs $19.95 plus $1.50 for shipping charges. Write to Softalk Book, 11021 Magnolia Blvd, North Hollywood, CA 91601.

"Graphics Software for Microcomputers", by B.J. Korites, will show you how to write your own graphics software. Not just simple lines and shapes, but 3-dimensional drawing with interactive input, rotations, translations, perspective transformations, scaling, clipping, shading, and more. Program listings written in Applesoft Basic are presented side-by-side the theoretical explanations. The book costs $19.95 and the 61 programs are available on disk for an additional $18.95. Once again, add $2.00 per item for shipping charges! Write to Kern Publications, 190 Duck Hill Road, P. O. Box 1029, Duxbury, MA 02332. Or call (617) 934-0445. I am trying to get a dozen of these for the next meeting, but I can't promise anything yet.

==============================================================================
DOCUMENT :AAL-8110:Articles:GRAM.Hello.AS.txt
==============================================================================

HELLO vs. LANGUAGE NOT AVAILABLE

When you are trying to send out a disk full of software to many Apple owners, you face a lot of problems trying to be compatible with every configuration.  One of those problems is the HELLO program.

If you write it in Applesoft, and the customer only has Integer BASIC (or vice versa), the message "LANGUAGE NOT AVAILABLE" will print out when he boots your disk.  There are several ways around this problem. If you get a license from Apple, you can include RAM Applesoft on your disk.  Or, you could require that he boot another disk first; you could warn him to ignore the error message.  You could tell customers to delete the HELLO program which is in the language he doesn't have, and rename the other one to correspond to your boot file name.

Or, you could do this:  write the primary boot program in Applesoft, and include an Integer BASIC version named "APPLESOFT".  I have done this on the S-C Assembler II Version 4.0 disks.  However, just yesterday, I discovered a problem with my Integer BASIC version.  It just hangs up!  It seems that DOS really isn't completely satisfied to just run my program named APPLESOFT.  It also wants to configure some internal addresses for RAM Applesoft and then try to RUN the Applesoft boot program.  By inserting two POKEs in my program named APPLESOFT, I can fool DOS completely.  Here they are:

```
     POKE -21935,0 : POKE -21918,0
```

These POKEs change the mode inside DOS so that the boot process is cut short.  The first one is $AA51; it had $C0 (or 192) in it before the POKE.  $C0 means a coldstart is in progress, and RAM Applesoft is in control.  We turn both of those off by POKEing 0.  The second POKE is at $AA62, the index of a pending command.  This was a 6 before the POKE, indicating that a RUN was pending (for the original HELLO file). We turn that off also.

================================================================================
DOCUMENT :AAL-8110:Articles:Sifting.Primes.txt
================================================================================

Sifting Primes Faster and Faster

Benchmark programs are sometimes useful for selecting between various processors. Quite a few articles have been published which compare and rank the various Z-80, 8080, 6800, and 6502 systems based on the speed with which they execute a given BASIC program. Some of us cannot resist the impulse to show them up by recoding the benchmark in our favorite language on our favorite processor, using our favorite secret tricks for trimming microseconds.

"A High-Level Language Benchmark" (by Jim Gilbreath, BYTE, September, 1981, pages 180-198) is just such an article. Jim compared execution time in Assembly, Forth, Basic, Fortran, COBOL, PL/I, C, and other languages; he used all sorts of computers, including the above four, the Motorola 68000, the DEC PDP 11/70, and more. He used a short program which finds the 1899 primes between 3 and 16384 by means of a sifting algorithm (Sieve of Eratosthenes).

His article includes table after table of comparisons. Some of the key items of interest to me were:

| Language and Machine | Seconds |
|---|---|
| Assembly Language 68000 (8 MHz) | 1.12 |
| Assembly Language Z80 | 6.80 |
| Digital Research PL/I (Z80) | 14.0 |
| Microsoft BASIC Compiler (Z80) | 18.6 |
| FORTH 6502 | 265. |
| Apple UCSD Pascal | 516. |
| Apple Integer BASIC | 2320. |
| Applesoft BASIC | 2806. |
| Microsoft COBOL Version 2.2 (Z80) | 5115. |

There is a HUGE error in the data above; I don't know if it is the only one or not. The time I measured for the Apple Integer BASIC version was only 188 seconds, not 2320 seconds! How could he be so far off? His data is obviously wrong, because Integer BASIC in his data is too close to the same speed as Applesoft.

I also don't know why they neglected to show what the 6502 could do with an assembly language version. Or maybe I do....were they ashamed?

William Robert Savoie, an Apple owner from Tennessee, sent me a copy of the article along with his program. He "hand-compiled" the BASIC version of the benchmark program, with no special tricks at all. His program runs in only 1.39 seconds! That is almost as fast as the 8 MHz Motorola 68000 system! The letter that accompanied his program challenged anyone to try to speed up his program.

How could I pass up a challenge like that? I wrote my own version of the program, and cut the time to .93 seconds! Then I made one small change to the algorithm, and produced exactly the same results in only .74 seconds!

Looking back at Jim Gilbreath's article, he concludes that efficient, powerful high-level languages are THE way to go. He eschews the use of assembly language for any except the most drastic requirements, because he could not see a clear speed advantage. He points out the moral that a better algorithm is superior to a faster CPU. (Note that his algorithm is by no means the fastest one, by the way.)

Here is Gilbreath's algorithm, in Integer BASIC:

<program#1>

The REM tagged onto the end of line 70, if changed to a real PRINT statement, will print the list of prime numbers as they are generated. Of course printing them was not included in any of the time measurements. According to my timing, printing adds 12 seconds to the program.

I modified the algorithm to take advantage of some more prior knowledge about sifting: There is no need to go through the loop in lines 50 and 60 if P is greater than 127 (the largest prime no bigger than the square root of 16384). This means changing line 40 to read:

```
     40 P=I+I+3 : IF P>130 THEN 70 : K=I+P
```

This change cut the time for the program from 188 seconds to 156 seconds. My assembly language version of the original algorithm ran in .93 seconds, or 202 times faster; the better algorithm ran in .74 seconds, or almost 211 times faster.

William Savoie has done a magnificent job in hand-compiling the first program. He ran the program 100 times in a loop, so that he could get an accurate time using his Timex watch. Here is the listing of his program.

<Bill Savoie's program>

Here is a listing of my fastest version. If you delete lines .... through ...., you get my code for the original algorithm.

<my program>

Michael R. Laumer, of Carrollton, Texas, has been working for about a year on a full-scale compiler for the Integer BASIC language. He has it nearly finished now, so just for fun he used it to compile the algorithm from Gilbreath's article. Mike used a slightly different form of the Integer BASIC program than I did, which took 238 seconds to execute. But the compiled version ran in only 20 seconds! If you are interested in compiling Integer BASIC programs, you can write to Mike at Laumer Research, 1832 School Road, Carrollton, TX 75006.

If you want to, you can easily cut the time of my program from .74 to about .69 seconds. Lines 1600-1650 in my program set each byte in ARRAY to $01. If I don't mind the extra program length, I can rewrite this loop to run in about 42 milliseconds instead of the over 90 it now takes. Here is how I would do it:

```
.1      STA ARRAY,Y
        STA ARRAY+$100,Y
        STA ARRAY+$200,Y
        STA ARRAY+$300,Y          TOTAL OF 32
         .                       LINES LIKE THESE
         .
         .
        STA ARRAY+$1E00,Y
        STA ARRAY+$1F00,Y
        INY
        BNE .1
```

If you can find a way to implement the same program in less than .69 seconds, you are hereby challenged to do so!

=============================================================================
DOCUMENT :AAL-8110:Articles:XAsm.6809.txt
=============================================================================

## 6809 Cross Assembler

Chris Wiggs, of Rockford, IL, has developed a cross assembler for the 6809 which runs in the Apple. In fact, it is really a set of patches to the S-C Assembler II Version 4.0. If you BLOAD your copy of the assembler, and then BRUN his patch file, and BSAVE the result, you have a brand new assembler for 6809 code.

It is set up to work with "The Mill". Typing MGO turns on the mill and starts 6809 code executing, while the Apple's 6502 is left in a waiting loop.

Chris has authorized me to distribute these patches. For only $20 you will get a disk which includes all of the source code for the patches (in S-C Assembler II Version 4.0 format), the already-assembled patch file, a sample 6809 program, and some instructions (in the form of an assembly source file of comments).

I have not put this program through any rigorous test, but Chris is using it himself and is satisfied that it is working correctly. Anyway, you will actually have the SOURCE code, so you can make any further changes you wish with ease.

You might also study how he did it, and then write a cross assembler for some other chip, such as Z-80, 68000, 1802, TMS7000, or whatever.

Here is a sample 6809 assembly:

<<<code here>>>

## Source Code for S-C Assembler II Version 4.0

At long last, I have decided to start selling the source code for my assembler. So many of you have asked for it! I am sure you understand my reluctance; after all, with a wife and five kids to support, and most of our income coming from this one product....

If I have your registration card for Version 4.0 on file, or some other proof-of-purchase, I will send you a disk with all of the commented source code on it. You can study it, assemble it, modify it, et cetera; just don't start selling it! With your check for $95, you will need to include the following signed declaration:

"I am purchasing the source code of S-C Assembler II Version 4.0 with the understanding that it is proprietary information belonging to S-C SOFTWARE. The disk, and any copies or listings I may make of it, are only for my own personal use."

==============================================================================
DOCUMENT :AAL-8110:Articles:Xtnd.Apples.Mtr.txt
==============================================================================

Extending the Apple Monitor

Just as the creators of Applesoft included the wonderful "&" statement to allow language extensions, so also Steve Wozniak included a means for adding new monitor commands. The "control-Y" command branches to a user-defined maching language routine, which can supplement the existing commands in the Monitor ROM.

The control-Y command executes your subroutine starting at $3F8. All there is room for at $3F8 is a JMP to where your subroutine is REALLY stored. When you boot DOS, a JMP $FF65 instruction is inserted at $3F8, setting the control-Y command to merely re-enter the monitor. By changing the address of that JMP instruction, you can have it jump to your own code. If you look ahead at the listing of MONITOR EXTENSIONS, lines 1170-1210 store the address of my CTRLY subroutine into the JMP instruction.

I have thought of at least three features that I miss all the time in the monitor. (I just now thought of several more, but they will have to wait for another article.)

1. The monitor already includes the ability to add and subtract single-byte values, and print the single-byte result. I would like to be able to do this with 16-bit values.

2. The monitor can already dump memory in hexadecimal, but I want to see it as ASCII characters also. There is room on the screen for both at once.

3. The monitor can already disassemble code to the screen, 20 lines at a time. If I want more than 20 lines, I can type "LLLLLL", one L for each 20 lines. But I would like to be able to just specify the beginning and ending addresses for the disassembly, like I do for the hexadecimal printout.

If you enter the MONITOR EXTENSIONS program, these three functions will be added to the monitor. To add or subtract two values, type the two values separated by "+" or "-"; then type control-Y, and carriage return. To dump in combined hex and ASCII, type the beginning and ending addresses separated by a period, then control-Y and carriage return. To disassemble a range of memory, type the beginning and ending addresses separated by a period, then control-Y, "L", and a carriage return.

Looking again at the listing, lines 1230-1340 figure out which of the above command options you have typed in. When the monitor branches to $3F8, the following conditions have been set up:

(A) = 0 if only one address was typed;

```
    = code for separator character if two addresses
      were typed.

(X) = 0 if no hex digit typed immediately before the
        control-Y;
    = 1 if any hex digits immediately before the
        control-Y.

(Y) = 0

($34) = index into input buffer of next character after
        the control-Y.
```

Up to five 16-bit variables (called A1, A2, A3, A4, and A5) are filled from the hexadecimal values in the command. If you type a "<" after the first value, then that value will be stored in A4 and A5 (A4 is at $42,43; A5 at $44,45). If you type a ".", "+", "-", or ":" after a hexadecimal value, then that value will be stored in A1 and A3 (A1 is at $3C,3D; A3 at $40,41). If you type a hexadecimal value immediately before the control-Y, then that value will be stored in A2 (which is at $3E,3F).

Looking again at lines 1230-1340, I branch to SUB if the separator is "-", or ADD if it is "+". If the separator is a colon, I just return; I don't have any control-Y command which accepts a colon separator. If the separator is not any of the above, then either there was no separator, or it was a period. In both of these cases, I want to dump memory. If the character after the control-Y is not "L", then I want a combined hex-ASCII dump; if it is "L", I want disassembly. Line 1340 increments the buffer pointer so that the "L" command will not be re-executed by the regular monitor routine after my control-Y routine is finished.

Lines 1360-1450 control the disassembly option. I used a monitor subroutine to copy the beginning address from A1 into PC. Then I wrote a loop that calles the monitor routine to disassemble one line, and then checks to see if we have reached the ending address. Compare this to the code in the monitor ROM at $FE5E through $FE74. There is one trick in this code. I wanted to compare PC to END.ADDR, and continue if PC was less than or equal to END.ADDR. The normal comparison technique would either SET carry at line 1390, but I CLEARed it. This has the same affect as using one less than the value in PC as the first comparand. I needed this, because BCC at line 1440 only branches if the first comparand is LESS THAN the second one. In other words, since it is difficult to implement IF PC <= END.ADDR THEN ..., I implemented IF PC-1 < END.ADDR THEN ....

Lines 1470-1780 perform the combined hex-ASCII dump. I must give credit to Hugh McKinney, of Dunwoody, GA, for some of the ideas in this code. Just for fun, I set it up to always print complete rows of eight bytes; the starting address is rounded down to the nearest multiple of 8, and the ending address is rounded up. This means that typing just one address will get you eight, also.

I had to make a judgment about what characters to display for the ASCII portion of the dump. There are 256 possible values, and only 96 printing characters. In fact, if you don't have a lower case adapter, your screen only shows 64 printing characters (unless you count inverse and flashing characters as different; in that case you have 192). I decided to display control characters (codes 00-1F and 80-9F) as flashing characters (codes 40-5F). Codes 60-7F and E0-FF display as lower case characters if you have a lower case adapter. Codes 20-5F and A0-DF display as normal video characters (the standard upper case set). If you want a different mapping, change lines 1660-1690 to do it your way.

Lines 1800-1930 perform the 16-bit addition and subtraction in the normal way. Lines 1940-1980 print out an equal sign, and the value.

If you get really ambitious, you might try programming for your Apple II Plus the S and T commands that Apple removed from the Autostart ROM. You can just about copy the code right out of the reference manual. You might also like to add a memory move command that will work correctly even when the target area overlaps the source area.

```
========================================================================
DOCUMENT :AAL-8110:DOS3.3:IB.Prime.Bench.txt
========================================================================

   10
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000
   20
U00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
   30
U00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
   40
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII3333333333333333333
   50  `SSSSSSSSSSSSSSSSSSSSSSSSSS$
F
   60  KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKrq
   70
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

   80  Y
   90  Q
```

```
==============================================================================
DOCUMENT :AAL-8110:DOS3.3:S.ASCII.Dump.P.txt
==============================================================================

 1000  *-----------------------------------
 1010  *      PATCHES TO ADD ASCII DUMP
 1020  *      TO THE APPLE MONITOR
 1030  *-----------------------------------
 1040  A1L        .EQ $3C
 1050  COUT       .EQ $FDED
 1060  *-----------------------------------
 1070         .OR $FDB8
 1080         .TA $0DB8
 1090         JSR PATCH    CALL MY PATCH CODE
 1100  *-----------------------------------
 1110         .OR $FCC9
 1120         .TA $0CC9
 1130  PATCH
 1140         JSR COUT     PRINT A SPACE
 1150         LDA (A1L),Y  GET BYTE TO BE DISPLAYED
 1160         PHA          SAVE IT ON STACK
 1170         LDA A1L      LOW BYTE OF DUMP ADDRESS
 1180         AND #7       MASK LINE POSITION
 1190         CLC
 1200         ADC #31      COMPUTE HORIZONTAL OFFSET
 1210         TAY
 1220         PLA          GET BYTE FROM STACK
 1230         STA ($28),Y  STORE IT ON THE SCREEN
 1240         LDY #0       RESTORE Y
 1250         RTS
```

```
==============================================================================
DOCUMENT :AAL-8110:DOS3.3:S.D33.B35F.B7FF.txt
==============================================================================

 1000  *---------------------------------
 1010  *      DOS 3.2.1/3.3 FILE MANAGER $B052-B0B5
 1020  *---------------------------------
 1030         .OR $B052
 1040         .TA $0852
 1050  *---------------------------------
 1060  MON.STATUS .EQ $48
 1070  IOB.ADDR   .EQ $AAC1
 1080  SAVE.FMW   .EQ $AE7E
 1090  RWTS       .EQ $BD00
 1100  MON.INIT   .EQ $FB2F
 1110  MON.HOME   .EQ $FC58
 1120  MON.PRBYTE .EQ $FDDA
 1130  MON.COUT   .EQ $FDED
 1140  MON.SETKBD .EQ $FE89
 1150  MON.SETVID .EQ $FE93
 1160  *---------------------------------
 1170  CALL.RWTS
 1180         STX IOB.TRACK
 1190         STY IOB.SECTOR
 1200  CALL.RWTS.1
 1210         STA IOB.OPCODE (SEEK=0, READ=1, WRITE=2, FORMAT=4)
 1220         CMP #2       OPCODE="WRITE"?
 1230         BNE .1
 1240         ORA FMW.FLAGS SET "LAST OP WAS WRITE" FLAG
 1250         STA FMW.FLAGS
 1260  .1     LDA FMW.VOLUME
 1270         EOR #$FF     UN-COMPLEMENT THE VOLUME #
 1280         STA IOB.VOLUME
 1290         LDA FMW.SLOT16  SLOT # TIMES 16
 1300         STA IOB.SLOT16
 1310         LDA FMW.DRIVE   DRIVE #
 1320         STA IOB.DRIVE
 1330         LDA FMW.SECTSZ  SECTOR LENGTH IN BYTES
 1340         STA IOB.SECTSZ
 1350         LDA FMW.SECTSZ+1
 1360         STA IOB.SECTSZ+1
 1370         LDA #1       SET TABLE TYPE
 1380         STA IOB.TYPE
 1390         LDY IOB.ADDR GET ADDRESS OF IOB
 1400         LDA IOB.ADDR+1
 1410         JSR ENTER.RWTS   PERFORM THE OPERATION
 1420         LDA IOB.ACTVOL   VOUME # FOUND
 1430         STA FMP.DATA+2
 1440         LDA #$FF     RESET VOLUME EXPECTED IN IOB
 1450         STA IOB.VOLUME
 1460         BCS .2       CARRY SET IF RWTS ERROR
 1470         RTS          RETURN TO CALLER
 1480  .2     LDA IOB.ERROR  GET ERROR CODE
```

```
1490        LDY #7       ERR=7 IF VOLUME MISMATCH
1500        CMP #$20     VOLUME MISMATCH?
1510        BEQ .3       YES
1520        LDY #4       ERR=4 IF WRITE PROTECTED
1530        CMP #$10     WRITE PROTECTED?
1540        BEQ .3       YES
1550        LDY #8       ERR=8 (I/O ERROR) FOR ALL OTHERS
1560 .3     TYA          ERR IN A-REG
1570        JMP FM.EXIT.ERROR
1580 *---------------------------------
1590 *      DOS 3.3 FILE MANAGER $B35F-B5FF
1600 *---------------------------------
1610               .OR $B35F
1620               .TA $0B5F
1630 FM.EXIT.ERR1  LDA #1 "LANGUAGE NOT AVAILABLE"
1640               BNE FM.EXIT.ERROR
1650 FM.EXIT.ERR2  LDA #2 "RANGE ERROR" (OPCODE)
1660               BNE FM.EXIT.ERROR
1670 FM.EXIT.ERR3  LDA #3 "RANGE ERROR" (SUBCODE)
1680               BNE FM.EXIT.ERROR
1690 FM.EXIT.ERR4  LDA #4 "WRITE PROTECTED"
1700               BNE FM.EXIT.ERROR
1710 FM.EXIT.ERR5  LDA #5 "END OF DATA"
1720               BNE FM.EXIT.ERROR
1730 FM.EXIT.ERR6  LDA #6 "FILE NOT FOUND"
1740               BNE FM.EXIT.ERROR
1750 FM.EXIT.ERR9  JMP $BFED "DISK FULL"
1760               NOP
1770 FM.EXIT.ERR10 LDA #10 "FILE LOCKED"
1780               BNE FM.EXIT.ERROR
1790 *---------------------------------
1800 FM.EXIT.GOOD
1810        LDA FMP.RETURN  GET RETURN CODE (ZERO)
1820        CLC          SIGNAL NO ERROR
1830        BCC FM.EXIT  ...ALWAYS
1840 *---------------------------------
1850 FM.EXIT.ERROR
1860        SEC
1870 *---------------------------------
1880 FM.EXIT
1890        PHP          SAVE STATUS ON STACK
1900        STA FMP.RETURN  RETURN CODE
1910        LDA #0       CLEAR MONITOR STATUS (JUST IN CASE)
1920        STA MON.STATUS
1930        JSR SAVE.FMW SAVE FM WORKAREA IN FILE BUFFER
1940        PLP          RETRIEVE STATUS FROM STACK
1950        LDX FMS.STACK RESTORE STACK POINTER
1960        TXS
1970        RTS          RETURN TO WHOEVER CALLED FM
1980 *---------------------------------
1990 *      SCRATCH AREA
2000 *---------------------------------
2010 FMS.TS.CD  .BS 2    T/S OF CURRENT DIRECTORY SECTOR
2020            .BS 2    ?
```

```
2030  FMS.STACK  .BS 1    S-REG WHEN FM CALLED
2040  FMS.DIRNDX .BS 1    VARIOUS USES
2050             .BS 1       "     "
2060             .BS 2    ?
2070             .HS 0000FFFF  USED BY INIT TO CLEAR VTOC ENTRY
2080  *---------------------------------
2090         .DA #1,#10,#100  DECIMAL CONVERSION TABLE
2100         .AS -/TIABSRAB/  FILE TYPE CODES
2110         .AS -/ EMULOV KSID/  MSG SPELLED BACKWARDS
2120  *---------------------------------
2130  *      VTOC SECTOR BUFFER
2140  *---------------------------------
2150         .BS 256
2160  *---------------------------------
2170  *      DIRECTORY SECTOR BUFFER
2180  *---------------------------------
2190         .BS 256
2200  *---------------------------------
2210  *      FILE MANAGER PARAMETERS
2220  *---------------------------------
2230  FMP.OPCODE    .BS 1
2240  FMP.SUBCOD    .BS 1
2250  FMP.DATA      .BS 8 USE DEPENDS ON OPCODE
2260  FMP.RETURN    .BS 1 ERROR CODE
2270                .BS 1 ?
2280  FMP.PNTR.WORK .BS 2 ADDR OF WORKAREA IN FILE BUFFER
2290  FMP.PNTR.TS   .BS 2 ADDR OF T/S LIST IN FILE BUFFER
2300  FMP.PNTR.DATA .BS 2 ADDR OF DATA IN FILE BUFFER
2310                .BS 4 ?
2320  *---------------------------------
2330  *      FILE MANAGER WORKAREA
2340  *---------------------------------
2350  FMW.TS.TS1  .BS 2 T/S OF FIRST T/S LIST SECTOR
2360  FMW.TS.TSC  .BS 2 T/S OF CURRENT T/S LIST SECTOR
2370  FMW.FLAGS   .BS 1 CHECKPOINT FLAGS
2380  FMW.TS.DATA .BS 2 T/S OF CURRENT DATA SECTOR
2390              .BS 2 DIRECTORY SECTOR INDEX
2400              .BS 2  # SECTORS PER TS LIST
2410              .BS 2  1ST SECTOR
2420              .BS 2  LAST SECTOR+1
2430              .BS 2  CURRENT SECTOR
2440  FMW.SECTSZ  .BS 2  SECTOR SIZE IN BYTES
2450              .BS 4 FILE POSITION
2460              .BS 2 RECORD LENGTH FROM OPEN
2470              .BS 2  RECORD NUMBER
2480              .BS 2  BYTE OFFSET INTO RECORD
2490              .BS 2  # SECTORS IN FILE
2500              .BS 6 SECTOR ALLOCATION AREA
2510  FMW.FILTYP  .BS 1
2520  FMW.SLOT16  .BS 1
2530  FMW.DRIVE   .BS 1
2540  FMW.VOLUME  .BS 1  (COMPLEMENT FORM)
2550  FMW.TRACK   .BS 1
2560              .BS 5  <NOT USED>
```

```
2570  *--------------------------------
2580  *      STAGE 1 OF BOOT (EXECUTES AT $0800)
2590  *--------------------------------
2600         .OR $800
2610         .TA $E00
2620  BOOT.STAGE1
2630         .HS 01
2640  *   COMES HERE AFTER EACH SECTOR IS READ
2650         LDA $27      NEXT PAGE TO READ INTO
2660         CMP #9       FIRST TIME HERE?
2670         BNE .1       NO, SKIP OVER INITIALIZATION
2680         LDA $2B      SLOT*16
2690         LSR          GET SLOT #
2700         LSR
2710         LSR
2720         LSR
2730         ORA #$C0     BUILD ADDRESS INTO ROM
2740         STA $3F      FOR READING A SECTOR
2750         LDA #$5C
2760         STA $3E
2770         CLC
2780         LDA BT1.ADDR+1  COMPUTE ADDRESS OF LAST PAGE
2790         ADC BT1.N       TO BE READ
2800         STA BT1.ADDR+1
2810  .1     LDX BT1.N    # PAGES LEFT TO READ - 1
2820         BMI .2       FINISHED
2830         LDA SECTOR.NUMBER,X  CONVERT TO PHYSICAL SECTOR #
2840         STA $3D
2850         DEC BT1.N
2860         LDA BT1.ADDR+1
2870         STA $27
2880         DEC BT1.ADDR+1
2890         LDX $2B      SLOT*16
2900         JMP ($3E)    READ NEXT SECTOR
2910  .2     INC BT1.ADDR+1  POINT AT STAGE 2 LOADER
2920         INC BT1.ADDR+1
2930         JSR MON.SETKBD
2940         JSR MON.SETVID
2950         JSR MON.INIT
2960         LDX $2B      SLOT*16
2970         JMP (BT1.ADDR)
2980  *--------------------------------
2990  SECTOR.NUMBER
3000         .HS 000D0B0907050301
3010         .HS 0E0C0A080604020F
3020  *--------------------------------
3030  *      DOS 3.3 PATCHES FOR APPEND AND VERIFY
3040  *--------------------------------
3050         .OR $B65D
3060         .TA $0E5D
3070  APPEND.FLAG .BS 1
3080  PATCH.DOS33.1
3090         JSR $A764    LOCATE AND FREE FILE BUFFER
3100         BCS .1
```

```
3110        LDA #0       CLEAR APPEND FLAG
3120        TAY
3130        STA APPEND.FLAG
3140        STA ($40),Y
3150  .1    LDA FMP.RETURN
3160        JMP $A6D2
3170  *---------------------------------
3180  PATCH.DOS33.2
3190        LDA APPEND.FLAG
3200        BEQ .1
3210        INC FMP.DATA
3220        BNE .1
3230        INC FMP.DATA+1
3240  .1    LDA #0       CLEAR APPEND FLAG
3250        STA APPEND.FLAG
3260        JMP $A546
3270  *---------------------------------
3280  PATCH.DOS33.3
3290        STA FMP.SUBCOD
3300        JSR $A6A8
3310        JSR $A2EA
3320        JMP $A27D
3330  *---------------------------------
3340  PATCH.DOS33.4
3350        LDY #19      LOOK AT FILE POSITION
3360  .1    LDA ($42),Y
3370        BNE .4       NOT AT 0000
3380        INY
3390        CPY #23
3400        BNE .1       TEST 4 BYTES
3410        LDY #25
3420  .2    LDA ($42),Y
3430        STA FMP.DATA-25,Y
3440        INY
3450        CPY #29      MOVE 4 BYTES
3460        BNE .2
3470  .3    JMP $A6BC
3480  .4    LDX #$FF
3490        STX APPEND.FLAG
3500        BNE .3       ...ALWAYS
3510        .BS 29       <NOT USED>
3520  *---------------------------------
3530  *     STRANGE CODE IN THE MIDDLE OF NOWHERE
3540  *---------------------------------
3550        JSR MON.HOME  CLEAR SCREEN
3560        LDA #$C2      PRINT "B01-00"
3570        JSR MON.COUT
3580        LDA #1
3590        JSR MON.PRBYTE
3600        LDA #$AD
3610        JSR MON.COUT
3620        LDA #0
3630        JSR MON.PRBYTE
3640        RTS
```

```
3650         .BS 21       <NOT USED>
3660         .OR $08FD
3670         .TA $0EFD
3680  BT1.ADDR .DA $3600
3690  BT1.N    .DA #9
3700  *---------------------------------
3710  *      SECOND STAGE OF BOOT
3720  *---------------------------------
3730         .OR $B700
3740         .TA $0F00
3750  BOOT.STAGE2
3760         STX IOB.SLOT16
3770         STX IOB.PRVSLT
3780         LDA #1
3790         STA IOB.PRVDRV
3800         STA IOB.DRIVE
3810         LDA BT.N
3820         STA BT.CNT
3830         LDA #2
3840         STA IOB.TRACK
3850         LDA #4
3860         STA IOB.SECTOR
3870         LDY BT.BT1+1
3880         DEY
3890         STY IOB.BUFFER+1
3900         LDA #1
3910         STA IOB.OPCODE
3920         TXA          SLOT*16
3930         LSR          GET SLOT #
3940         LSR
3950         LSR
3960         LSR
3970         TAX
3980         LDA #0
3990         STA $4F8,X
4000         STA $478,X
4010         JSR RW.PAGES
4020         LDX #$FF
4030         TXS          EMPTY STACK
4040         STX IOB.VOLUME
4050         JMP $BFC8    PATCH TO SETVID AND CLOBBER
4060  *                   THE LANGUAGE CARD, IF IN SLOT 0
4070         JSR MON.SETKBD
4080         JMP $9D84    DOS HARD ENTRY
4090  *---------------------------------
4100  *      WRITE DOS IMAGE ON TRACKS 0-2
4110  *---------------------------------
4120  WRITE.DOS.IMAGE
4130         LDA BT.BT1+1 COMPUTE # OF PAGES
4140         SEC
4150         SBC IOB.BUFFER+1
4160         STA BT.CNT
4170         LDA BT.BT1+1 START AT END, WORK BACKWARD
4180         STA IOB.BUFFER+1
```

```
4190          DEC IOB.BUFFER+1
4200          LDA #2       START ON TRACK 2
4210          STA IOB.TRACK
4220          LDA #4       SECTOR 4
4230          STA IOB.SECTOR
4240          LDA #2
4250          STA IOB.OPCODE
4260          JSR RW.PAGES WRITE STAGE2 PART OF DOS
4270          LDA BT.BT1+1 SET UP BOOT SECTOR IMAGE
4280          STA BT1.ADDR+1+$B600-$0800
4290          CLC          COMPUTE STARTING ADDRESS OF WRITE
4300          ADC #9
4310          STA IOB.BUFFER+1
4320          LDA #10      WRITE 10 PAGES
4330          STA BT.CNT
4340          SEC
4350          SBC #1
4360          STA BT1.N+$B600-$0800
4370          STA IOB.SECTOR
4380          JSR RW.PAGES WRITE SECTORS 9-0 ON TRACK 0
4390          RTS
4400   *--------------------------------
4410          .HS 000000000000   <NOT USED>
4420   *--------------------------------
4430   *      READ/WRITE A GROUP OF PAGES
4440   *
4450   *      BT.CNT       # OF SECTORS TO READ/WRITE
4460   *      IOB          SET UP FOR FIRST TS TO R/W
4470   *--------------------------------
4480   RW.PAGES
4490          LDA BT.IOB+1 GET IOB ADDRESS
4500          LDY BT.IOB
4510          JSR ENTER.RWTS  READ/WRITE ONE SECTOR
4520          LDY IOB.SECTOR  IGNORE ERRORS IF ANY
4530          DEY             BACK UP SECTOR #
4540          BPL .1          STILL IN SAME TRACK
4550          LDY #15         START WITH SECTOR 15 IN NEXT TRACK
4560          NOP
4570          NOP
4580          DEC IOB.TRACK   BACKWARD THROUGH THE TRACKS
4590   .1     STY IOB.SECTOR
4600          DEC IOB.BUFFER+1  DOWN ONE PAGE IN MEMORY
4610          DEC BT.CNT      ANY MORE PAGES TO DO?
4620          BNE RW.PAGES    YES
4630          RTS             NO, RETURN
4640   *--------------------------------
4650   *      ENTER RWTS
4660   *--------------------------------
4670   ENTER.RWTS
4680          PHP          SAVE STATUS ON STACK
4690          SEI          DISABLE INTERRUPTS
4700          JSR RWTS     CALL RWTS
4710          BCS .1       ERROR RETURN
4720          PLP          RESTORE STATUS
```

```
4730        CLC          SIGNAL NO RWTS ERROR
4740        RTS          RETURN TO CALLER
4750 .1     PLP          RESTORE STATUS
4760        SEC          SIGNAL RWTS ERROR
4770        RTS          RETURN TO CALLER
4780 *--------------------------------
4790 *      SET UP RWTS TO WRITE DOS
4800 *--------------------------------
4810 SETUP.WRITE.DOS
4820        LDA FMP.SUBCOD  IMAGE ADDRESS
4830        STA IOB.BUFFER+1
4840        LDA #0
4850        STA IOB.BUFFER
4860        LDA FMW.VOLUME  VOLUME #
4870        EOR #$FF        UNCOMPLEMENT IT
4880        STA IOB.VOLUME
4890        RTS
4900 *--------------------------------
4910 *      CLEAR 256 BYTES STARTING AT ($42,43)
4920 *--------------------------------
4930 ZERO.CURRENT.BUFFER
4940        LDA #0
4950        TAY
4960 .1     STA ($42),Y
4970        INY
4980        BNE .1
4990        RTS
5000 *--------------------------------
5010 *      PARAMETERS FOR SECOND STAGE OF BOOT PROCESS
5020 *--------------------------------
5030        .BS 1        <NOT USED>
5040 BT.N   .DA #27      # OF PAGES TO R/W (PARAMETER)
5050 BT.CNT .BS 1        # OF PAGES TO R/W (VARIABLE)
5060 BT.1S  .DA #10      1ST SECTOR # IN THIS STAGE
5070        .BS 1
5080 BT.IOB .DA IOB      ADDRESS OF IOB
5090 BT.BT1 .DA BOOT.STAGE1+$B600-$0800  ADDR OF 1ST STAGE BOOT
5100 *--------------------------------
5110 *      IOB FOR RWTS CALLS
5120 *--------------------------------
5130 IOB
5140 IOB.TYPE   .BS 1    0--MUST BE $01
5150 IOB.SLOT16 .BS 1    1--SLOT # TIMES 16
5160 IOB.DRIVE  .BS 1    2--DRIVE # (1 OR 2)
5170 IOB.VOLUME .BS 1    3--DESIRED VOL # (0 MATCHES ANY)
5180 IOB.TRACK  .BS 1    4--TRACK # (0 TO 34)
5190 IOB.SECTOR .BS 1    5--SECTOR # (0 TO 15)
5200 IOB.PNTDCT .DA DCT  6--ADDRESS OF DCT
5210 IOB.BUFFER .BS 2    8--ADDRESS OF DATA
5220 IOB.SECTSZ .BS 2   10--# BYTES IN A SECTOR
5230 IOB.OPCODE .BS 1   12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
5240 IOB.ERROR  .BS 1   13--ERROR CODE: 0, 8, 10, 20, 40, 80
5250 IOB.ACTVOL .BS 1   14--ACTUAL VOLUME # FOUND
5260 IOB.PRVSLT .BS 1   15--PREVIOUS SLOT #
```

```
5270  IOB.PRVDRV .BS 1   16--PREVIOUS DRIVE #
5280             .BS 2
5290  DCT    .HS 0001EFD8
5300         .BS 1
```

==================================================================
DOCUMENT :AAL-8110:DOS3.3:S.Mtr.Xtns.txt
==================================================================

```
 1000  *--------------------------------
 1010  *      MONITOR EXTENSIONS
 1020  *--------------------------------
 1030  MON.YSAV   .EQ $34
 1040  PC         .EQ $3A,3B
 1050  BGN.ADDR   .EQ $3C,3D
 1060  END.ADDR   .EQ $3E,3F
 1070  WBUF       .EQ $200
 1080  MON.PRNTYX .EQ $F940
 1090  MON.NXTA1  .EQ $FCBA
 1100  MON.XAM8   .EQ $FDA3
 1110  MON.COUT   .EQ $FDED
 1120  MON.LIST   .EQ $FE63
 1130  MON.A1PC   .EQ $FE75
 1140  *--------------------------------
 1150         .OR $300
 1160  *--------------------------------
 1170  SETUP  LDA #CTRLY
 1180         STA $3F9
 1190         LDA /CTRLY
 1200         STA $3FA
 1210         RTS
 1220  *--------------------------------
 1230  CTRLY  CMP #$AD     MINUS?
 1240         BEQ SUB
 1250         CMP #$AB     PLUS?
 1260         BEQ ADD
 1270         CMP #$BA     COLON?
 1280         BEQ RETURN
 1290         LDY MON.YSAV LOOK BEYOND CONTROL-Y
 1300         LDA WBUF,Y
 1310         LDY #0
 1320         CMP #'L+$80
 1330         BNE DUMP
 1340         INC MON.YSAV
 1350  *--------------------------------
 1360  DISASM JSR MON.A1PC
 1370  .1     LDA #1       DISASSEMBLE ONE LINE
 1380         JSR MON.LIST
 1390         CLC
 1400         LDA PC
 1410         SBC END.ADDR
 1420         LDA PC+1
 1430         SBC END.ADDR+1
 1440         BCC .1
 1450  RETURN RTS
 1460  *--------------------------------
 1470  DUMP   LDA END.ADDR
 1480         ORA #7       FINISH LAST ROW OF 8
```

```
1490        STA PC
1500        LDA END.ADDR+1
1510        STA PC+1
1520        LDA BGN.ADDR START WITH FULL ROW OF 8
1530        AND #$F8
1540        STA BGN.ADDR
1550  .1    JSR MON.XAM8
1560        SEC          BACK UP POINTER FOR ROW
1570        LDA BGN.ADDR
1580        SBC #8
1590        STA BGN.ADDR
1600        BCS .2       NO BORROW
1610        DEC BGN.ADDR+1
1620  .2    LDA #$A0     PRINT BLANK
1630        JSR MON.COUT
1640  .3    LDY #0
1650        LDA (BGN.ADDR),Y
1660        ORA #$80     MAKE NORMAL VIDEO
1670        CMP #$A0     SEE IF PRINTABLE
1680        BCS .4       YES
1690        EOR #$C0     MAKE CONTROLS INTO FLASHING ALPHA
1700  .4    JSR MON.COUT PRINT IT
1710        JSR MON.NXTA1  ADVANCE POINTER
1720        BCC .3       MORE ON THIS ROW
1730        LDA BGN.ADDR
1740        CMP PC       SEE IF FINISHED WITH DUMP
1750        LDA BGN.ADDR+1
1760        SBC PC+1
1770        BCC .1       NO
1780        RTS          YES
1790  *---------------------------------
1800  SUB   SEC
1810        LDA BGN.ADDR
1820        SBC END.ADDR
1830        TAX
1840        LDA BGN.ADDR+1
1850        SBC END.ADDR+1
1860        JMP AS1
1870  *---------------------------------
1880  ADD   CLC
1890        LDA BGN.ADDR
1900        ADC END.ADDR
1910        TAX
1920        LDA BGN.ADDR+1
1930        ADC END.ADDR+1
1940  AS1   TAY
1950
1960        LDA #$BD     EQUAL SIGN
1970        JSR MON.COUT
1980        JMP MON.PRNTYX
```

```
=====================================================================
DOCUMENT :AAL-8110:DOS3.3:S.Prm.B..Savoie.txt
=====================================================================
```

```
 1000        .LIF
 1010  *---------------------------------
 1020  *      SIEVE PROGRAM:
 1030  *      CALCULATES FIRST 1899 PRIMES IN 1.39 SECONDS!
 1040  *
 1050  *      INSPIRED BY JIM GILBREATH, BYTE, 9/81
 1060  *
 1070  *      WRITTEN BY WILLIAM ROBERT SAVOIE
 1080  *                 4405 DELASHMITT RD. APT 15
 1090  *                 HIXSON, TENN  37343
 1100  *---------------------------------
 1110  BUFF   .EQ $3500    START OF BUFFER (#BUFF=0)
 1120  SIZE   .EQ 8189     SIZE OF FLAG ARRAY
 1130  *---------------------------------
 1140  *      PAGE-ZERO VARIABLES
 1150  *---------------------------------
 1160  INDEX  .EQ $06      PAGE ZERO INDEX (LOCATION FOR I)
 1170  PRIME  .EQ $08      PRIME LOCATION
 1180  KVAR   .EQ $19      K VARIABLE
 1190  CVAR   .EQ $1B      COUNT OF PRIME
 1200  ARRAY  .EQ $1D      ARRAY POINTER
 1210  SAVE   .EQ $1F      COUNT LOOP
 1220  *---------------------------------
 1230  *      ROM ROUTINES
 1240  *---------------------------------
 1250  HOME   .EQ $FC58    CLEAR VIDEO
 1260  CR     .EQ $FD8E    CARRIAGE RETURN
 1270  LINE   .EQ $FD9E    PRINT "-"
 1280  PRINTN .EQ $F940    PRINT 2 BYTE NUMBER IN HEX
 1290  BELL   .EQ $FBE2    SOUND BELL WHEN DONE
 1300  *---------------------------------
 1310  * RUN PROGRAM 100 TIMES FOR ACCURATE TIME MEASUREMENTS!
 1320  *---------------------------------
 1330  START  JSR HOME     CLEAR SCREEN
 1340         JSR CR       CARRIAGE RETURN
 1350         LDA #100     LOOP 100 TIMES
 1360         STA SAVE     SET COUNTER
 1370  .01    JSR GO       RUN PRIME
 1380         DEC SAVE     DECREASE SAVE
 1390         BNE .01      LOOP
 1400         JSR PRINT    PRINT COUNT
 1410         JSR BELL     READ WATCH!
 1420         RTS
 1430  *---------------------------------
 1440  *      RESET VARIABLES
 1450  *---------------------------------
 1460  GO     LDY #00      CLEAR INDEX
 1470         STY CVAR     CLEAR COUNT VARIABLE
 1480         STY CVAR+1   HI BYTE TOO
```

```
       STY INDEX    CLEAR INDEX
       STY INDEX+1  HI BYTE TOO
       STY ARRAY    LOW BYTE OF ARRAY
       LDA /BUFF    GET BUFFER LOCATION
       STA ARRAY+1  SET ARRAY POINTER
       LDA #$01     LOAD WITH ONE
       LDX /SIZE    LOAD STOP BYTE
       INX          MAKE PAGE LARGER
*---------------------------------
*      SET EACH ELEMENT IN ARRAY TO ONE
*---------------------------------
SET    STA (ARRAY),Y  SET MEMORY
       DEY          NEXT LOCATION
       BNE SET      GO 256 TIMES
       INC ARRAY+1  MOVE ARRAY INDEX
       DEX          TEST END
       BNE SET      GO TELL END

* SET ARRAY INDEX AT START OF BUFFER
       LDA #BUFF    SET BUFFER LOCATION
       STA ARRAY    IN ARRAY POINTER LOW
       LDA /BUFF    SET BUFFER LOCATION
       STA ARRAY+1  IN ARRAY POINTER
       JMP FORIN    ENTER SIEVE ALGORITHM

* SCAN ENTIRE ARRAY AND PROBAGATE LAST PRIME
FORNXT INC INDEX    INCREASE LOW BYTE
       BNE FORIN    GO IF < 256
       INC INDEX+1  INCREASE HI BYTE
FORIN  LDA INDEX    GET INDEX TO ARRAY
       CLC          READY ADD
       STA ARRAY    SAVE LOW BYTE
       LDA INDEX+1  GET HI BYTE
       ADC /BUFF    ADD BUFFER LOCATION
       STA ARRAY+1  SET POINTER
       LDY #00      CLEAR Y REGISTER
       LDA (ARRAY),Y  GET ARRAY VALUE
       BEQ FORNXT   GO IF FLAG=0 SINCE NOT PRIME
* CALCULATE NEXT PRIME NUMBER WITH P=I+I+3
       LDA INDEX    MAKE P=I+3
       ADC #03      ADD THREE
       STA PRIME
       LDA INDEX+1
       ADC #00      ADD CARRY
       STA PRIME+1
* NOW P=I+3
       LDA PRIME
       ADC INDEX    MAKE P=P+I
       STA PRIME
       LDA PRIME+1
       ADC INDEX+1  ADD HI BYTE
       STA PRIME+1  SAVE P

* NOW CALCULATE K=I+PRIME (CLEAR BEYOND PRIME)
```

```
2030        LDA INDEX    ADD I TO P
2040        ADC PRIME
2050        STA KVAR     SAVE IN K
2060        LDA INDEX+1
2070        ADC PRIME+1  ADD HI BYTE TOO
2080        STA KVAR+1   SAVE K VALUE
2090
2100  * SEE IF K > SIZE AND MODIFY ARRAY IF NOT
2110  .02   LDA KVAR     GET K VAR
2120        SEC          SET CARRY FOR SUB
2130        SBC #SIZE    SUBTRACT SIZE
2140        LDA KVAR+1   GET HI BYTE
2150        SBC /SIZE    SUBTRACT TOO
2160        BCS .03      GO IF K < SIZE
2170  * ASSIGN ARRAY(K)=0 SINCE PRIME CAN BE ADDED TO MAKE NUMBER
2180  * THEREFORE THIS CANNOT BE PRIME! (PROBAGATE THROUGH ARRAY)
2190        LDA KVAR     GET INDEX TO ARRAY
2200        STA ARRAY    SAVE LOW BYTE
2210        LDA KVAR+1   GET HI BYTE
2220        ADC /BUFF    ADD BUFFER OFFSET
2230        STA ARRAY+1  SAVE ARRAY INDEX
2240        LDA #00      CLEAR A
2250        TAY          AND Y REGISTER
2260        STA (ARRAY),Y  CLEAR ARRAY LOCATION
2270  * CREATE NEW K FROM K=K+PRIME (MOVE THROUGH ARRAY)
2280        LDA KVAR     GET K LOW
2290        ADC PRIME    ADD PRIME
2300        STA KVAR     SAVE K
2310        LDA KVAR+1   NOW ADD HI BYTES
2320        ADC PRIME+1
2330        STA KVAR+1
2340        JMP .02      LOOP TELL ARRAY DONE
2350  * NOW COUNT PRIMES FOUND  (C=C+1)
2360  .03
2370  * --NOTE-- DELETE NEXT LINE TO TIME PROGRAM (JSR PRINTP)
2380        JSR PRINTP   PRINT PRIME
2390        INC CVAR     ADD ONE
2400        BNE .04      GO IF NO OVERFLOW
2410        INC CVAR+1   HI BYTE COUNTER
2420  .04   LDA INDEX    GET INDEX
2430  * TEST TO SEE IF WE HAVE INDEXED THROUGH ENTIRE ARRAY
2440        SBC #SIZE    SUBTRACT SIZE
2450        LDA INDEX+1  GET HI BYTE TOO
2460        SBC /SIZE    SUBTRACT HI BYTE
2470        BCC FORNXT   CONTINUE?
2480        RTS
2490  *----------------------------------
2500  * PRINT THE NUMBER OF PRIMES FOUND
2510  *----------------------------------
2520  PRINT  LDY CVAR+1   GET HI BYTE OF COUNT
2530        LDX CVAR
2540        JSR PRINTN   PRINT PRIMES FOUND
2550        RTS          JOB DONE, RETURN
2560  *----------------------------------
```

```
2570  *      PRINT THE PRIME NUMBER (OPTIONAL)
2580  *---------------------------------
2590  PRINTP LDY PRIME+1  HI BYTE
2600         LDX PRIME
2610         JSR PRINTN
2620         JSR LINE     VIDEO "-" OUT
2630         SEC
2640         RTS
```

```
=============================================================================
DOCUMENT :AAL-8110:DOS3.3:S.Prm.Bnch.Fst.txt
=============================================================================

 1000  *---------------------------------
 1010  * SIEVE PROGRAM:
 1020  * CALCULATES FIRST 1899 PRIMES IN .74 SECONDS!
 1030  *
 1040  * INSPIRED BY JIM GILBREATH
 1050  *   (SEE BYTE MAGAZINE, 9/81, PAGES 180-198.)
 1060  * AND BY WILLIAM ROBERT SAVOIE
 1070  *   4405 DELASHMITT RD. APT 15
 1080  *   HIXSON, TENN  37343
 1090  *---------------------------------
 1100  ARRAY  .EQ $3500    FLAG BYTE ARRAY
 1110  SIZE   .EQ 8192     SIZE OF FLAG ARRAY
 1120  *---------------------------------
 1130  * PAGE-ZERO VARIABLES
 1140  *---------------------------------
 1150  A.PNTR .EQ $06,07   POINTER TO FLAG ARRAY FOR OUTER LOOP
 1160  B.PNTR .EQ $08,09   POINTER TO FLAG ARRAY FOR INNER LOOP
 1170  PRIME  .EQ $1B,1C   LATEST PRIME NUMBER
 1180  COUNT  .EQ $1D,1E   # OF PRIMES SO FAR
 1190  TIMES  .EQ $1F      COUNT LOOP
 1200  *---------------------------------
 1210  * APPLE ROM ROUTINES USED
 1220  *---------------------------------
 1230  PRINTN .EQ $F940    PRINT 2 BYTE NUMBER FROM MONITOR
 1240  HOME   .EQ $FC58    CLEAR VIDEO
 1250  CR     .EQ $FD8E    CARRIAGE RETURN
 1260  LINE   .EQ $FD9E    PRINT "-"
 1270  BELL   .EQ $FBE2    SOUND BELL WHEN DONE
 1280  *---------------------------------
 1290  * RUN PROGRAM 100 TIMES FOR ACCURATE TIME MEASUREMENTS!
 1300  *---------------------------------
 1310  START  JSR HOME     CLEAR SCREEN
 1320         LDA #100     LOOP 100 TIMES
 1330         STA TIMES    SET COUNTER
 1340  .1     JSR GENERATE.PRIMES
 1350         LDA $400     TOGGLE SCREEN FOR VISIBLE INDICATOR
 1360         EOR #$80     OF ACTION
 1370         STA $400
 1380         DEC TIMES
 1390         BNE .1       LOOP
 1400         JSR BELL     READ WATCH!
 1410         LDY COUNT+1  GET HI BYTE OF COUNT
 1420         LDX COUNT
 1430         JSR PRINTN   PRINT PRIMES FOUND
 1440         RTS
 1450  *---------------------------------
 1460  *      GENERATE THE PRIMES
 1470  *---------------------------------
 1480  GENERATE.PRIMES
```

```
1490        LDY #0       CLEAR INDEX
1500        STY COUNT    CLEAR COUNT VARIABLE
1510        STY COUNT+1
1520        STY A.PNTR   SET UP POINTER FOR OUTER LOOP
1530        LDA /ARRAY
1540        STA A.PNTR+1
1550        LDA #1       LOAD WITH ONE
1560        LDX /SIZE      NUMBER OF PAGES TO STORE IN
1570 *-------------------------------
1580 * SET EACH ELEMENT IN ARRAY TO ONE
1590 *-------------------------------
1600 .1     STA (A.PNTR),Y  SET FLAG TO 1
1610        INY          NEXT LOCATION
1620        BNE .1       GO 256 TIMES
1630        INC A.PNTR+1 POINT AT NEXT PAGE
1640        DEX          NEXT PAGE
1650        BNE .1       MORE PAGES
1660 *-------------------------------
1670 * SCAN ENTIRE ARRAY, LOOKING FOR A PRIME
1680 *-------------------------------
1690        LDA /ARRAY   SET A.PNTR TO BEGINNING AGAIN
1700        STA A.PNTR+1
1710 .2     LDY #0       CLEAR INDEX
1720        LDA (A.PNTR),Y  LOOK AT NEXT FLAG
1730        BEQ .6       NOT PRIME, ADVANCE POINTER
1740 *-------------------------------
1750 * CALCULATE CURRENT INDEX INTO FLAG ARRAY
1760 *-------------------------------
1770        SEC
1780        LDA A.PNTR+1
1790        SBC /ARRAY
1800        TAX          SAVE HI-BYTE OF INDEX
1810        LDA A.PNTR   LO-BYTE OF INDEX
1820 *-------------------------------
1830 * CALCULATE NEXT PRIME NUMBER WITH P=I+I+3
1840 *-------------------------------
1850        ASL          DOUBLE THE INDEX
1860        TAY
1870        TXA          HI-BYTE OF INDEX
1880        ROL
1890        TAX
1900        TYA          NOW ADD 3
1910        ADC #3
1920        STA PRIME
1930        BCC .3
1940        INX
1950 .3     STX PRIME+1
1960 *-------------------------------
1970 * FOLLOWING 4 LINES CHANGE ALGORITHM SLIGHTLY
1980 * TO SPEED IT UP FROM .93 TO .74 SECONDS
1990 *-------------------------------
2000        TXA          TEST HIGH BYTE
2010        BNE .5       PRIME > SQRT(16384)
2020        CPY #127
```

```
2030        BCS .5       PRIME > SQRT(16384)
2040  *---------------------------------
2050  * NOW CLEAR EVERY P-TH ENTRY AFTER P
2060  *---------------------------------
2070        LDY #0
2080        LDA A.PNTR   USE CURRENT OUTER POINTER FOR INNER POINTER
2090        STA B.PNTR
2100        LDA A.PNTR+1
2110        STA B.PNTR+1
2120        CLC          BUMP ARRAY POINTER BY P
2130  .4    LDA B.PNTR   BUMP TO NEXT SLOT
2140        ADC PRIME
2150        STA B.PNTR
2160        LDA B.PNTR+1
2170        ADC PRIME+1
2180        STA B.PNTR+1
2190        CMP /ARRAY+SIZE     SEE IF BEYOND END OF ARRAY
2200        BCS .5       YES, FINISHED CLEARING
2210        TYA          NO, CLEAR ENTRY IN ARRAY
2220        STA (B.PNTR),Y
2230        BEQ .4       ...ALWAYS
2240  *---------------------------------
2250  * NOW COUNT PRIMES FOUND  (C=C+1)
2260  *---------------------------------
2270  .5
2280  *     JSR PRINTP   PRINT PRIME
2290        INC COUNT
2300        BNE .6
2310        INC COUNT+1
2320  *---------------------------------
2330  * ADVANCE OUTER POINTER AND TEST IF FINISHED
2340  *---------------------------------
2350  .6    INC A.PNTR
2360        BNE .7
2370        INC A.PNTR+1
2380  .7    LDA A.PNTR+1
2390        CMP /ARRAY+SIZE
2400        BCC .2
2410        RTS
2420  *---------------------------------
2430  * OPTIONAL PRINT PRIME SUBROUTINE
2440  *---------------------------------
2450  PRINTP LDY PRIME+1  HI BYTE
2460        LDX PRIME
2470        JSR PRINTN   PRINT DECIMAL VAL
2480        JSR LINE     VIDEO "-" OUT
2490        RTS
```

```
=====================================================================
DOCUMENT :AAL-8110:DOS3.3:S.Prm.Bnch.RBSC.txt
=====================================================================

 1000  *--------------------------------
 1010  * SIEVE PROGRAM:
 1020  * CALCULATES FIRST 1899 PRIMES IN 1.03 SECONDS!
 1030  *
 1040  * INSPIRED BY JIM GILBREATH
 1050  * SEE BYTE MAGAZINE, 9/81, PAGES 180-198.
 1060  *
 1070  * WRITTEN 9-3-81 BY:
 1080  * WILLIAM ROBERT SAVOIE
 1090  * 4405 DELASHMITT RD. APT 15
 1100  * HIXSON, TENN  37343
 1110  *
 1120  * EXTENSIVELY REVISED BY BOB SANDER-CEDERLOF
 1130  * TO SHAVE TIME FROM 1.39 SECONDS TO 1.03 SECONDS
 1140  *--------------------------------
 1150  * SIEVE PARAMETERS
 1160  *--------------------------------
 1170  BUFF   .EQ $3500    START OF BUFFER (#BUFF=0)
 1180  SIZE   .EQ 8192     SIZE OF FLAG ARRAY
 1190  *--------------------------------
 1200  * PAGE-ZERO VARIABLES
 1210  *--------------------------------
 1220  INDEX  .EQ $06,07   PAGE ZERO INDEX (LOCATION FOR I)
 1230  PRIME  .EQ $08,09   PRIME LOCATION
 1240  CVAR   .EQ $1B,1C   COUNT OF PRIME
 1250  ARRAY  .EQ $1D,1E   ARRAY POINTER
 1260  TIMES  .EQ $1F      COUNT LOOP
 1270  *--------------------------------
 1280  * APPLE ROM ROUTINES USED
 1290  *--------------------------------
 1300  PRINTN .EQ $F940    PRINT 2 BYTE NUMBER FROM MONITOR
 1310  HOME   .EQ $FC58    CLEAR VIDEO
 1320  CR     .EQ $FD8E    CARRIAGE RETURN
 1330  LINE   .EQ $FD9E    PRINT "-"
 1340  BELL   .EQ $FBE2    SOUND BELL WHEN DONE
 1350  *--------------------------------
 1360  * RUN PROGRAM 100 TIMES FOR ACCURATE TIME MEASUREMENTS!
 1370  *--------------------------------
 1380  START  JSR HOME     CLEAR SCREEN
 1390         LDA #100     LOOP 100 TIMES
 1400         STA TIMES    SET COUNTER
 1410  .1     JSR GO       RUN PRIME
 1420         LDA $400     TOGGLE SCREEN FO VISIBLE INDICATOR
 1430         EOR #$80     OF ACTION
 1440         STA $400
 1450         DEC TIMES
 1460         BNE .1       LOOP
 1470         JSR BELL     READ WATCH!
 1480         LDY CVAR+1   GET HI BYTE OF COUNT
```

```
1490        LDX CVAR
1500        JSR PRINTN   PRINT PRIMES FOUND
1510        RTS
1520 *---------------------------------
1530 * RESET VARIABLES
1540 *---------------------------------
1550 GO     LDY #00      CLEAR INDEX
1560        STY CVAR     CLEAR COUNT VARIABLE
1570        STY CVAR+1   HI BYTE TOO
1580        STY INDEX    CLEAR INDEX
1590        STY INDEX+1  HI BYTE TOO
1600        STY ARRAY    LOW BYTE OF ARRAY
1610        LDA /BUFF    GET BUFFER LOCATION
1620        STA ARRAY+1  SET ARRAY POINTER
1630        LDA #$01     LOAD WITH ONE
1640        LDX /SIZE      NUMBER OF PAGES TO STORE IN
1650 *---------------------------------
1660 * SET EACH ELEMENT IN ARRAY TO ONE,
1670 *---------------------------------
1680 SET    STA (ARRAY),Y  SET MEMORY
1690        INY          NEXT LOCATION
1700        BNE SET      GO 256 TIMES
1710        INC ARRAY+1  MOVE ARRAY INDEX
1720        DEX          TEST END
1730        BNE SET      GO TELL END
1740 *---------------------------------
1750 * SCAN ENTIRE ARRAY, LOOKING FOR A PRIME
1760 *---------------------------------
1770 FORIN  LDA INDEX    GET INDEX TO ARRAY
1780        CLC          READY ADD
1790        STA ARRAY    SAVE LOW BYTE
1800        LDA INDEX+1  GET HI BYTE
1810        ADC /BUFF    ADD BUFFER LOCATION
1820        STA ARRAY+1  SET POINTER
1830        LDY #00      CLEAR Y REGISTER
1840        LDA (ARRAY),Y  GET ARRAY VALUE
1850        BEQ .4       NOT PRIME, TRY NEXT ONE
1860 *---------------------------------
1870 * CALCULATE NEXT PRIME NUMBER WITH P=I+I+3
1880 *---------------------------------
1890        LDA INDEX    MAKE I+I IN X,Y
1900        ASL
1910        TAY
1920        LDA INDEX+1
1930        ROL
1940        TAX
1950        TYA          NOW ADD 3
1960        ADC #3
1970        STA PRIME
1980        BCC .1
1990        INX
2000 .1     STX PRIME+1
2010 *---------------------------------
2020 * NOW CLEAR EVERY P-TH ENTRY AFTER P
```

```
2030  *--------------------------------
2040         LDY #0
2050         CLC          BUMP ARRAY POINTER BY P
2060  .2     LDA ARRAY
2070         ADC PRIME
2080         STA ARRAY
2090         LDA ARRAY+1
2100         ADC PRIME+1
2110         STA ARRAY+1
2120         CMP /BUFF+SIZE      SEE IF BEYOND END OF ARRAY
2130         BCS .3       YES, FINISHED CLEARING
2140         TYA          NO, CLEAR ENTRY IN ARRAY
2150         STA (ARRAY),Y
2160         BEQ .2       ...ALWAYS
2170  *--------------------------------
2180  * NOW COUNT PRIMES FOUND  (C=C+1)
2190  *--------------------------------
2200  .3
2210  * --NOTE-- DELETE NEXT LINE TO TIME PROGRAM (JSR PRINTP)
2220         JSR PRINTP   PRINT PRIME
2230         INC CVAR     ADD ONE
2240         BNE .04      GO IF NO OVERFLOW
2250         INC CVAR+1   HI BYTE COUNTER
2260  *--------------------------------
2270  * INCREMENT INDEX AND TEST IF FINISHED
2280  *--------------------------------
2290  .4     INC INDEX
2300         BNE .5
2310         INC INDEX+1
2320  .5     LDA INDEX+1
2330         CMP /SIZE
2340         BCC FORIN
2350         RTS
2360  *--------------------------------
2370  * OPTIONAL PRINT PRIME SUBROUTINE
2380  *--------------------------------
2390  PRINTP LDY PRIME+1  HI BYTE
2400         LDX PRIME
2410         JSR PRINTN   PRINT DECIMAL VAL
2420         JSR LINE     VIDEO "-" OUT
2430         RTS
```

==============================================================================
DOCUMENT :AAL-8111:Articles:AS.ROMsFromAsm.txt
==============================================================================

Using Applesoft ROM's from Assembly Language

There are many useful entry points in the Applesoft ROM's. The problem is figuring out how to use them. John Crossley's article "Applesoft Internal Entry Points" (originally published in Apple Orchard Volume 1 Number 1 March 1980) gives a brief description of most of the usable subroutines. If you missed the article, you can still get it from the International Apple Corps. It has also recently been reprinted in "Call A.P.P.L.E. in Depth--All About Applesoft".

Now I want to show you how to use the floating point math subroutines. I won't cover every one of them, but enough to do most of the things you would ever need to do. This includes load, store, add, subtract, complement, compare, multiply, divide, print, and formatted-print.

Internal Floating Point Number Format

Applesoft stores floating point numbers in five bytes. The first byte is the binary exponent; the other four bytes are the mantissa: ee mm mm mm mm.

The exponent (ee) is a signed number in excess-$80 form. That is, $80 is added to the signed value. An exponent of +3 will be stored as $83; of -3, as $7D. If ee = $00, the entire number is considered to be zero, regardless of what the mantissa bytes are.

The mantissa is considered to be a fraction between $.80000000 and $.FFFFFFFF. Since the value is always normalized, the first bit of the mantissa is always "1". Therefore, there is no need to actually use that bit position for a mantissa bit. Instead, the sign of the number is stored in that position (0 for +, 1 for -). Here are some examples:

```
-10.0   84 A0 00 00 00
+10.0   84 20 00 00 00
+1.0    81 00 00 00 00
+1.75   81 60 00 00 00
-1.75   81 E0 00 00 00
+.1     7D 4C CC CC CD
```

The Applesoft math subroutines use a slightly different format for faster processing, called "unpacked format". In this format the leading mantissa bit is explicitly stored, and the sign value is stored separately. Several groups of page-zero locations are used to store operands and results. The most frequently used are called "FAC" and "ARG". FAC occupies locations $9D thru $A2; ARG, $A5 thru $AA.

Loading and Storing Floating Point Values

There are a handful of subroutines in ROM for moving numbers into and out of FAC and ARG. Here are the five you need to know about.

```
AS.MOVFM   $EAF9   unpack (Y,A) into FAC
AS.MOVMF   $EB2B   pack FAC into (Y,X)
AS.MOVFA   $EB53   copy ARG into FAC
AS.MOVAF   $EB63   copy FAC into ARG
AS.CONUPK  $E9E3   unpack (Y,A) into ARG
```

All of the above subroutines return with the exponent from FAC in the A-register, and with the Z-status bit set if (A)<0.

Here is an example which loads a value into FAC, and then stores it at a different location.

```
LDA #VAR1
LDY /VAR1    ADDRESS IN (Y,A)
JSR AS.MOVFM
LDX #VAR2
LDY /VAR2    ADDRESS IN (Y,X)
JSR AS.MOVMF
```

Arithmetic Subroutines

Once a number is unpacked in FAC, there are many subroutines which can operate on it.

```
AS.NEGOP   $EED0   FAC = -FAC

AS.FOUT    $ED34   convert FAC to decimal ASCII string
                   starting at $0100

AS.FCOMP   $EBB2   compare FAC to packed number at (Y,A)
                   return (A) = 1 if (Y,A) < FAC
                          (A) = 0 if (Y,A) = FAC
                          (A) =FF if (Y,A) > FAC

AS.FADD    $E7BE   load (Y,A) into ARG, and fall into...
AS.FADDT   $E7C1   FAC = ARG + FAC

AS.FSUB    $E7A7   load (Y,A) into ARG, and fall into...
AS.FSUBT   $E7AA   FAC = ARG - FAC

AS.FMUL    $E97F   load (Y,A) into ARG, and fall into...
AS.FMULT   $E982   FAC = ARG * FAC

AR.FDIV    $EA66   load (Y,A) into ARG, and fall into...
AS.FDIVT   $EA69   FAC = ARG / FAC
```

Here is an example which calculates VAR1 = (VAR2 + VAR3) / (VAR2 - VAR3).

```
LDA #VAR2   VAR2+VAR3
LDY /VAR2
JSR AS.MOVFM    VAR2 INTO FAC
```

```
LDA #VAR3
LDY /VAR3
JSR AS.FADD     + VAR3
LDX #VAR1
LDY /VAR1
JSR AS.MOVMF    STORE SUM TEMPORARILY IN VAR1
LDA #VAR3   VAR2-VAR3
LDY /VAR3
JSR AS.MOVFM    VAR3 INTO FAC
LDA #VAR2
LDY /VAR2
JSR AS.FSUB     VAR2-VAR3
LDA #VAR1
LDY /VAR1
JSR AS.FDIV     DIVIDE DIFFERENCE BY SUM
LDX #VAR1
LDY /VAR1
JSR AS.MOVMF    STORE THE QUOTIENT
```

As you can see, it is easy to get confused when writing this kind of code. It is so repetitive, there are so many setups of (Y,A) and (Y,X) addresses, that I make a lot of typing mistakes. It would be nice if there was an interface program between my assembly language coding and the Applesoft ROMs. I would rather write the above program like this:

```
JSR FP.LOAD   VAR2 INTO FAC
.DA VAR2
JSR FP.SUB    -VAR3
.DA VAR3
JSR FP.STORE  SAVE AT VAR1
.DA VAR0
JSR FP.LOAD   VAR2 INTO FAC
.DA VAR2
JSR FP.ADD    +VAR3
.DA VAR3
JSR FP.DIV    /(VAR2-VAR3)
.DA VAR1
JSR FP.STORE  STORE IN VAR1
.DA VAR1
```

Easy Interface to Applesoft ROMs

The first step in constructing the "easy interface" is to figure out a way to get the argument address from the calling sequence. That is, when I execute:

```
     JSR FP.LOAD
     .DA VAR1
```

how does FP.LOAD get the address VAR1?

I wrote a subroutine called GET.ADDR which does the job. Every one of my FP. subroutines starts by calling GET.ADDR to save the A-, X-, and Y-registers, and to return with the address which followed the JSR FP... in the Y- and A-registers. In fact, I return the low-byte of the address in both the A- and X-registers. That way the address is ready in both (Y,A) and (Y,X) form.

GET.ADDR is at lines 4260-4480. I save A, X, and Y in three local variables, and then pull off the return address from the stack and save it also. (This is the return to whoever called GET.ADDR). Then I save the current TXTPTR value. This is the pointer Applesoft uses when picking up bytes from your program to interpret them. I am going to borrow the CHRGET subroutine, so I need to save the current TXTPTR and restore it when I am finished. Then I pull the next address off the stack and stuff it into TXTPTR. This address is the return address to whoever called the FP... subroutine. It currently points to the third byte of that JSR, one byte before the .DA address we want to pick up.

I next call GET.ADDR2, which uses CHRGET twice to pick up the next two bytes after the JSR and returns them in X and Y. Then I push the return address I saved at the beginning of GET.ADDR, and RTS back. Note that TXTPTR now points at the second byte of the .DA address. It is just right for picking up another argument, or for returning. If there is another argument, I get it by calling GET.ADDR2 again. When I am ready for the final return, I do it by JMPing to FP.EXIT.

FP.EXIT, at lines 4670-4790, pushes the value in TXTPTR on the stack. It is the correct return address for the JSR FP.... Then I restore the old value of TXTPTR, along with the A-, X-, and Y-registers. And the RTS finishes the job.

The Interface Subroutines

I have alluded above to the "FP..." subroutines. In the listing I have shown eight of them, and you might add a dozen more after you get the hang of it.

```
FP.LOAD        load a value into FAC
FP.STORE       store FAC at address
FP.ADD         FAC = FAC + value
FP.SUB         FAC = FAC - value
FP.MUL         FAC = FAC * value
FP.DIV         FAC = FAC / value
FP.PRINT       print value the way Applesoft would
FP.PRINT.WD    print value with D digits after decimal
               in a W-character field
```

FP.LOAD, FP.STORE, FP.ADD, and FP.MUL are quite straightforward. All they do is call GET.ADDR to get the argument address, JSR into the Applesoft ROM subroutine, and JMP to FP.EXIT.

FP.SUB and FP.DIV are a little more interesting. I didn't like the way the Applesoft ROM subroutines ordered the operands. It looks to me like they want me to think in complements and reciprocals. Remember that AS.FDIV performs FAC = (Y,A) / FAC. It is more natural for me to think left-to-right, so my FP.DIV permorms FAC = FAC / value. Likewise for FP.SUB.

I reversed the sense of the subtraction after-the-fact, by just calling AS.NEGOP to complement the value in FAC. Reversing the division has to be done before calling AS.FDIV. I saved the argument address on the stack, called AS.MOVAF to copy FAC into ARG, called AS.MOVFM to get the argument into FAC, and then called AS.FDIVT.

FP.PRINT, at lines 1830-1930, is also quite simple. I call GET.ADDR to set up the argument address, and AS.MOVFM to load it into FAC. Then AS.FOUT converts it to an ASCII string starting at $0100. It terminates with a $00 byte. A short loop picks up the characters of this string and prints them by calling AS.COUT. I called AS.COUT, rather than $FDED in the monitor, so that Applesoft FLASH, INVERSE, and NORMAL would operate on the characters.

And now for the really interesting one....

Formatted Print Subroutine

FP.PRINT.WD expects three arguments: the address of the value to be printed, the field width to print it in, and the number of digits to print after the decimal point. Leading blanks and trailing zeroes will be printed if necessary. The Applesoft E-format will be caught and converted to the more civilized form. Fields up to 40 characters wide may be printed, which will accommodate up to 39 digits and a decimal point. If you try to print a number that is too wide for the field, it will try to fit it in by shifting off fractional digits. If it is still too wide, it will print a field of ">>>>" indicating overflow.

For example, look at how values 123.4567and 12345.67 would be printed for corresponding W and D:

| W | D | 123.4567 | 12345.67 |
|---|---|---|---|
| 10 | 1 | bbbbb123.4 | bbb12345.6 |
| 10 | 3 | bbb123.456 | b12345.670 |
| 10 | 5 | b123.45670 | 12345.6700 |
| 10 | 7 | 123.456700 | 12345.6700 |
| 7 | 1 | bb123.4 | 12345.6 |
| 4 | 1 | 123. | >>>> |

Sound pretty useful? I can hardly wait to start using it! Now let's walk through the code.

Lines 2380-2410 pick up the arguments. The value is loaded into FAC, and converted to a string at $0100 by AS.FOUT. Then I get the W and D values into X and Y.

Lines 2420-2510 check W and D. W must not be more than 40; if it is, use 40. (I arbitrarily chose 40 as the limit. If you want a different limit, you can use any value less than 128.) I also make sure that D is less than W. I save W in WD.GT in case I later need to print a field full of ">". Lines 2520-2560 compute W-D-1, which is the number of characters in the field to the left of the decimal point. I save the result back in W.

Lines 2570-2590 check whether AS.FOUT converted to the Applesoft E-format or not. The decimal exponent printed after E is still in $9A as a binary value. Numbers formatted the civilized way are handled by lines 2600-3160. E-format numbers are restructured by lines 3200-3930.

Lines 2600-2750 scan the string at $0100 up to the decimal point (or to the end if no decimal point). In other words, I am counting the number of characters AS.FOUT put before the decimal point. If W is bigger than that, the difference is the number of leading blanks I need to print. Since W is decremented inside the loop, the leading blank count is all that is left in W. But what if W goes negative, meaning that the number is too big for the field? Then I reduce D and try again. If I run out of "D" also, then the field is entirely too small, so I go to PRINT.GT to indicate overflow. If there was no decimal point on the end, the code at lines 2790-2820 appends one to the string.

Lines 2870-2980 scan over the fractional digits. If there are more than D of them, I store the end-of-string code ($00) after D digits. I also decrement D inside this loop, so that when the loop is finished D represents the number of trailing zeroes that I must add to fill out the field. (If the string runs out before D does, I need to print trailing zeroes.)

At line 3020, the leading blanks are printed (if any; remember that W had the leading blank count). Then lines 3060-3110 print the string at $0100. And finally, line 3150 prints out D trailing zeroes (D might be zero).

E-formatted numbers are a little tougher; we have to move the decimal point left or right depending on the exponent. We also might have to add zeroes before the decimal point, as well as after the fraction. Lines 3200-3330 scan through the converted string at $0100; the decimal point (if any) is removed, and an end-of-string byte ($00) is put where the "E" character is. Now all we have at $0100 is the sign and a string of significant digits, without decimal point or E-field.

Lines 3350-3600 test the range of the decimal exponent. Negative exponents are handled at lines 3370-3660, and positive ones at lines 3700-3930.

Negative exponents mean that the decimal point must be printed first, then possibly some leading zeroes, and then some significant digits. Lines 3370-3410 compute how many leading zeroes are needed. For example, the value .00123 would be converted by AS.COUT as "1.23E-03". The decimal exponent is -3, and we need two leading zeroes. The number of leading zeroes is -(dec.exp+1).

There is a little coding trick at line 3370. I want to compute -(dec.exp+1), and dec.exp is negative. By executing the EOR #$FF, the value is complemented and one is added at the same time! Why? Because the 6502 uses 2's complement arithmetic. Negative numbers are in the form 256-value. EOR #$FF is the same as doing 255-value, which is the same as 256-(value+1). Got it?

Line 3430 prints the leading blanks; lines 3450-3460 print the decimal point. Lines 3480-3520 print the leading zeroes, decrementing D along the way. When all the leading zeroes are out, D will indicate how many significant digits need to be printed.

Lines 3540-3620 print as many significant digits as will fit in the remaining part of the field (maybe none). Of course, the field might be large enough that we also need trailing zeroes. If so, line 3650 prints them.

What if the exponent was positive? Then lines 3700-3710 see if the number will fit in the field. If not, PRINT.GT will fill the field with ">". If it will fit, then the exponent is the number of digits to be printed. The number of leading blanks will be W-dec.exp-1 (the -1 is for the decimal point). Note that line 3740 complements and adds one at the same time, to get -(exp+1).

Line 3770 prints the leading blanks, if any. Lines 3780-3830 print the significant digits from the string at $0100. Lines 3840-3890 print any zeroes needed between the significant digits and the decimal point. Lines 3900-3910 print the decimal point, and line 3920 prints the trailing zeroes.

Possible Modifications

You might like to add a dozen or so more FP... subroutines, and hand-compile your favorite Applesoft programs into machine language. You might want to revise the FP.PRINT.WD subroutine to work from Applesoft using the & statement, or using a CALL. This would give you a very effective way of formatting values. You also might want to make it put the result in an Applesoft string variable, rather than directly printing it. You might want to add a floating dollar sign capability, or comma insertion between every three digits. If you implement any of these, let me know. I would like to print them in future issues of AAL.

```
=================================================================
DOCUMENT :AAL-8111:Articles:Front.Page.txt
=================================================================
```

$1.20

Volume 2 -- Issue 2                                November, 1981

In This Issue...



Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage).  Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)

Things For Sale

Here is an up-to-date list of some of the things which I have that you might need: Notice that the prices on books, diskettes, and bags are below retail.

```
S-C ASSEMBLER II Version 4.0...........................$55.00
Source code on Disk for above assembler................$95.00
Cross Assembler Patches for 6809 (for 4.0 owners.......$20.00
Cross Assembler for 6800/6801/6802 (for 4.0 owners)....$22.50
Quarterly Disk #1 (source code from Oct 80 - Dec 80)...$15.00
Quarterly Disk #2 (source code from Jan 81 - Mar 81)...$15.00
Quarterly Disk #3 (source code from Apr 81 - Jun 81)...$15.00
Quarterly Disk #4 (source code from Jul 81 - Sep 81)...$15.00
Blank Diskettes (Verbatim, with hub rings, no labels,
                 plain white jackets, in cellophane
                 wrapper)................20 disks for $50.00
Zip-lock Bags (2-mil, 6"x9")...............100 bags for $8.50
Zip-lock Bags (2-mil, 9"x12").............100 bags for $13.00
Back Issues of "Apple Assembly Line"...............each $1.20
"Beneath Apple DOS", Don Worth & Peter Lechner.........$18.00
"What's Where in the Apple", William Luebbert..........$14.00
"6502 Assembly Language Programming", Lance Leventhal..$16.00
```

I add shipping charges to orders for books and bags. If you are in Texas, remember to add 5% sales tax on books, disks, and bags. Software isn't taxable in Texas.

Advertising Info

If you have a software or hardware product that you want to sell, you can reach over 500 serious Apple owners by advertising in AAL.  A full page is only $20, and a half page $10.  I print 1000 copies, because many orders for back issues come in.

```
================================================================================
DOCUMENT :AAL-8111:Articles:Loops4Begs.txt
================================================================================
```

Loops

When you want to program repetitive code in, you write a FOR-NEXT loop or an IF loop.  For example, you might write:

```
     10 FOR I = 1 TO 10     or:  10 I=0
     20 ...                      20 I=1+1 : IF I > 10 THEN 100
     30 NEXT I                   30 ...
                                 90 GO TO 20
                                 100
```

How do you do it in assembly language?

Loop Variable in X or Y

One of the simplest kind of loops holds the loop variable in the Y- or X-register, and decrements it once each trip.

```
     LOOP   LDY #10    Loop for Y = 10 to 1
     .1     ...
            DEY
            BNE .1
```

Note that the loop variable is in the Y-reigster, and that it counts from 10 to 1, backwards.  When the DEY opcode changes Y from 1 to 0, the loop terminates.

If you want the loop to execute one more time, with Y=0, change it to this:

```
     LOOP   LDY #10    Loop for Y = 10 to 0
     .1     ...
            DEY
            BPL .1
```

Of course, a loop count of 129 or more would not work with this last example, because Y would look negative after each DEY until the value was less than 128.

If you want the loop variable to run up instead of down, like from 0 to 9, you need to add a comparison at the end of loop:

```
     LOOP   LDY #0     Loop for Y = 0 to 9
     .1     ...
            INY
            CPY #10
            BCC .1     Carry clear if Y < 10
```

All the examples above use the Y-register, but you can do the same thing with the X-register. In fact, using the X-register, you can nest one loop inside another:

```
     LOOPS  LDY #0     FOR Y = 0 TO 9
     .1     LDX #10    FOR X = 10 TO 1 STEP 1
     .2     ...
            DEX
            BNE .2     NEXT X
            ...
            INY
            CPY #10    NEXT Y
            BCC .1
```

Loop Variable on Stack

Sometimes X and Y are needed for other purposes, and so I use the stack to save my loop variable. Also, the step size can be larger than 1.

```
LOOP   LDA #0    FOR VAR=5 TO 15 STEP 3
.1     PHA       SAVE VAR ON STACK
       ...
       PLA       GET VAR FROM STACK
       CLC
       ADC #3    ADD STEP SIZE
       CMP #16
       BCC .1    VAR <= 15
```

In the Apple Monitor ROM there is a double loop using the stack to hold one of the variables. It is used just for a delay loop, with the length of delay depending on the contents of A when you call it. It is at $FCA8.

```
WAIT   SEC
.1     PHA       outer loop
.2     SBC #1    ...inner loop
       BNE .2    ...next
       PLA
       SBC #1
       BNE .1    next
       RTS
```

The outer loop runs from A down to 1, and the inner loop runs from whatever the current value of the outer loop variable is down to 1. The delay time, by the way, is 5*A*A/2 + 27*A/2 + 13 cycles. (A cycle in the Apple II is a little less than one microsecond.)

16-bit Loop Variables

What if you need to run a loop from $1234 to $2345? That is a little trickier, but not too hard:

```
LOOP  LDA #$1234    START AT $1234
```

```
       STA VARL
       LDA /$1234
       STA VARH
.1     ....
       INC VARL      NEXT: ADD 1
       BNE .2
       INC VARH
.2     LDA VARL
       CMP #$2346    COMPARE TO LIMIT
       LDA VARH
       SBC /$2346
       BCC .1        NOT FINISHED
```

A good example of this kind of loop is in the monitor ROMs also. The code for the end of loop incrementing and testing is at $FCB4-$FCC8. The memory move command ("M") at $FE2C-$FE35 uses this.

Conclusion

There are as many variations on the above themes as there are problems and programmers. Look around in the ROMs, and in programs published in AAL and other magazines; try to understand how the loops you find are working, and adapt them to your own needs.

==============================================================================
DOCUMENT :AAL-8111:Articles:PoorMansDisasm.txt
==============================================================================

Poor Man's Disassembler.....................James O. Church

I wanted a quick and cheap way to get machine language code into the S-C Assembler II Version 4.0, via a text file. I didn't need labels or other automatic features like those $25-$30 Two-Pass Disassemblers have. Or at least not badly enough to pay the price and wait for delivery.

There is a fundamental disassembler in the Apple Monitor ROM, which the "L" command invokes. The problems with it are that it only writes on the screen (not on a text file), and it is not in the correct format for the assembler to use. It has too many spaces between the opcode and operand fields, and there is and address rather than a line number at the beginning of each line.

I wrote a program in Applesoft that gets the starting address of the memory you want to disassemble, and then calls on the monitor "L" command as long as you like. The opcode and operand of each disassembled line are packed into a string array until you want to quit. Then you have the option to write the string array on a text file. The program squeezes out the two extra spaces mentioned above, and omits the hex address from each line. In place of the address and blanks which precede the opcode, this program inserts two control-I characters.

Later, when you use EXEC to get the text file into the S-C Assembler II, the first control-I will generate a line number, and the second one will tab over to the opcode column.

To speed it up a little, I wrote a machine language routine to move the second screen line into the string array. I used the last 15 lines of the Field Input Routine from the September, 1981, issue of AAL as a guide. (Thank you, Bob Potts!)

I chose to not use the already overworked "&" way to call my subroutine. Instead I just used CALL 768, followed by the string reference. It works just as well, as far as I'm concerned.

Also, rather than BLOADing such a short little program, I included it as a hexadecimal string inside the Applesoft program. I used an old technique from B. Lam (Call A.P.P.L.E., many moons ago) for passing the hex code to the monitor and thence into memory. (It's all in line 50.)

Line 100 sets up my array for 1280 lines. That's enough for about 2K of code at a time. Plenty. Make it bigger if you like.

Lines 110-120 ask for and process the starting memory address you want. If you type a negative value, I add 65536 to it to make it positive (from 0 thru 65535, rather than -32768 thru 32767). Then I test the range to make sure you ARE in that range.

Line 130 puts the address where the monitor "L" command wants to find it.

The CALL -418 on line 140 disassembles 20 lines. Line 150 shuffles the operand field two spaces left. Then CALL 768A$(X) puts the 11-byte string starting with the first character of the opcode on the second screen line, into A$(X). CALL -912 on line 180 scrolls the screen up one line, so the next line of disassembly is now on the second screen line. The process repeats until 20 lines have been processed.

Then you have the choice to continue or not. If not, you have the option to write A$() on a text file. If you choose to write it on a file, the file is OPENed, DELETEd, OPENed again, and primed for WRITE. Why the DELETE and extra OPEN? So that if the file was already there, it will be replaced with a new one. If a pre-existing file was longer than my new disassembly, the extra old lines would remain in the file.

You know, once the program is in the string array in text form, you could go ahead and scan it for particular addresses in the operand column. Then you could replace them with meaningful symbols. And you could add meaningful labels on lines that are branched to....

[James Church is a special agent for the Northwestern Mutual Life Insurance Agency; he lives in Trumbull, CT. Article ghost-written and program slightly modified by Bob Sander-Cederlof]

```
=============================================================================
DOCUMENT :AAL-8111:DOS3.3:PoorMans.Dsasm.txt
=============================================================================

s(ó:¢10:ñ9:∫"POOR MAN'S DISASSEMBLER":ñ9:∫"-------------------- -
":ñ13:∫"JAMES O. CHURCH":ñ14:∫"SPECIAL AGENT"  2HEX$-"300:20 E3 DF A9
0B 20 52 E4 A0 00 91 83 A5 71 C8 91 83 A5 72 C8 91 83 A2 94 A0 04 A9
0B 20 E2 E5 60 N D823G":ÅI-
1¡„(HEX$):π511»I,Ê(Í(HEX$,I,1))»128:Ç:π72,0:å…1441  dÜA$(1280):X-0t
     nó:¢10:Ñ"START LOCATION IN DECIMAL: ";L$:L-Â(L$):≠L—0ƒL-L»65536â
     x≠L—0ŒLœ65535ƒ110≥  ÇLH-”(LÀ256):LL-L…LH 256:π58,LL:π59,LH√  åJ-
0:ó:å…418,     ñÅI-0¡6:π1176»I,,(1178»I):Ç   †å768A$(X)
™X-X»1:≠Xœ1280ƒ∫"ARRAY FULL":´210/
¥å…912:J-J»1:≠J—20ƒ150Z
æ∫:∫"CONTINUE? (Y/N) ";:æA$:≠A$-"Y"ƒ140d
»ó:¢10•
“∫"DO YOU WANT TO PUT IT IN A FILE? (Y/N) ";:æA$:≠A$—œ"Y"ƒó:Ä¿
‹∫:Ñ"NAME OF FILE: ";F$ÿ
ÊD$-Á(4):∫D$"OPEN"F$
∫D$"DELETE"F$:∫D$"OPEN"F$:∫D$"WRITE"F$#
˙ÅJ-0¡X…1:∫Á(9);Á(9);A$(J):Ç4
∫D$"CLOSE":Ä
```

```
==========================================================================
DOCUMENT :AAL-8111:DOS3.3:S.FrmtPrint.txt
==========================================================================

 1000  *---------------------------------
 1010  *      TEST
 1020  *---------------------------------
 1030  TEST   LDY #10      LOOP 10 TIMES
 1040         JSR FP.LOAD  VAR1 = 1.0
 1050         .DA AS.ONE
 1060         JSR FP.STORE
 1070         .DA VAR1
 1080         JSR FP.LOAD  VAR2 = 10.0
 1090         .DA AS.TEN
 1100         JSR FP.STORE
 1110         .DA VAR2
 1120  .1     JSR FP.LOAD  VAR1=(VAR1+1)/VAR2
 1130         .DA VAR1
 1140         JSR FP.ADD
 1150         JSR AS.ONE
 1160         JSR FP.DIV
 1170         .DA VAR2
 1180         JSR FP.STORE
 1190         .DA VAR1
 1200         JSR FP.LOAD  VAR2=VAR2-1
 1210         .DA VAR2
 1220         JSR FP.SUB
 1230         .DA AS.ONE
 1240         JSR FP.STORE
 1250         .DA VAR2
 1260         JSR FP.PRINT.WD
 1270         .DA VAR1,#8,#3
 1280         JSR FP.PRINT.WD
 1290         .DA VAR1,#19,#4
 1300         JSR MON.BLANKS  3 SPACES
 1310         JSR FP.PRINT
 1320         .DA VAR1
 1330         JSR MON.CROUT  PRINT CARRIAGE RETURN
 1340         DEY          NEXT TRIP AROUND THE LOOP
 1350         BNE .1
 1360         RTS          FINISHED
 1370  VAR1   .BS 5        MY VARIABLES
 1380  VAR2   .BS 5
 1390  *---------------------------------
 1400  *      ARITHMETIC PACKAGE
 1410  *---------------------------------
 1420  AS.FOUT.E  .EQ $9A
 1430  AS.TEMP1   .EQ $93 THRU $97
 1440  AS.TXTPTR  .EQ $B8,B9
 1450  *---------------------------------
 1460  AS.CHRGET  .EQ $00B1
 1470  AS.COUT    .EQ $DB5C
 1480  AS.FSUB    .EQ $E7A7 FAC=ARG-FAC
```

```
1490  AS.FADD    .EQ $E7BE
1500  AS.ONE     .EQ $E913 CONSTANT 1.0
1510  AS.FMUL    .EQ $E97F
1520  AS.TEN     .EQ $EA50 CONSTANT 10.0
1530  AS.FDIVT   .EQ $EA69 DIVIDE ARG BY FAC
1540  AS.MOVFM   .EQ $EAF9
1550  AS.MOV1F   .EQ $EB21
1560  AS.MOVMF   .EQ $EB2B
1570  AS.MOVAF   .EQ $EB63 MOVE FAC TO ARG
1580  AS.FOUT    .EQ $ED34
1590  AS.NEGOP   .EQ $EED0 FAC = -FAC
1600  *--------------------------------
1610  MON.BLANKS .EQ $F948 PRINT 3 BLANKS
1620  MON.CROUT  .EQ $FD8E PRINT CRLF
1630  *--------------------------------
1640  *      JSR FP.LOAD      LOAD VALUE INTO FAC
1650  *      .DA <ADDR OF VALUE>
1660  *--------------------------------
1670  FP.LOAD
1680         JSR GET.ADDR IN Y,X AND Y,A
1690         JSR AS.MOVFM
1700         JMP FP.EXIT
1710  *--------------------------------
1720  *      JSR FP.STORE     STORE FAC
1730  *      .DA <ADDR TO STORE IN>
1740  *--------------------------------
1750  FP.STORE
1760         JSR GET.ADDR IN Y,X AND Y,A
1770         JSR AS.MOVMF
1780         JMP FP.EXIT
1790  *--------------------------------
1800  *      JSR FP.PRINT     PRINT VALUE IN FREE FORMAT
1810  *      .DA <ADDR OF VALUE TO BE PRINTED>
1820  *--------------------------------
1830  FP.PRINT
1840         JSR GET.ADDR
1850         JSR AS.MOVFM
1860         JSR AS.FOUT
1870         LDY #0
1880  .1     LDA $100,Y
1890         BEQ .2
1900         JSR AS.COUT
1910         INY
1920         BNE .1       ...ALWAYS
1930  .2     JMP FP.EXIT
1940  *--------------------------------
1950  *      JSR FP.ADD   FAC = FAC + VALUE
1960  *      .DA <ADDR OF VALUE>
1970  *--------------------------------
1980  FP.ADD JSR GET.ADDR IN Y,X AND Y,A
1990         JSR AS.FADD  FAC=ARG+FAC
2000         JMP FP.EXIT
2010  *--------------------------------
2020  *      JSR FP.SUB   FAC = FAC - VALUE
```

```
2030  *      .DA <ADDR OF VALUE>
2040  *--------------------------------
2050  FP.SUB JSR GET.ADDR
2060         JSR AS.FSUB   FAC=ARG-FAC
2070         JSR AS.NEGOP  FAC=-FAC
2080         JMP FP.EXIT
2090  *--------------------------------
2100  *      JSR FP.MUL   FAC = FAC + VALUE
2110  *      .DA <ADDR OF VALUE>
2120  *--------------------------------
2130  FP.MUL JSR GET.ADDR IN Y,X AND Y,A
2140         JSR AS.FMUL  FAC=ARG*FAC
2150         JMP FP.EXIT
2160  *--------------------------------
2170  *      JSR FP.DIV   FAC = FAC / VALUE
2180  *      .DA <ADDR OF VALUE>
2190  *--------------------------------
2200  FP.DIV JSR GET.ADDR
2210         PHA
2220         TYA
2230         PHA
2240         JSR AS.MOVAF  MOVE FAC TO ARG
2250         PLA
2260         TAY
2270         PLA
2280         JSR AS.MOVFM
2290         JSR AS.FDIVT
2300         JMP FP.EXIT
2310  *--------------------------------
2320  *      JSR FP.PRINT.WD   PRINT VALUE WITH W.D FORMAT
2330  *      .DA <ADDR OF VALUE>,#<W>,#<D>
2340  *          D = # OF DIGITS AFTER DECIMAL POINT
2350  *          W = # OF CHARACTERS IN WHOLE FIELD
2360  *--------------------------------
2370  FP.PRINT.WD
2380         JSR GET.ADDR ADDRESS OF VALUE
2390         JSR AS.MOVFM VALUE INTO FAC
2400         JSR AS.FOUT  CONVERT TO STRING AT $100
2410         JSR GET.ADDR2  (X)=W, (Y)=D
2420         CPX #41      LIMIT FIELD WIDTH TO 40 CHARS
2430         BCC .14
2440         LDX #40
2450  .14    STX W        # CHARACTERS IN WHOLE FIELD
2460         STX WD.GT
2470         CPY W        FORCE D<W
2480         BCC .13
2490         LDY W
2500         DEY
2510  .13    STY D
2520         DEX          COMPUTE W-D-1
2530         TXA
2540         SEC
2550         SBC D
2560         STA W
```

```
2570        LDA AS.FOUT.E  SEE IF E-FORMAT
2580        BEQ .12      NO
2590        JMP E.FORMAT
2600 .12    LDY #0
2610 *--------------------------------
2620 *      SCAN TO "." OR END, DECREMENTING W
2630 *--------------------------------
2640 .1     LDA $100,Y   SCAN TO END OR DECIMAL POINT
2650        BEQ .2       FOUND END, NO DECIMAL POINT
2660        CMP #'.
2670        BEQ .3       FOUND DECIMAL POINT
2680        INY          COUNT STRING LENGTH
2690        DEC W
2700        BPL .1       ...UNLESS TOO MANY DIGITS FOR FIELD
2710        LDA #0
2720        STA W        NEED NO LEADING BLANKS
2730        DEC D        BACK UP D IF POSSIBLE
2740        BPL .1       TRY AGAIN
2750        JMP PRINT.GT OVERFLOW
2760 *--------------------------------
2770 *      APPEND DECIMAL POINT SINCE NONE PRESENT
2780 *--------------------------------
2790 .2     LDA #'.      PUT DECIMAL POINT BACK ON END
2800        STA $100,Y
2810        LDA #0       END OF STRING CHAR
2820        STA $101,Y
2830 *--------------------------------
2840 *      SCAN TO END, DECREMENTING D
2850 *      (PUT EOS AFTER D DIGITS)
2860 *--------------------------------
2870 .3     INY          NEXT CHAR
2880        LDA D
2890        BEQ .5       NO FRACTIONAL DIGITS
2900 .4     LDA $100,Y   COUNT FRACTIONAL DIGITS TO END
2910        BEQ .6       END OF STRING
2920        INY
2930        DEC D
2940        BNE .4       STILL NEED MORE DIGITS
2950 *--------------------------------
2960 .5     LDA #0       MAKE EOS
2970        STA $100,Y
2980        STA D        NEED NO TRAILING ZEROES
2990 *--------------------------------
3000 *      PRINT LEADING BLANKS AS NEEDED
3010 *--------------------------------
3020 .6     JSR LEADING.BLANKS
3030 *--------------------------------
3040 *      PRINT CONVERTED STRING
3050 *--------------------------------
3060 *      COMES HERE WITH (Y)=0
3070 .8     LDA $100,Y
3080        BEQ .9
3090        JSR AS.COUT
3100        INY
```

```
3110        BNE .8       ...ALWAYS
3120 *---------------------------------
3130 *      PRINT TRAILING ZEROES AS NEEDED
3140 *---------------------------------
3150 .9     JSR TRAILING.ZEROES
3160        JMP FP.EXIT
3170 *---------------------------------
3180 *      HANDLE NUMBERS WHICH COME IN E-FORMAT
3190 *---------------------------------
3200 E.FORMAT
3210        LDX #0
3220        LDY #0
3230 .1     LDA $100,Y   SCAN TO "E", CHANGE TO EOS
3240        CMP #'E
3250        BEQ .3
3260        CMP #'.      SHUFFLE DIGITS AFTER "."
3270        BEQ .2       LEFT ONE POSITION
3280        STA $100,X
3290        INX
3300 .2     INY
3310        BNE .1       ...ALWAYS
3320 .3     LDA #0       EOS
3330        STA $100,X
3340 *---------------------------------
3350        LDA AS.FOUT.E  EXP AGAIN
3360        BPL .12      EXP>0
3370        EOR #$FF     -(EXP+1) IS # ZEROES
3380        CMP D        SEE IF MORE THAN WE NEED
3390        BCC .4       NO
3400        LDA D        YES, JUST USE D
3410 .4     TAX
3420 *---------------------------------
3430        JSR LEADING.BLANKS
3440 *---------------------------------
3450        LDA #'.      DECIMAL POINT
3460        JSR AS.COUT
3470 *---------------------------------
3480 .7     LDA #'0      ZEROES
3490        JSR AS.COUT
3500        DEC D        REDUCE DIGIT COUNT
3510        DEX
3520        BNE .7       MORE ZEROES
3530 *---------------------------------
3540        LDY #0
3550        LDA D        HOW MANY DIGITS?
3560        BEQ .9       NONE
3570 .8     LDA $100,Y   GET A DIGIT
3580        BEQ .10      OUT OF DIGITS
3590        JSR AS.COUT
3600        INY
3610        DEC D
3620        BNE .8       MORE
3630 .9     JMP FP.EXIT
3640 *---------------------------------
```

```
.10    JSR TRAILING.ZEROES
       JMP FP.EXIT
*--------------------------------
*      E-FORMAT WITH EXP>0
*--------------------------------
.12    CMP W        SEE IF ENOUGH ROOM
       BCS PRINT.GT FILL FIELD WITH ">"
       TAX
       INX          # DIGITS AND TRAILING ZEROES
       EOR #$FF     -(EXP+1)
       ADC W        COMPUT # LEADING BLANKS
       STA W
       JSR LEADING.BLANKS
.13    LDA $100,Y   PRINT SIGNIFICANT DIGITS
       BEQ .14
       JSR AS.COUT
       DEX
       INY
       BNE .13      ...ALWAYS
.14    LDA D        SAVE TRAILING ZERO CNT
       PHA
       STX D        SET UP ZEROES BEFORE "."
       JSR TRAILING.ZEROES
       PLA          RESTORE REAL TRAILING ZERO CNT
       STA D
       LDA #'.      PRINT DECIMAL POINT
       JSR AS.COUT
       JSR TRAILING.ZEROES
       JMP FP.EXIT
*--------------------------------
*      PRINT (WD.GT) GREATER THAN SIGNS (">")
*--------------------------------
PRINT.GT
       LDA #'>      OVERFLOW
       LDY WD.GT
       JSR PRINT.ACHAR.YTIMES
       JMP FP.EXIT
*--------------------------------
*      OUTPUT (W) LEADING BLANKS
*--------------------------------
LEADING.BLANKS
       LDA #$20     BLANK
       LDY W        # TO PRINT
       JMP PRINT.ACHAR.YTIMES
*--------------------------------
*      OUTPUT (D) TRAILING ZEROES
*--------------------------------
TRAILING.ZEROES
       LDA #'0
       LDY D
* FALL INTO PRINT.ACHAR.YTIMES
*--------------------------------
*      PRINT (Y) REPETITIONS OF (A)
*--------------------------------
```

```
4190  PRINT.ACHAR.YTIMES
4200         BEQ .2       (Y) IS 0, DON'T PRINT ANY
4210  .1     JSR AS.COUT
4220         DEY
4230         BNE .1
4240  .2     RTS
4250  *---------------------------------
4260  GET.ADDR
4270         STA SAVE.A   SAVE A,X,Y REGISTERS
4280         STX SAVE.X
4290         STY SAVE.Y
4300         PLA          SAVE GET.ADDR RETURN ADDRESS
4310         STA RETLO
4320         PLA
4330         STA RETHI
4340         LDA AS.TXTPTR  SAVE APPLESOFT TEXT POINTER
4350         STA SAVE.T
4360         LDA AS.TXTPTR+1
4370         STA SAVE.T+1
4380         PLA          POINT AT BYTES AFTER JSR FP.<WHATEVER>
4390         STA AS.TXTPTR
4400         PLA
4410         STA AS.TXTPTR+1
4420         JSR GET.ADDR2  GET FIRST TWO BYTES AFTER
4430         LDA RETHI    RETURN
4440         PHA
4450         LDA RETLO
4460         PHA
4470         TXA          ADDR ALSO IN Y,A
4480         RTS
4490  *---------------------------------
4500  GET.ADDR2
4510         JSR AS.CHRGET  GET NEXT BYTE IN CALLING SEQUENCE
4520         TAX
4530         JSR AS.CHRGET  GET NEXT BYTE IN CALLING SEQUENCE
4540         TAY
4550         RTS
4560  *---------------------------------
4570  W      .BS 1
4580  D      .BS 1
4590  RETHI  .BS 1
4600  RETLO  .BS 1
4610  SAVE.A .BS 1
4620  SAVE.X .BS 1
4630  SAVE.Y .BS 1
4640  SAVE.T .BS 2        TXTPTR
4650  WD.GT  .BS 1
4660  *---------------------------------
4670  FP.EXIT
4680         LDA AS.TXTPTR+1  GET HIGH BYTE
4690         PHA
4700         LDA AS.TXTPTR    GET LOW BYTE
4710         PHA
4720         LDA SAVE.T
```

```
4730        STA AS.TXTPTR
4740        LDA SAVE.T+1
4750        STA AS.TXTPTR+1
4760        LDA SAVE.A
4770        LDX SAVE.X
4780        LDY SAVE.Y
4790        RTS
```

================================================================
DOCUMENT :AAL-8112:Articles:AS.GotoFromAsm.txt
================================================================

Applesoft GOTO from Assembly Language.........Bob Sander-Cederlof

Bob Potts called the other day with an interesting question. Suppose you want to jump to a particular line (by line number) of an Applesoft program, rather than simply returning from an assembly language program.

For example, I might call an assembly language subroutine at $300 with "CALL 768". After it does its job, the subroutine may decide either to return to the following Applesoft statement by an "RTS" instruction, or to GOTO a particular line number in the program. (Perhaps an error processing subroutine in the Applesoft code.) Can it be done?

Yes, and it is fairly simple. First we need to put the binary value of the line number into locations $50 and $51. Then we must jump to $D944 in the Applesoft ROMs to finish the GOTO operation. Here is the code to jump to line number 1350, for example:

```
GOTO1350 LDA #1350   LOW BYTE OF "1350"
         STA $50
         LDA /1350   HIGH BYTE OF "1350"
         STA $51
         JMP $D955   APPLESOFT GOTO PROCESSOR
```

That's all there is to it!

I wrote a tiny little subroutine to demonstrate that this works. It expects to find the line number in $2FE and $2FF. You can POKE it there before CALLing 768. Here is my subroutine:

   <code here>

Now here is a test program in Applesoft. Can you tell what it will do before you try it? The first two lines poke in the GOTO subroutine. The next five lines call the subroutine for successive values 1000, 2000, 3000 etc. up to 9000. The code in line 10000 jumps back to line 140 to continue the loop. Try it!

```
==============================================================================
DOCUMENT :AAL-8112:Articles:AS.HiRes.Subs.txt
==============================================================================
```

Applesoft Hi-Res Subroutines................Bob Sander-Cederlof

One of the questions I hear the most is "How can I call the Hi-Res subroutines in the Applesoft ROMs?" The basic information about those subroutines has been published (in Apple Orchard, Vol. 1 No. 1), but with an error in the subroutine addresses.

First, some important locations in page zero:

```
$1A,1B  Shape pointer used by DRAW and XDRAW
$1C     Last used color byte
$26,27  Address of byte containing X,Y point
$30     Bit mask for bit in that byte
$E0,E1  X-coordinate (0-279)
$E2     Y-coordinate (0-191)
$E4     Color
$E6     Page ($20 if HGR, $40 if HGR2)
$E7     SCALE= value
$E8,E9  Address of beginning of shape table
$EA     Collision counter
$F9     ROT= value
```

The software uses some other page zero variables, but I am not too clear yet on their purpose.

Now here are the major entry points:

```
HGR2     $F3D8   Initialize and clear hi-res page 2.

HGR      $F3E2   Initialize and clear hi-res page 1.

HCLR     $F3F2   Clear the current hi-res screen to black.

BKGND    $F3F6   Clear the current hi-res screen to the
                 last plotted color (from ($1C).

HPOSN    $F411   Positions the hi-res cursor without
                 plotting a point.
                 Enter with (A) = Y-coordinate, and
                 (Y,X) = X-coordinate.

HPLOT    $F457   Calls HPOSN and tries to plot a dot at
                 the cursor's position.  If you are
                 trying to plot a non-white color at
                 a complementary color position, no
                 dot will be plotted.

HLIN     $F53A   Draws a line from the last plotted
                 point or line destination to:
```

```
                 (X,A) = X-coordinate, and
                 (Y) = Y-coordinate.

HFIND    $F5CB   Converts the hi-res coursor's position
                 back to X- and Y-coordinates; stores
                 X-coordinate at $E0,E1 and Y-coordinate
                 at $E2.

DRAW     $F601   Draws a shape.  Enter with (Y,X) = the
                 address of the shape table, and (A) =
                 the rotation factor.  Uses the current
                 color.

XDRAW    $F65D   Draws a shape by inverting the existing
                 color of the dots the shape draws over.
                 Same entry parameters as DRAW.

SETHCOL  $F6EC   Set the hi-res color to (X), where (X)
                 must be between 0 and 7.
```

I wrote a sample demonstration program of the hi-res subroutines. First, here is an Applesoft version. Note that it first sets the whole screen to a particular color, and then draws a series of nested squares in a complementary color. Since it is nice and short, why don't you type it in and try it?

<code here for applesoft version>

Now here is the assembly language program for the same task. It seemed to run about twice as fast as the Applesoft version, but I didn't use the stopwatch on it.

================================================================
DOCUMENT :AAL-8112:Articles:AS.LineEditAID.txt
================================================================

Applesoft Line Editing Aid..................Sandy Mossberg

[ Sandy is an M.D. in Port Chester, New York.  You have probably seen his excellent articles and programs in NIBBLE. ]

The following program is a developmental tool for line-editing Applesoft programs.  It places the line you specify at the top of the screen, ready to be cursor edited.  The line is displayed without added blanks at the end of each screen line, which can mess up editing of PRINT statements.  Obviously, adding Konzen-like PLE features would make it much nicer, but that's a story for another day.

The program loads at the ever-popular $300.  If you BRUN it, or BLOAD and CALL768, it installs itself.  To use it, type a slash and a line number.  For example, to edit line 150, type "/150" and a carriage return.  The screen will be cleared and line 150 displayed on the top.  The cursor will be placed over the first character, and you will be ready to edit it with standard cursor-editing techniques.  (If there is no line 150 in memory, the bell will ring instead.)

Several aspects of the code should be of interest to assembly language programmers:

(1) As noted in AAL of 9/81, the CHRGET/CHRGOT routine screens for the command character (a slash).  This technique permits concurrent use of an amper-utility.  The KSW hook could be employed as yet another filter, making a trio of vectors operative.

(2) To allow "illegal" line numbers (64000-65535) to be accessed, the LINGET routine is replaced by calls to FRMEVL and GETADR (see Lines 1800-1810).

(3) The de-parsing secton (see Lines 2030-2500) is an offspring of Applesoft's LIST routine, modified to pring a single program line rather than an entire listing.  I also eliminated the code which adds those extra blanks in the middle of quoted strings which take more than one screen line to LIST.  To me it seems pretty neat!

Since I did not make any test to determine whether or not the program is RUNning at the time the slash is trapped in my filter, you have to be careful about using the slash character in REM statements.  For example, "REM /150" will clear the screen and list line 150 at the top before proceeding.  Other combinations of "/" in REM's may blow up. Also, typing "/" when Applesoft is executing an INPUT statement is now dangerous.  Anyone know how to fix this?

==============================================================
DOCUMENT :AAL-8112:Articles:ASCII.Mon.Dump.txt
==============================================================

Adding ASCII to Apple Monitor Dump....Bob Sander-Cederlof

Peter Bartlett (subscriber in Chicago, IL) sent me some source code for patches to the Apple Monitor ROM. Of course, patching a ROM may be a little too much hardware work, but if you have a 16K RAM card you can put the revised monitor up there. The space needed for the patch is stolen from the cassette I/O command, so if you install this patch you will lose cassette I/O.

Peter's patches add the ASCII dump to the Apple Monitor's hex dump. That is, when I type a command like "800.87F" in the monitor, it will not only print out the hex values, but also the ASCII values of each byte. I modified his patches a little, to shorten the code to the following:

<code here>

These patches will work with either the old monitor ROM, or the Autostart ROM. The JSR PATCH line goes right into the hex dump program, over the top of a JSR COUT that printed a space. That space is normally printed right before the next byte value is printed in hex. The address of the next byte is kept in A1L,A1H ($3C,3D). The Y-register has 0 in it.

The main patch subroutine is stored on top of part of the cassette tape I/O, at $FC99; it begins with the JSR COUT that was covered up at $FDB8. Lines 1150,1160 pick up the byte to be displayed and save it on the stack. Lines 1170-1210 compute the horizontal postition for poking the byte on the screen. The low-order three bits of the memory address determine which column will be used, from column 31 through 38. Lines 1220,1230 retrieve the byte from the stack and store it into the screen buffer. Lines 1240,1250 restore Y=0 and return to the hex dump subroutine.

Note that this patch does not "print" the ASCII codes on the screen; it "pokes" them. Therefore if your printer is on, the printed copy will only contain the hex dump. The ASCII codes will only appear on the screen.

How do you patch the RAM card version of the monitor? Here's how I did it:

1) Load the language card using your DOS 3.3 Master Disk, or whatever technique you like to use.

2) Turn on the language that is in the card (using FP or INT).

3) BSAVE MONITOR,A$F800,L$800.

4)  BRUN ASMDISK 4.0

5)  BLOAD MONITOR,A$800

6)  Enter the source code for the patches and assemble them with the ASM command.  This will patch the monitor copy which you loaded at A$800 in step 5.

7)  Type "$C081 C081" to write enable the language card.

8)  Type "$F800<800.FFFM" to move the patched monitor into the real monitor space.

9)  Type "BSAVE <your file name>,A$D000,L$3000" to save the combined language and monitor for later loading into the language card.

If you really do want to burn a new monitor ROM, follow the instructions with your ROM Burner.

================================================================================
DOCUMENT :AAL-8112:Articles:Excel.9.Review.txt
================================================================================

EXCEL-9: A 6809 Card with FLEX..............Bob Sander-Cederlof

For the last month and a half I have been working with a fantastic new device: the EXCEL-9 from Seikou Electronics in Japan.  The EXCEL-9 contains a 6809E CPU, 8K bytes of ROM, and an interval timer.  The 8K ROM contains a monitor with 35 commands (including mini-assembler anddis-assembler commands).  The introductory price of $399.95 includes the FLEX Operating System from Technical Systems Consultants (TSC), with utilities, text editor, and macro assembler.

The board will soon be appearing in your local computer stores, courtesy of ESD Laboratories.  I worked with them to translate the excellent reference manual into English.  (That explains how I obtained one of the boards so early.)

EXCEL-9 has a lot of unique features that should make it a very popular board:

*  An on-board interval timer (with 24 intervals from 2 microseconds to 16 seconds) can be used from both the 6809 and 6502.

*  Built-in linkage routines for calling 6809 subroutines from Applesoft, Integer BASIC, or 6502 machine language.  You can also call 6502 routines and even DOS 3.3 commands from 6809 programs.

*  Option of using standard Apple intelligent interfaces with 6502 firmware, or of using new cards with 6809 firmware.

*  Memory Mapping that supports the FLEX operating system.  Future option to add external memory to EXCEL-9, allowing full-speed multiprocessing.

I intend to handle these boards.  You can order them from me now, but please allow a while for delivery.  The documentation is ready for the printer, but not yet printed.

================================================================================
DOCUMENT :AAL-8112:Articles:Front.Page.txt
================================================================================

$1.20

Volume 2 -- Issue 3                                      December, 1981

In This Issue...



Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage).  Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)

==============================================================================
DOCUMENT :AAL-8112:Articles:FstrStringInput.txt
==============================================================================

Improved Applesoft Fast String Input....Bob Sander-Cederlof

In the April 1981 issue of AAL I printed a subroutine to read a line from the keyboard or a text file into an Applesoft string. The original version had a minor flaw (or major, if you happened to run into it): it left the high-order bit on in each byte, so that Applesoft could not compare them properly with strings from other sources. I printed a correction in a later issue, which stripped off the leading bit from each byte before putting it in the string.

Now Sherm Ostrowsky (from Goleta, California) has pointed out a more elegant solution. He uses a subroutine inside Applesoft that reads a line, terminates it with hex 00, and strips off the leading bit from each byte. The subroutine starts at $D52C. The only thing it doesn't do that we need is give us the length of the input line. Here is a commented listing of it.

    <D52E listing here>

Since $D52C stores $80 (null) in the prompt character, you might want to load the X-register with $87 (bell) and enter at $D52E instead.

Since the subroutine returns with $FF in the X-register, and we need the length of the input line instead, we can use the following code to get the line length in X:

```
       JSR $D52C
.1     INX
       LDA $200,X
       BNE .1
```

Here is a new version, then, of my fast string input subroutine:

   <subroutine here>

Here is how you might use it from an Applesoft program, to read a series of lines from a file:

```
100 D$ = CHR$ (4)
110 PRINT D$"BLOAD B.FAST READ"
120 POKE 1013,76 : POKE 1014,0 : POKE 1015,3
210 PRINT D$"OPEN MY.FILE"
220 PRINT D$"READ MY.FILE"
230 FOR I = 1 TO 10
240 & GET A$(I)
250 NEXT I
```

Note that the subroutine is fully relocatable. Since there are no internal JMP's or JSR's, and no internal variables, you can load the program anywhere it will fit and run it without any modifications. Just be sure to change line 120 above to POKE the correct address in 1014 and 1015.

==========================================================================
DOCUMENT :AAL-8112:Articles:Hex.Const.AS.txt
==========================================================================

Hex Constants in Applesoft...................David Bartley

Coding in BASIC has several frustrations for the assembly language programmer. One small but constant irritant for me has been the inability to directly specify hexadecimal values in Applesoft statements or in response to an INPUT command. I finally decided to do something about it when I read Bob Sander-Cederlof's article on the CHRGET routine in the September Apple Assembly Line. The result is the short program shown here.

My goal was to be able to enter a hex constant, defined as a "$" followed by one or more hex digits, anywhere Applesoft would allow an integer constant to appear. I nearly succeeded -- I'll discuss the exceptions a little later. I now can write statements like:

```
       100 FOR I = $0 TO $FF
       110 INPUT X,Y
       120 Z(I) = $100*X + Y - $3DEF
```

The responses to the INPUT statement may also be hex constants. Values may range from -$FFFF (-65535) to $FFFF (65535); the left-most bit is not considered a sign bit.

My program is set up by BRUN-ning the object file XB.A/S HEX CONSTANTS (see line 1010). Initialization consists of modifying the Applesoft CHRGET subroutine to branch into new code starting at line 1400. As you may recall, CHRGET is used by the BASIC interpreter to fetch characters and tokens from the program text of keyboard when a program is executing. The new CHRGET code watches for a "$" character; when one is found, it scans forward until it hits a character which is not a hex digit, converting to a binary value (in VAL) on the fly.

Variable IDX serves two purposes. It is normally negative, signifying that characters are to be fetched without special action until a "$" is encountered. After a hex constant is found and converted to a binary value, IDX becomes a positive index into a power-of-ten table to facilitate converting VAL to a decimal value. Each subsequent call to CHRGET then returns a successive character of the decimal integer representation of VAL until IDX becomes -1, the entire value has been transformed from hex to decimal, and the normal mode is restored.

There are, of course, several complications. One is the BASIC "DEF" command, which happens to consist of a string of hex digits. Applesoft therefore parses a constant like "$3DEF" as the ASCII characters "$" and "3" followed by the DEF token (hex 88). Lines 1760 to 1840 take care of that.

A more serious complication is the existence of a frequently used alternate entry point to CHRGET callled CHRGOT. CHRGOT is called to fetch the previous item from the text rather than the next one. It seems that numeric constants are parsed from several places within the Applesoft interpreter, with some using CHRGOT and others not. When I fixed things up so CHRGOT would work for inline constants and the INPUT command, it no longer worked for values in DATA statements (or for hex line numbers, for that matter!)

The trick that makes CHRGOT work (most of the time) is to back up TXTPTR and then return a leading zero to start off the converted decimal value. The zero causes no consternation for the parts of the interpreter that see it and is not missed by those that don't. If CHRGOT is not called, however, TXTPTR should not be backed up. You can't win!

I hope others will be able to make use of this routine -- better, that someone will overcome the problem with DATA statement values. It has been quite valuable to me as it is, as well as quite an education in understanding the inner workings of the Applesoft interpreter.

```
=======================================================================
DOCUMENT :AAL-8112:Articles:Price.List.txt
=======================================================================

S-C ASSEMBLER II Version 4.0...................................$55.00
    Includes Manual, Diskette with Assembler and sample
    source programs, and Quick Reference Card.

Source code of Version 4.0 on disk.............................$95.00
    Fully commented, easy to understand and modify to
    your own tastes.

Cross Assembler Patches for 6809...............................$20.00
    Requires possession of Version 4.0.  Enables you to
    develop programs for the Motorola 6809 CPU.  (The
    MILL from Stellation, EXCEL-9 from ESD Laboratories,
    or the Radio Shack Color Computer.)

Cross Assembler for 6800.......................................$22.50
    Requires possession of Version 4.0.  Enables you to
    develop programs for the Motorola 6800, 6801, and
    6802 CPUs.

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three
    issues of "Apple Assembly Line", to save you lots
    of typing and testing time.
    QD#1:  Oct - Dec 1980    QD#4:  Jul - Sep 1981
    QD#2:  Jan - Mar 1981    QD#5:  Oct - Dec 1981
    QD#3:  Apr - Jun 1981

Double Precision Floating Point for Applesoft..................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes subroutines for standard math functions.

Some Simple Integer BASIC Games................................$10.00
    Includes 4x4x4 tic-tac-toe, lo-res space war, lo-res
    jig-saw puzzle, and mastermind.

Blank Diskettes.............................package of 20 for $50.00
    Verbatim Datalife, with hub rings, no labels, in plain
    white jackets, in cellophane wrapper.

Lower-Case Display Encoder ROM.................................$25.00
    Works only Revision level 7 Apples.  Replaces the
    encoder ROM.  Comes with instructions.

Diskette Mailing Protectors.....................10-99:  40 cents each
                                             100 or more:  25 cents each
    Corrugated folder specially designed for mailing
    mini-floppy diskettes.  Fits in standard 6x9-inch
    envelope.  (Envelopes 5-cents each, if you need them.)
```

```
Zip-Lock Bags (2-mil, 6"x9")............................100 for $8.50
              (2-mil, 9"x12")..........................100 for $13.00

Books, Books, Books......................compare our discount prices!
    "Beneath Apple DOS", Worth & Lechner.............($19.95)  $18.00
    "What's Where in the Apple", William Leubert.....($14.95)  $14.00
    "6502 Assembly Language Programming", Leventhal..($16.99)  $16.00
    "Apple Assembly Language", Don & Kurt Inman......($12.95)  $12.00

      *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
      *** (214) 324-2050    We take Master Charge and VISA ***
```

```
=============================================================================
DOCUMENT :AAL-8112:DOS3.3:AS.DEMO.HI.RES.txt
=============================================================================
```

( DTC removed -- lots of garbage characters )

```
==============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.ASoft.Inline.txt
==============================================================================

 1000  *---------------------------------
 1010  *      APPLESOFT LINE INPUT SUBROUTINE
 1020  *---------------------------------
 1030         .OR $D52C
 1040         .TA $82C
 1050  *---------------------------------
 1060  MON.PROMPT .EQ $33
 1070  MON.RDLINE .EQ $FD6A
 1080  BUFFER     .EQ $200
 1090  *---------------------------------
 1100  AS.INLINE
 1110         LDX #$80     NULL CHARACTER
 1120  INLIN2 STX MON.PROMPT  FOR THE PROMPT CHARACTER
 1130         JSR MON.RDLINE  READ A LINE INTO BUFFER
 1140         CPX #239     TRUNCATE TO 239 CHARACTERS
 1150         BCC .1
 1160         LDX #239
 1170  .1     LDA #0       MARK END OF LINE WITH $00
 1180         STA BUFFER,X
 1190         TXA          # REAL CHARS IN LINE
 1200         BEQ .3       EMPTY LINE
 1210  .2     LDA BUFFER-1,X  STRIP OFF ALL SIGN BITS
 1220         AND #$7F
 1230         STA BUFFER-1,X
 1240         DEX
 1250         BNE .2
 1260  .3     LDA #0
 1270         LDX #BUFFER-1
 1280         LDY /BUFFER-1
 1290         RTS
```

```
==============================================================
DOCUMENT :AAL-8112:DOS3.3:S.Fast.Read.txt
==============================================================
```

```
 1000  *---------------------------------
 1010  *      FAST STRING INPUT ROUTINE
 1020  *      &GET <STRING VARIABLE>
 1030  *      ACCEPTS ANY CHARACTER, UNLIKE NORMAL INPUT
 1040  *---------------------------------
 1050         .OR $300
 1060         .TF B.FAST READ
 1070  *---------------------------------
 1080  AS.CHRGET  .EQ $00B1
 1090  AS.SYNERR  .EQ $DEC9
 1100  AS.INLINE  .EQ $D52C
 1110  AS.PTRGET  .EQ $DFE3
 1120  AS.GETSPA  .EQ $E452
 1130  AS.MOVSTR  .EQ $E5E2
 1140  *---------------------------------
 1150  ADDR       .EQ $71 AND 72
 1160  PNTR       .EQ $83 AND 84
 1170  LENGTH     .EQ $9D
 1180  BUFFER     .EQ $200
 1190  *---------------------------------
 1200  GET    CMP #$BE     "GET" TOKEN
 1210         BEQ .1       YES
 1220         JMP AS.SYNERR  SORRY...
 1230  .1     JSR AS.CHRGET SET UP THE FOLLOWING CHARACTER
 1240         JSR AS.PTRGET FIND THE STRING VARIABLE POINTER
 1250         JSR AS.INLINE READ A LINE INTO BUFFER
 1260  .2     INX          COMPUTE THE LENGTH OF THE LINE
 1270         LDA BUFFER,X
 1280         BNE .2       NOT AT END OF LINE YET
 1290         STX LENGTH   SAVE LINE LENGTH
 1300         TXA
 1310         JSR AS.GETSPA  GET SPACE IN STRING AREA
 1320         LDY #0       SET UP STRING VARIABLE POINTER
 1330         STA (PNTR),Y   LENGTH
 1340         INY
 1350         LDA ADDR
 1360         STA (PNTR),Y   ADDRESS (LO-BYTE)
 1370         INY
 1380         LDA ADDR+1
 1390         STA (PNTR),Y   ADDRESS (HI-BYTE)
 1400         LDY /BUFFER    SET UP TO COPY STRING DATA
 1410         LDX #BUFFER    INTO STRING AREA
 1420         LDA LENGTH
 1430         JMP AS.MOVSTR  COPY IT NOW, AND RETURN
```

==============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.GOTO.txt
==============================================================================

```
 1000  *--------------------------------
 1010  *      GO TO <LINE #>
 1020  *      POKE THE LINE # INTO 766,767
 1030  *      AND CALL768 TO GO TO IT
 1040  *--------------------------------
 1050         .OR $300
 1060  GOTO   LDA $2FE
 1070         STA $50
 1080         LDA $2FF
 1090         STA $51
 1100         JMP $D944
```

```
==============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.HEX.CONSTANTS.txt
==============================================================================

 1000        .OR $300
 1010        .TF XB.A/S HEX CONSTANTS
 1020  *--------------------------------
 1030  *
 1040  *      APPLESOFT HEX CONSTANTS
 1050  *
 1060  *      WRITTEN BY DAVID H. BARTLEY
 1070  *      AUSTIN, TEXAS -- AUGUST 1981
 1080  *
 1090  *      TO INITIALIZE:
 1100  *          BRUN THIS PROGRAM (XB.A/S HEX CONSTANTS)
 1110  *
 1120  *      TO USE:
 1130  *          PRECEDE HEX CONSTANTS
 1140  *          WITH A "$" CHARACTER
 1150  *
 1160  *--------------------------------
 1170  BASIC  .EQ $E003    SOFT RE-ENTRY
 1180  CHRGET .EQ $00B1    A/S CHRGET RTN
 1190  CHRGOT .EQ $00B7    A/S CHRGOT RTN
 1200  CHRCHK .EQ CHRGOT+3
 1210  TXTPTR .EQ $B8      A/S TEXT PTR
 1220  OVERR  .EQ $E8D5    OVERFLOW ERROR
 1230  TEMP   .EQ $FC      16-BIT TEMPORARY
 1240  VAL    .EQ $FE      16-BIT VALUE
 1250  *--------------------------------
 1260  INIT
 1270         LDA #$4C     MODIFY CHRGET
 1280         STA CHRGET   TO CALL HERE
 1290         LDA #NEW.CHRGET
 1300         STA CHRGET+1
 1310         LDA /NEW.CHRGET
 1320         STA CHRGET+2
 1330         JMP BASIC    RETURN TO A/S
 1340  NEXTCH
 1350         INC TXTPTR   DUPLICATE THE
 1360         BNE .10      OLD CHRGET
 1370         INC TXTPTR+1
 1380  .10    JMP CHRGOT
 1390  *--------------------------------
 1400  NEW.CHRGET
 1410         BIT IDX      NORMAL MODE?
 1420         BPL .60      -NO
 1430  *
 1440  * CHECK FOR "$" AS NEXT CHARACTER
 1450  *
 1460         JSR NEXTCH   GET CHAR
 1470         CMP #$24    "$"?
 1480         BNE .50      -NO, RETURN IT
```

```
1490  .10
1500  * PARSE A HEX NUMBER AND CONVERT
1510  * IT TO A BINARY VALUE
1520  *
1530         LDA #0
1540         STA VAL      VAL = 0
1550         STA VAL+1
1560         LDA #4       INDEX TO POWER
1570         STA IDX
1580  .20
1590         JSR NEXTCH   GET HEX DIGIT
1600         BEQ .40      -EOL OR ":"
1610         SEC
1620         SBC #$30     CHECK FOR DIGIT
1630         BMI .35      -NOT A DIGIT
1640         CMP #10
1650         BCC .30      -OK (0-9)
1660         SBC #17
1670         BMI .40      -NOT A DIGIT
1680         CMP #6
1690         BCS .40      -NOT A DIGIT
1700         ADC #10
1710  .30    JSR ASL4     MULT VAL BY 16
1720         ORA VAL      ADD NEW DIGIT
1730         STA VAL
1740         JMP .20
1750  .35
1760         CMP #$88     "DEF" TOKEN?
1770         BNE .40      -NO
1780         JSR ASL4     -YES
1790         LDA VAL
1800         ORA #$0D     ASL BY 12 AND
1810         STA VAL+1    ADD $0DEF
1820         LDA #$EF
1830         STA VAL
1840         BNE .20      (ALWAYS)
1850  .40
1860         LDA TXTPTR   BACK UP THE
1870         BNE .41
1880         DEC TXTPTR+1
1890  .41    DEC TXTPTR
1900         LDA TXTPTR   SAVE TXTPTR
1910         STA TEMP     IN CASE IS IS
1920         LDA TXTPTR+1 DECREMENTED
1930         STA TEMP+1   BY THE CALLER
1940  *
1950         LDA #$30     ASCII "0"
1960  .50    JMP CHRCHK   -EXIT
1970  .60
1980  * CONVERT BINARY VALUE TO DECIMAL
1990  * AND RETURN THE NEXT ASCII DIGIT
2000  *
2010         LDA TEMP     FIX ANY ATTEMPT
2020         STA TXTPTR   TO DECREMENT
```

```
2030        LDA TEMP+1   TXTPTR
2040        STA TXTPTR+1
2050        STX SAVE.X
2060        LDX IDX      POWER OF TEN
2070        DEC IDX
2080        LDA #$30     ASCII "0"
2090 .70
2100        PHA          ASCII DIGIT
2110        LDA VAL
2120        CMP LO.TENS,X  SET CARRY
2130        LDA VAL+1
2140        SBC HI.TENS,X
2150        BCC .80      -EXIT LOOP
2160        STA VAL+1
2170        LDA VAL
2180        SBC LO.TENS,X
2190        STA VAL
2200        PLA          ASCII DIGIT
2210        CLC
2220        ADC #1       INCREMENT IT
2230        BNE .70      -LOOP
2240 .80
2250        PLA          ASCII DIGIT
2260        LDX SAVE.X
2270 .90
2280        JMP CHRCHK   PROCESS IT
2290 *---------------------------------
2300 ASL4   JSR ASL2     ASL VAL BY 4
2310 ASL2   JSR ASL1     ASL VAL BY 2
2320 ASL1   ASL VAL      ASL VAL BY 1
2330        ROL VAL+1
2340        BCS OVFLOW   -OVERFLOW ERROR
2350        RTS          -EXIT
2360 OVFLOW
2370        JMP OVERR    REPORT OVERFLOW
2380 *---------------------------------
2390 LO.TENS .DA #1
2400        .DA #10
2410        .DA #100
2420        .DA #1000
2430        .DA #10000
2440 HI.TENS .DA /1
2450        .DA /10
2460        .DA /100
2470        .DA /1000
2480        .DA /10000
2490 IDX    .DA #$FF     TABLE INDEX
2500 SAVE.X .DA #0       SAVE X-REG
2510 *---------------------------------
2520 ZZZZZZ .EN
```

```
==============================================================
DOCUMENT :AAL-8112:DOS3.3:S.HI.RES.DEMO.txt
==============================================================

 1000  *---------------------------------
 1010  *      SAMPLE PLOTTING PROGRAM
 1020  *---------------------------------
 1030  AS.LASTCLR .EQ $1C
 1040  *---------------------------------
 1050  AS.HGR2    .EQ $F3D8 SET UP HI-RES PAGE 2
 1060  AS.HCLR    .EQ $F3F2 CLEAR HI-RES SCREEN
 1070  AS.BKGND   .EQ $F3F6 CLEAR HI-RES SCREEN TO LAST COLOR
 1080  AS.HPOSN   .EQ $F411 MOVE CURSOR TO (Y,X),(A)
 1090  AS.HPLOT   .EQ $F457 PLOT A DOT AT (Y,X),(A)
 1100  AS.HLIN    .EQ $F53A DRAW A LINE FROM LAST POINT TO (X,A),(Y)
 1110  AS.SETHCOL .EQ $F6EC SET HI-RES COLOR
 1120  MON.TEXT   .EQ $FB2F
 1130  *---------------------------------
 1140  HI.RES.DEMO
 1150         JSR AS.HGR2
 1160         LDX #0       FOR COLOR = 0 TO 7
 1170  .1     STX COLOR
 1180         JSR AS.SETHCOL
 1190         STA AS.LASTCLR
 1200         JSR AS.BKGND   CLEAR SCREEN TO SOLID COLOR
 1210         LDA COLOR
 1220         EOR #7       COMPLEMENTARY COLOR
 1230         TAX
 1240         JSR AS.SETHCOL
 1250         JSR DRAW.SQUARE
 1260         LDX COLOR      NEXT COLOR
 1270         INX
 1280         CPX #8
 1290         BCC .1
 1300         JSR MON.TEXT
 1310         RTS
 1320  *---------------------------------
 1330  DRAW.SQUARE
 1340         LDA #10      FOR SIZE=10 TO 190 STEP 10
 1350  .1     STA SIZE
 1360         LSR          SIZE/2
 1370         STA SIZE2
 1380         LDA #0
 1390         STA XSTART+1
 1400         STA XSTOP+1
 1410         SEC          XSTART=140-SIZE/2
 1420         LDA #140
 1430         SBC SIZE2
 1440         STA XSTART
 1450         CLC          XSTOP=XSTART+SIZE
 1460         ADC SIZE
 1470         STA XSTOP
 1480         SEC
```

```
1490        LDA #95      YSTART=95-SIZE/2
1500        SBC SIZE2
1510        STA YSTART
1520        CLC          YSTOP=YSTART+SIZE
1530        ADC SIZE
1540        STA YSTOP
1550        LDY XSTART+1 HPLOT XSTART,YSTART
1560        LDX XSTART
1570        LDA YSTART
1580        JSR AS.HPLOT
1590        LDX XSTOP+1    TO XSTOP,YSTART
1600        LDA XSTOP
1610        LDY YSTART
1620        JSR AS.HLIN
1630        LDX XSTOP+1    TO XSTOP,YSTOP
1640        LDA XSTOP
1650        LDY YSTOP
1660        JSR AS.HLIN
1670        LDX XSTART+1   TO XSTART,YSTOP
1680        LDA XSTART
1690        LDY YSTOP
1700        JSR AS.HLIN
1710        LDX XSTART+1   TO XSTART,YSTART
1720        LDA XSTART
1730        LDY YSTART
1740        JSR AS.HLIN
1750        CLC
1760        LDA SIZE     NEXT SIZE
1770        ADC #10
1780        CMP #191
1790        BCC .1
1800  DELAY.LOOP
1810        LDY #0       DELAY LOOP SO WE CAN SEE IT
1820  .1    LDX #0
1830  .2    DEX
1840        BNE .2
1850        LDA $C030    AND HEAR IT
1860        DEY
1870        BNE .1
1880        RTS
1890  *--------------------------------
1900  COLOR  .BS 1
1910  SIZE   .BS 1
1920  SIZE2  .BS 1
1930  XSTART .BS 2
1940  YSTART .BS 1
1950  XSTOP  .BS 2
1960  YSTOP  .BS 1
```

```
==============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.INTEGER.INPUT.txt
==============================================================================

 1000  *--------------------------------
 1010  *      PROGRAM TO INPUT AN INTEGER FROM
 1020  *      0-65535, AND PUT IT IN $50,51
 1030  *
 1040  *      BY PETER MEYER, 10/24/81
 1050  *      MAY BE FREELY USED WITH ACKNOWLEDGEMENT
 1060  *--------------------------------
 1070  *      CALL:   JSR GET.INTEGER.INTO.LINNUM
 1080  *    RETURN:   INTEGER VALUE IN LINNUM ($50,51)
 1090  *              AND CARRY CLEAR,
 1100  *              OR CARRY SET IF VALUE NEGATIVE
 1110  *                   OR TOO LARGE, OR HAS A
 1120  *                   LETTER IN IT.
 1130  *--------------------------------
 1140  LINNUM .EQ $50,51
 1150  FACEXP .EQ $9D
 1160  FACMO  .EQ $A0
 1170  FACLO  .EQ $A1
 1180  FACSGN .EQ $A2
 1190  TXTPTR .EQ $B8,B9
 1200  BUFFER .EQ $200
 1210  *--------------------------------
 1220  CHRGOT .EQ $B7
 1230  GDBUFS .EQ $D539
 1240  QINT   .EQ $EBF2
 1250  FIN    .EQ $EC4A
 1260  NXTCHR .EQ $FD75
 1270  *--------------------------------
 1280         .OR $300     (BUT MAY BE LOADED ANYWHERE)
 1290  *--------------------------------
 1300  GET.INTEGER.INTO.LINNUM
 1310         LDX #0
 1320         JSR NXTCHR
 1330         TXA          CHECK FOR NULL ENTRY
 1340         BEQ .2       NULL
 1350  *--------------------------------
 1360  *   CHECK FOR ALPHA INPUT
 1370  *   AND ALSO WEED OUT ENTRIES SUCH AS
 1380  *   "1E99" AND "99999...." WHICH WOULD
 1390  *   CAUSE OVERFLOW.
 1400  *--------------------------------
 1410         PHA          SAVE LENGTH
 1420         JSR GDBUFS
 1430         PLA          RETRIEVE LENGTH
 1440         CMP #36
 1450         BCS .2
 1460         TAX
 1470         DEX
 1480  .1     LDA BUFFER,X
```

```
1490        CMP #'A
1500        BCS .2
1510        DEX
1520        BPL .1
1530 *   GET NUMBER INTO FAC
1540        LDA #BUFFER
1550        LDY /BUFFER
1560        STA TXTPTR
1570        STY TXTPTR+1
1580        JSR CHRGOT
1590        JSR FIN
1595        .PG
1600 *--------------------------------
1610 *   CHECK IF NUMBER IS NEGATIVE
1620 *--------------------------------
1630        LDA FACSGN
1640        BPL .3
1650 .2     SEC          NUMBER IS TOO LARGE, NEGATIVE
1660        RTS          OR HAS A LETTER IN IT.
1670 *--------------------------------
1680 *   CHECK IF NUMBER IS TOO LARGE
1690 *--------------------------------
1700 .3     LDA FACEXP
1710        CMP #$91
1720        BCS .4       TOO LARGE
1730 *--------------------------------
1740 *   PLACE IN LINNUM
1750 *--------------------------------
1760        JSR QINT     CONVERT TO INTEGER
1770        LDA FACLO
1780        LDY FACMO
1790        STA LINNUM
1800        STY LINNUM+1
1810        CLC          SIGNAL GOOD VALUE
1820 .4     RTS
```

```
==============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.Mossberg.LE.txt
==============================================================================

 1000  *--------------------------------
 1010  *          LINE.EDIT
 1020  *
 1030  *      BY SANDY MOSSBERG
 1040  *
 1050  *   COMMERCIAL RIGHTS RESERVED
 1060  *
 1070  *--------------------------------
 1080  * 1.PACKS PROGRAM LINE FOR EASY EDITING.
 1090  *
 1100  * 2.USES CHRGET/CHRGOT FILTER ROUTINE NOTED IN AAL 9/81.
 1110  *
 1120  * 3.CHARACTER OUTPUT ROUTINE MODIFIED FROM APSOFT ROM
 1130  *   CODE (LIST, $D6A5-$D765).
 1140  *
 1150  * 4.INSTALLATION AND USE:
 1160  *    (A) BRUN LINE.EDIT.
 1170  *    (B) COMMAND "/LINENUMBER" PRODUCES PACKED LINE AT
 1180  *        TOP OF SCREEN.
 1190  *    (C) IF CHRGET/CHRGOT VECTOR DESTROYED BY APSOFT
 1200  *        COLDSTART (]FP, *E000G, *CTL-B), RESET LINE.EDIT
 1210  *        VECTOR BY CALL 768.
 1220  *--------------------------------
 1230          .OR $300
 1240  *--------------------------------
 1250  *          APPLESOFT POINTERS
 1260  *--------------------------------
 1270  AS.FORPNT  .EQ  $85     ;HOLD Y-REGISTER
 1280  AS.LOWTR   .EQ  $9B,$9C ;LOCATION OF CHARACTER OR TOKEN IN PGM
 1290  AS.DSCTMP  .EQ  $9D,$9E ;LOCATION IN KEYWORD TABLE
 1300  *--------------------------------
 1310  *          APPLESOFT CHRGET/CHRGOT
 1320  *--------------------------------
 1330  AS.CHRGET  .EQ  $B1     ;GETS CHARACTER AT TEXT POINTER
 1340  AS.TXTPTR  .EQ  $B8,$B9 ;TEXT POINTER
 1350  AS.CHREXT  .EQ  $BA     ;CHRGET/CHRGOT VECTOR TO LINE.EDIT
 1360  AS.CHRENT  .EQ  $BE     ;RE-ENTRY TO CHRGET/CHRGOT
 1370  *--------------------------------
 1380  *          APPLESOFT ROM
 1390  *--------------------------------
 1400  AS.FNDLIN  .EQ  $D61A   ;ADDR NMBR IN LINNUM ($50,$51) TO LOWTR
 1410  AS.CRDO    .EQ  $DAFB   ;LINEFEED
 1420  AS.OUTSP   .EQ  $DB57   ;OUTPUT SPACE
 1430  AS.OUTDO   .EQ  $DB5C   ;OUTPUT CHARACTER
 1440  AS.FRMEVL  .EQ  $DD7B   ;FORMULA AT TEXT POINTER TO FAC ($9D-
$A2)
 1450  AS.GETADR  .EQ  $E752   ;FAC TO INTEGER IN LINNUM ($50,$51)
 1460  AS.LINPRT  .EQ  $ED24   ;PRINT DECIMAL OF (A,X)
 1470  *--------------------------------
```

```
1480  *          MONITOR ROM
1490  *--------------------------------
1500  MON.TABV   .EQ  $FB5B   ;VTAB TO VALUE IN (A)
1510  MON.HOME   .EQ  $FC58   ;HOME CURSOR, CLEAR SCREEN
1520  MON.BELL   .EQ  $FF3A   ;BEEP!
1530         .PG
1540  *--------------------------------
1550  *  PUT LINE.EDIT VECTOR INTO CHRGET/CHRGOT
1560  *--------------------------------
1570  START  LDA #$4C         ;JMP 'LINE.EDIT'
1580         STA AS.CHREXT
1590         LDA #EDIT
1600         STA AS.CHREXT+1
1610         LDA /EDIT
1620         STA AS.CHREXT+2
1630  RTS1   RTS
1640  *--------------------------------
1650  *  CHECK FOR VALID COMMAND
1660  *--------------------------------
1670  EDIT   CMP #$2F         ;IS IT A SLASH (/)?
1680         BNE .1           ;NO. RETURN
1690         INC AS.TXTPTR    ;YES. BUMP TEXT POINTER
1700         BNE .2           ;BRANCH ALWAYS
1710  *--------------------------------
1720  *  RETURN TO CHRGET/CHRGOT OR CALLER
1730  *--------------------------------
1740  .1     CMP #$3A         ;IF COLON (EOS), SET Z AND C
1750         BCS RTS1         ; FLAGS AND RETURN TO CALLER
1760         JMP AS.CHRENT    ;IF NOT EOS, RE-ENTER CHRGET/CHRGOT
1770  *--------------------------------
1780  *  FIND LOCATION OF LINE NUMBER
1790  *--------------------------------
1800  .2     JSR AS.FRMEVL    ;PUT LINE NUMBER INTO FAC ($9D-$A2)
1810         JSR AS.GETADR    ;PUT FAC INTO LINNUM ($50,$51)
1820         JSR AS.FNDLIN    ;PUT ADDR OF LINE INTO LOWTR
1830         BCC .5           ;CARRY CLEAR IF LINE NMBR NOT FOUND
1840  *--------------------------------
1850  *  CLEAR SCREEN AND SET TO ROW 2, COLUMN 2
1860  *--------------------------------
1870         JSR MON.HOME
1880         JSR AS.CRDO
1890         JSR AS.OUTSP
1900  *--------------------------------
1910  *  PRINT LINE NUMBER
1920  *--------------------------------
1930         LDY #02          ;SET INDEX TO LINE NUMBER BYTES
1940         LDA (AS.LOWTR),Y ;PUT LINE NUMBER LO
1950         TAX              ; INTO (X)
1960         INY
1970         LDA (AS.LOWTR),Y ;PUT LINE NUMBER HI INTO (A)
1980         STY AS.FORPNT    ;HOLD (Y)
1990         JSR AS.LINPRT    ;PRINT DECIMAL OF (A,X)
2000  *--------------------------------
2010  *  GET CHARACTER OR TOKEN
```

```
2020  *---------------------------------
2030         LDA #$20         ;SPACE
2040  .3     LDY AS.FORPNT    ;RESTORE (Y)
2050  .4     JSR AS.OUTDO     ;PRINT CHARACTER IN (A)
2060         INY
2070         LDA (AS.LOWTR),Y ;GET CHARACTER OR TOKEN
2080         BNE .8           ;IF NOT EOS (0), GET MORE
2090         .PG
2100  *---------------------------------
2110  *  TWO ENDINGS -- ONE HAPPY, ONE SAD
2120  *---------------------------------
2130         LDA #00          ;LINE WAS FOUND. END WITH
2140         JMP MON.TABV     ; CURSOR AT ROW 2, COLUMN 2
2150  .5     JSR MON.BELL     ;LINE WAS NOT FOUND. END WITH
2160         JMP AS.CRDO      ; CURSOR BELOW COMMAND INPUT
2170  *---------------------------------
2180  *  GET CHARACTER IN KEYWORD TABLE
2190  *---------------------------------
2200  .6     INY
2210         BNE .7
2220         INC AS.DSCTMP+1
2230  .7     LDA (AS.DSCTMP),Y
2240         RTS
2250  *---------------------------------
2260  *  PRINT CHARACTER OR KEYWORD
2270  *---------------------------------
2280  .8     BPL .4           ;NON-TOKEN IS POS ASCII
2290         SEC              ;TOKEN MINUS $7F EQUALS INDEX TO
2300         SBC #$7F         ; LOCATION OF KEYWORD IN TABLE
2310         TAX              ;PUT INDEX IN (X)
2320         STY AS.FORPNT    ;HOLD (Y)
2330         LDY #$D0         ;KEYWORD TABLE STARTS AT $D0D0
2340         STY AS.DSCTMP
2350         LDY #$CF
2360         STY AS.DSCTMP+1
2370         LDY #$FF         ;WHEN BUMPED, (Y) WILL BE ZERO
2380  .9     DEX              ;DEC INDEX TO KEYWORD LOCATION
2390         BEQ .11          ;WHEN (X) IS ZERO, KEYWORD LOCATED
2400  .10    JSR .6           ;GET CHARACTER IN KEYWORD TABLE
2410         BPL .10          ;IF POS ASCII, GET ANOTHER
2420         BMI .9           ;IF NEG ASCII, DEC LOCATION INDEX
2430  .11    JSR AS.OUTSP     ;PRINT SPACE
2440  .12    JSR .6           ;GET CHARACTER IN KEYWORD TABLE
2450         BMI .13          ;IT'S THE FINAL CHAR IN KEYWORD
2460         JSR AS.OUTDO     ;PRINT NON-FINAL CHAR (POS ASCII)
2470         BNE .12          ;BRANCH ALWAYS
2480  .13    JSR AS.OUTDO     ;PRINT FINAL CHAR (NEG ASCII)
2490         LDA #$20         ;SPACE
2500         BNE .3           ;BRANCH ALWAYS
2510  *---------------------------------
2520  SIZE   .EQ *-START
2530         .PG
```

```
=============================================================================
DOCUMENT :AAL-8112:DOS3.3:S.PMD.Subr.txt
=============================================================================

 1000  *---------------------------------
 1010  *      BUILD STRING FROM SECOND LINE ON SCREEN
 1020  *---------------------------------
 1030         .OR $300
 1040  *---------------------------------
 1050  PTRGET .EQ $DFE3  PUTS STRING POINTER ADDRESS IN $83,84
 1060  GETSPA .EQ $E452  PUTS ADDRESS OF STRING SPACE IN $71,72
 1070  MOVSTR .EQ $E5E2  MOVES DATA FROM (Y,X) TO STRING SPACE
 1080  *---------------------------------
 1090  SPCPTR .EQ $71,72 PNTR TO STRING SPACE RESERVED BY GETSPA
 1100  STRPTR .EQ $83,84 PNTR TO STRING VARIABLE PTRGET GOT
 1110  *---------------------------------
 1120  *      TO USE:
 1130  *          CALL 768A$(X)
 1140  *---------------------------------
 1150  GO     JSR PTRGET   GET ADDRESS OF STRING INTO $83,84
 1160         LDA #11      MOVE 11 BYTES
 1170         JSR GETSPA   GET SPACE FOR 11-BYTE STRING
 1180         LDY #0
 1190         STA (STRPTR),Y  PUT LENGTH IN STRING DESCRIPTOR
 1200         LDA SPCPTR   LOW BYTE OF STRING ADDRESS
 1210         INY
 1220         STA (STRPTR),Y
 1230         LDA SPCPTR+1 HIGH BYTE OF STRING ADDRESS
 1240         INY
 1250         STA (STRPTR),Y
 1260         LDX #$0494   START OF OPCODE ON SECOND SCREEN LINE
 1270         LDY /$0494       ADDRESS IN (Y,X)
 1280         LDA #11      11 BYTES LONG
 1290         JSR MOVSTR   MOVE IT IN
 1300         RTS
```

=============================================================================
DOCUMENT :AAL-8112:DOS3.3:TEST.FAST.READ.txt
=============================================================================

```
dD$–Á(4)(n∫D$"BLOAD B.FAST READ"Exπ1013,76:π1014,0:π1015,3[Ç∫D$"OPEN
MY.FILE"kåÅI–1¡10:ÑA$àñ∫D$"WRITE MY.FILE":∫A$:Çó†∫D$"CLOSE"≠“∫D$"OPEN
MY.FILE"√‹∫D$"READ MY.FILE"œÊÅI–1¡10¤ØæA$(I),˙ÇIÒ∫D$"CLOSE"
     ,ÅI–1¡10:∫I" "A$(I):Ç
```

==============================================================================
DOCUMENT :AAL-8112:DOS3.3:Test.GotoFromML.txt
==============================================================================

( DTC removed -- lots of garbage characters )

==========================================================================
DOCUMENT :AAL-8201:Articles:Front.Page.txt
==========================================================================

$1.20

Volume 2 -- Issue 4                                   January, 1982

In This Issue...



Renew Now, the Price is Going Up

If you renew your subscription before March 1, 1982, you can renew at the current rate of $12/year.  Starting March 1st, the price will go up to $15/year (2nd class mail in the USA).  Subscriptions sent First Class Mail to USA, Canada, and Mexico will be $18/year.  Air Mail subscriptions to all other countries will be $28/year.  The price for back issues will be $1.50 each (plus $1.00 postage outside of USA, Canada, and Mexico).

S-C MACRO Assembler II is almost here!

I am committed to having a finished product by February 15th.  This is what I have been calling Version 5.0, but I have decided to call it S-C MACRO Assembler II instead.  Version 4.0 will still be sold at $55. The MACRO version will be $80.  Owners of Version 4.0 can upgrade for only $27.50.  There will be an all new manual, rather than the current 2-part manual.

The MACRO Assembler includes macros (of course!), conditional assembly, EDIT, COPY, global string replacement, and many more new features.  And it assembles even faster than version 4.0!

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050.  Subscription rate is $12 per year in the U.S.A., Canada, and Mexico.  Other countries add $12/year for extra postage.  Back issues are available for $1.20 each (other countries add $1 per back issue for postage).   Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof.  (Apple is a registered trademark of Apple Computer, Inc.)

```
==============================================================================
DOCUMENT :AAL-8201:Articles:HandyExecFiles.txt
==============================================================================
```

Handy EXEC Files........................Bob Sander-Cederlof

Now that I have my Firmware card with Integer BASIC on it plugged into slot 4, I am all too frequently needing to fix those two bytes in DOS. For some reason I don't get around to putting the patched DOS onto every disk.  But with a few EXEC files I can make the patches very easily now.

The first EXEC file, which I call INT, is like this:

```
CALL -151          (get into the monitor)
C081               (turn off the language card, if on)
C0C1               (turn off the firmware card, if on)
A5B8:C0            (patch DOS to use firmware card)
A5C0:C1
3D3G               (return to DOS and Applesoft)
INT                (enter Integer BASIC)
```

The second file I use to load LANGASM into the Language Card (see Paul Schlyter's article elsewhere in this issue of AAL).  Here is what it looks like:

```
CALL-151           (get into the monitor)
C0C1               (turn off the firmware card, if on)
C081 C081          (write enable the language card)
BLOAD LANGASM      (load LANGASM into the language card)
A5B8:80            (patch DOS to use the language card)
A5C0:81
3D3G               (return to DOS and Applesoft)
INT                (enter the assembler)
```

The third EXEC file I use to patch DOS back to its normal mode of using the language card in slot 0.  If I have already loaded the S-C Assembler II (LANGASM) into that card, but was using Integer BASIC, EXEC ASM will get me back to the assembler.

```
CALL-151           (get into the monitor)
C081               (turn off the language card, if on)
C0C1               (turn off the firmware card, if on)
A5B8:80            (patch DOS to use the language card)
A5C0:81
3D3G               (return to DOS and Applesoft)
INT                (enter the assembler)
```

Just for fun, here is one more EXEC file.  This one copies the contents of the firmware card in slot 4 into the language card in slot 0.  A much faster way of loading it with Integer BASIC than running HELLO on the DOS 3.3 System Master!

```
CALL-151          (get into the monitor)
C0C0              (turn on the firmware card)
1000<D000.FFFFM   (copy firmware card into mother RAM)
C0C1              (turn off the firmware card)
C081 C081         (write enable language card)
D000<1000.3FFFM   (copy stuff into the language card)
3D0G              (return to DOS)
```

If you don't have an editor that will help you build EXEC files like these, here is a short Applesoft program which will do it. I have also included a short program to display the file, in case you need to do that.

Both of these programs CALL 64874, which is the Apple Monitor subroutine to read a line into the system buffer starting at $200. The CALL -3288 in READ EXEC FILE is to fix the ONERR stack pointer.

==============================================================================
DOCUMENT :AAL-8201:Articles:HiresScrnColor.txt
==============================================================================

Hi-Res SCRN Function with Color...............David Doudna

I am a 15-year-old living in St. Louis, Missouri. While looking through the back issues of Apple Assembly Line, I found "Hi-Res SCRN Function for Applesoft" (May, 1981 issue). I noticed the routine only returned a 0 or 1, and you challenged readers to write one to return a color value 0-7. Well, I did it. My version is not interfaced to Applesoft; that is an exercise for the reader! (I use the Programmer's Aid ROM with FORTH.)

I am not going to explain how hi-res colors work, beyond the facts that two adjacent dots are white; the upper bit in each byte of the hi-res screen adds 4 to the color value; an isolated bit is color 1 or 2 (or 5 or 6) depending on the X-position. If you want to understand my program, you should study more about hi-res plotting first.

A word about the color value.... In Applesoft you specify color value with a number from 0 to 7. The Programmer's Aid ROM uses color values of 0, 42, 85, 127, 128, 170, 213, and 255. My program returns both numbers for the color: the 0-7 index in HCOLOR, and the P.A.ROM color value in COLOR.BYTE.

Lines 1060-1140 define the variables used; these are in the same locations as those used by the Programmer's Aid ROM. If you want to modify the program to work with Applesoft, be sure to put these variables in the correct locations. Two more variables are defined at lines 2120,2130.

Lines 1160-1180 pick up the X- and Y-coordinates. I assume you have stored the coordinates here before calling HSCRN. Lines 1190-1390 calculate the base address for the particular horizontal line your point is on. This code is just copied out of P.A.ROM. Lines 1410-1530 divide the X-coordinate by 7 to get the byte offset on the line. The quotient is left in the Y-register. The remainder is used to pick up a bit mask to select the particular bit within the byte.

Lines 1540-1650 make the first color check. The high-order bit of the byte (half-dot shift control). If the bit specified = zero, the color is black. If it = one, the color depends on whether either neighbor of this dot = one. If neither neighbor = one, the color depends on whether this dot is in an even or odd column. If the color is not black, I put 1 or 2 in the X-register to indicate the color it will be if it is not white.

Lines 1660-1790 check the neighbor bit on the left to see if it = one. Notice that there are several special cases. First, the left-neighbor might be in the same byte. Second, it might be in the byte to the left of this one. Third, there might not be a byte to the left of this one.

Lines 1800-1920 check the neighbor bit on the right. The same kind of special cases exist here, and they are handled the same way.

Line 1940 sets X = 3 for white color. Line 1960 gets the color value in the A-register. All paths merge at line 1980, with the color index 0-3 in the A-register. All that remains is to add 4 if the half-dot shift control = 1 (Lines 1980-2010.

Lines 2020-2060 convert the color index to a color byte (by simple table-lookup), and return. Line 2100 is the table of color values.

Here is a table of colors (their names, index numbers, and P.A.ROM numbers):

| | | Color Byte Value | | |
|---|---|---|---|---|
| Color | Index | Hex | Dec | Binary |
| BLACK | 0 | 00 | 0 | 00000000 |
| GREEN | 1 | 2A | 42 | 00101010 |
| VIOLET | 2 | 55 | 85 | 01010101 |
| WHITE | 3 | 7F | 127 | 01111111 |
| BLACK2 | 4 | 80 | 128 | 10000000 |
| ORANGE | 5 | AA | 170 | 10101010 |
| BLUE | 6 | D5 | 213 | 11010101 |
| WHITE2 | 7 | FF | 255 | 11111111 |

The program works with either page 1 or page 2 of Hi-Res. Set HPAGE to $20 for page 1, or $40 for page 2.

To call this program from Integer BASIC, you would first POKE the X- and Y-coordinates, then CALL the program, and then PEEK the color value. From assembly language, set up the coordinates and JSR HSCRN. The color index is returned in the X-register, and the color byte value in the A-register.

[ Program and article modified somewhat by the editor ]

=============================================================================
DOCUMENT :AAL-8201:Articles:OneChip6500.1.txt
=============================================================================

6500/1 One-Chip Computer...........................Dan Pote

Commodore Semiconductor Group has announced a new one-chip microcomputer, called the 6500/1. (I believe the same chip is available from Synertek and Rockwell.) The 6500/1 has a 6502 CPU and is compatible with existing 6502 programs. There are also four I/O ports (32 bi-directional lines, the equivalent of two 6522 devices), a counter, 2048 bytes of ROM, and 64 bytes of static RAM. Your choice of 1- or 2-MHz internal clock. It can be ordered as masked-ROM, PROM, or piggy-back EPROM. For more information call Commodore at (214) 387-0006.

==============================================================================
DOCUMENT :AAL-8201:Articles:Relocator.6502.txt
==============================================================================

6502 Relocator..........................Bob Sander-Cederlof

Programs that are already assembled usually must be loaded at a specific memory address to execute properly. If you want to run it somewhere else, you have a problem. All the data references, JMP's, and JSR's will have to be examined to see if they need to be modified for the new location. If you don't have the source code, you can't re-assemble it. The other way, patching, can be quite a tedious operation!

Fortunately, way back in 1977, the WOZ (Steve Wozniak to you newcomers) wrote a program to do the work automatically. If you have the Programmer's Aid ROM then you have his RELOCATE program. You who have Apple II Plusses with the Language Card (also called 16K RAM card) can also use his program, because it is in the INTBASIC file along with Integer BASIC. (The latter group of people probably don't have the manual, though, because they didn't buy the ROM.)

I would like to see the RELOCATE program made more widely available, but it cannot be used as is unless you have Integer BASIC. Why? Because it uses SWEET-16 opcodes. RELOCATE also is itself tied to running at whatever location it is assembled for, so it can be a little trouble to find a place for it sometimes. By now you have probably guessed that I have recoded RELOCATE to solve both of these problems!

Paul Schlyter's article elsewhere in this issue of AAL shows RELOCATE put to good use. You can examine his instructions and learn most of what you need to know to use RELOCATE on your own programs. Basically, there are four steps:

1. Initialize. This sets up the control-Y monitor command. If RELOCATE is on a file, you do this with "BRUN RELOCATE".

2. Specify the program start and end addresses (where it now is in memory), and the new starting address (where you want it to be relocated to). This is done with the monitor command:

```
    target<start.end^Y*
```

where "target" is the new starting address, and "start" and "end" are the addresses of the program where it is now. "^Y" means "control-Y". The "*" after the control-Y signals RELOCATE that you are in step 2 rather than step 3 or 4.

3. Specify the FIRST block to be copied "as-is" or to be "relocated" to the destination area. This is done with the monitor command:

```
    target<start.end^Y
```

```
or  target<start.endM
```

where "target" is the starting address in the new area for this block, and "start" and "end" define the block itself. Note that there is no trailing asterisk this time. Use control-Y if you want this block relocated, or M if you want it copied as-is.

4. Specify the NEXT block to be copied as-is or relocated. You do this with the monitor command:

```
       .end^Y
or     .endM
```

where the target and start addresses are assumed to immediately follow the previously handled block, and "end" specifies the end of this new block. Use control-Y to relocate the block, or M to copy it as-is.

Obviously, step 4 above is repeated until the whole program has been copied/relocated. For each block of your program that is to be copied as-is, with no modification at all, you use the "M" command; for each block to be relocated you use the "control-Y" command.

If you need more detailed instructions and explanation, I must refer you to the manual. The Programmer's Aid #1 Manual is sold at most computer stores separately from the ROM package. Pages 11-28 explain why and how to use RELOCATE, and pages 80 and 81 contain the assembly listing.

Now here is my new version, which can be BRUN anywhere you have 134 ($86) bytes available. I have eliminated the SWEET-16 usage; this made the program slightly bigger, and a lot faster.

Lines 1260-1380 are the initialization code. They build the control-Y vector at $3F8-3FA. A JMP opcode is stored at $3F8; if you have DOS up this is redundant, but it won't hurt. Next I have to try to find myself. That is, where in memory am I (the program RELOCATE) located? JSR MON.RETURN (which is only an RTS instruction, so it comes right back without doing anything) puts the address of the third byte of the JSR instruction on the stack. Lines 1290-1370 use that address to compute the address of RELOC, and store it in $3F9 and $3FA.

When you type in a control-Y command, the monitor will now branch to RELOC at line 1400. Lines 1400-1430 look at the character after the control-Y in the command input buffer; if it is an asterisk, then you are trying to do step 2 above. If not, then you are on step 3 or 4. Lines 1440-1500 handle step 2, and lines 1510-1990 handle steps 3 and 4.

The part which used to be coded in SWEET-16 was lines 1690-1880. The SWEET-16 version took only 14 bytes, while the 6502 code takes 34 bytes. The 6502 version may take about 100 microseconds to execute, and the SWEET-16 version on the order of 1000 microseconds (for each instruction relocated).

================================================================================
DOCUMENT :AAL-8201:Articles:Review.Index.txt
================================================================================

A Review of THE INDEX...................Bob Sander-Cederlof

THE INDEX is a new book that you can use. No doubt you subscribe to three or more magazines and newsletters, out of the 100 or so that are being published with information Apple owners want and need. Wouldn't you like a composite index that covered the best ones?

Bill Wallace an attorney in St. Louis, Missouri, has put together just such an index. His book compiles over 12000 articles, editorials, and columns from over 900 issues of personal computer magazines published during the last six years. Over 40 different magazines and newsletters are covered. I am honored that Bill has chosen to include both of my newsletters: Apple Assembly Line, and AppleGram.

Organized as a Key-Word in Context (KWIC) index, there are over 30000 entries. There are 92 pages of Apple-related articles, 160 pages covering other computers (Apple owners will be interested in the CP/M and 6502 sections), and over 200 pages of general articles. All the information necessary for obtaining copies and/or subscriptions of the various magazines and newsletters is also included.

Bill plans to publish a second edition later this year to include the issues published since the cutoff date of the first edition, as well as lots of additional publications that were not previously covered.

THE INDEX costs $14.95, and is available from Missouri Indexing, Inc., P. O. Box 301, St. Ann, MO 63074. Bill is responsive to requests for group rates, if you can interest your local Apple club; call him at (314) 997-6470.

================================================================================
DOCUMENT :AAL-8201:Articles:SCAsm.2.LC.txt
================================================================================

Putting S-C Assembler II on the Language Card . . Paul Schlyter

[ Paul is a subscriber in Stockholm, Sweden. ]

Introduction

I have owned the S-C Assembler II for only a little more than three weeks, and already I have stopped using the two other assemblers I used to use before ("Apple Text Processing System" and "DOS Tool Kit Assembler"). Although the others have some powerful features, the S-C Assembler is so much easier to use it now takes me only about half the time to finish an assembly language program as it did before.

The many similarities between the S-C Assembler and Integer BASIC made me curious, so I disassembled the Assembler. Earlier I have done the same thing with Integer BASIC, Applesoft, and DOS. It wasn't too long (about a week) that I had a fair understanding of the first third of the assembler. Then the idea turned up in my head: "Why not try to relocate it into the language card?" Another week of sleepless nights and it was up and running!

There were several traps on the way. It took a long time for me to discover the address stack put into DOS at $1333-133C. Sometimes I just entered the regular assembler at $1000-24FF and didn't notice anything, sometimes the machine crashed when a DOS error occurred. But that was a week ago, and the last week nothing like that has happened...now I feel fairly confident that I have found all bytes that need to be relocated. If anything does turn up, I will let you know.

Why and How

Have you ever thought about how very similar to Integer BASIC the S-C Assembler II is? It stores its source files as DOS type-I files, and numbers the lines the same way as Integer BASIC. Just like in Integer BASIC, you have access to all DOS commands. Well, the similarities don't stop there. Integer BASIC starts at address $E000 and ends a little bit above $F400; the S-C Assembler starts at $1000 and ends a little bit above $2400. The byte at $E000 is $20 in Integer BASIC (JSR opcode) while it is $4C in Applesoft (JMP opcode); it is by looking at this byte that DOS decides whether Integer BASIC or Applesoft is the current language. Well, guess what the byte at $1000 in the S-C Assembler is: it's $20!

When putting all these facts together, I started to wonder if it wasn't possible to relocate the S-C Assembler up into the Language card, making DOS believe it is in Integer BASIC. This of course requires that you have Applesoft on the motherboard ROMs, so that DOS will be able to distinguish between the ROM and Language card languages.

Sure enough, it is possible. I did move it up there, and it works, and it turned out to be really convenient. The DOS command FP puts me in Applesoft, while INT puts me into the S-C Assembler! Also, if I am currently in Applesoft and LOAD an S-C Assembler source file (type-I, of course), DOS will automatically start up the assembler! Can you really ask for more?

To relocate the S-C Assembler into the language card, you need of course a language card (any of the several RAM cards no available will do). You also need to have Applesoft in ROM on the motherboard (not Integer BASIC). You also need a relocation program; I used the one in the Programmer's Aid #1 software, which is in the INTBASIC file on the DOS 3.3 System Master. You could use the one in Bob Sander-Cederlof's article elsewhere in this AAL just as well.

Step-by-step Procedure

1. Boot the DOS 3.3 System Master. This will load Integer BASIC into the Language Card.

2. Type INT to enter Integer BASIC.

3. Put in the S-C Assembler II disk, and BLOAD ASMDISK 4.0 (do not BRUN it).

4. Enter the Apple monitor by typing CALL -151. (Throughout the following steps, be sure you do NOT hit RESET!)

5. Now that you are in the monitor, type the following commands:

```
C083 C083          (write-enable the language card)

D4D5G              (initialize the relocation program)

E000<1000.24FF^Y*  (specify source and destination
                    blocks for the relocation program.
                    Note that "^Y" means "control-Y".
                    The asterisk at the end IS necessary.)

E000<1000.100E^Y   (relocate the first segment)

.100FM .121E^Y .1282M .1330^Y .133CM .1436^Y

.1438M .147C^Y .14A9M .14DB^Y .141EM .14F3^Y

.14F5M .15D0^Y .15D6M .17A6^Y .17AEM .1A8C^Y

.1A91M .1BB7^Y .1CAEM .2149^Y .2150M .221C^Y .24FFM
```

(The monitor commands on the above four lines relocate the program segments and move the data segments. They can be typed as shown, or one per line, or even all on one line.  Just be sure to type them correctly -- check and double check -- before pressing RETURN.)

6.  The machine code relocator automatically updates any direct address references in the program being relocated.  This saves us a lot of work, but it does not finish the work.  We also have to fix all the address tables and all immediate address references.  Enter the following monitor commands to fix all of these (only one command per line):

```
E042:E2     E254:E2     E334:E0     F234:F1
E227:E7     E259:EB     E336:E0     F239:F0
E22C:E5     E25E:E3     E338:E0     F23E:F0
E231:E0     E263:E0     E33A:E3     F243:F0
E236:E5     E268:E5     E33C:E0     F248:F2
E23B:E1     E26D:F1     E4A3:E4     F24D:F0
E240:E4     E272:E6     E83E:F2     F252:F1
E245:E1     E277:E1     F225:F0     F257:F0
E24A:E6     E27C:E0     F22A:F1     F25C:EE
E24F:E3     E281:EB     F22F:F0     F261:E0
```

7.  The cold start routine in the Assembler must be patched:

```
    E030:ED E2

    E2E0:AD 83 C0 AD 83 C0 A9 00 85 D9 4C 08 E3 AD 83 C0

    E2F0:AD 83 C0 4C 75 E3
```

8.  If you wish, you may change the starting address of the Assembler Symbol Table to make more space:

```
    E011:10

    E2D6:10
```

9.  If you enter Applesoft from the S-C Assembler, the output hook from DOS will still be connected to the S-C Assembler output routine. But the assembler will be banked away since now the motherboard ROMs are enabled!  The result is that the Apple will hang.  To cure this problem, you will have to sacrifice the SLOW and FAST commands, and the ability to suspend/abort listings using the space bar and RETURN keys.  This is not such a big sacrifice anyway, since all language card owners have the Autostart Monitor:  you can use control-S to suspend a listing.  You can also use RESET to abort one (provided your language card has a switch and it is in the UP position).  [If you can't bear to part with SLOW and FAST, you can type FP and then hit RESET to get out of the Assembler.]

Here are the patches to eliminate SLOW and FAST:

```
    E1E9:EA EA EA EA

    E22D:4D 4E 54 68 FF
```

```
    E273:FF FF FF
```

These patches also change FAST to MNT, a command that gracefully enters the monitor. From the monitor, a control-C will re-enter the S-C Assembler with the current source program intact; a control-B will cold start the S-C Assembler.

10. Save the completed package on disk with:

```
     BSAVE LANGASM,A$E000,L$2000.
```

11. Modify a copy of the HELLO program from the DOS 3.3 System Master Disk to BLOAD LANGASM instead of INTBASIC, and use this as your HELLO program. When you boot it will automatically load the S-C Assembler II into your language card.

Parting Shots

Maybe you think that I must have a thorough knowledge of how the S-C Assembler II works internally to be able to do this relocation, but this is not actually the case. I made a disassembly and also hex and ASCII dumps of the whole assembler, and I also started to untangle the code, but I only really know about a third of the code fairly well. I still have not the faintest idea of how the actual assembly is performed, although looking at the ASCII dump immediately revealed where the opcode and command tables were located, and the error messages. I also did find out the places where the error messages are produced...this helps a lot in figuring out what is happening in the code. And with this not-too-well understanding of the inner workings, and with a lot of trial-and-error, I was able to find all the places where changes needed to be made.

My S-C Assembler has been running from my language card for over a week, and I have used it a lot during this time; all has gone very well. And believe me, it is SO CONVENIENT to have it there! I really benefit from the language card, not only when using Pascal or CP/M, but also when I am running DOS. And I use the S-C Assembler II much more than Integer BASIC, so having the assembler in the language card is really the right thing for me. Maybe it is for you too!

So, Bob, although you have made an excellent and very easy to use assembler, it is not quite true anymore that the S-C Assembler II is the easiest assembler to use...LANGASM is. And as you have guessed, LANGASM is nothing but the S-C Assembler II relocated into the language card!

[ If the instructions for making LANGASM leave you breathless, you can order Quarterly Disk #6 ($15.00). It will include all the source code from this issue and the next two of AAL, and also an EXEC file which will create LANGASM from ASMDISK 4.0. It will be ready in early March, 1982. Another shortcut is to order the source code of the S-C Assembler II ($95.00). Then simply change the origin, modify the SLOW and FAST commands, and re-assemble it. Voila! LANGASM! ]

===========================================================================
DOCUMENT :AAL-8201:Articles:SeriousDOSPro.txt
===========================================================================

Serious Problem in Apple DOS..............Bob Sander-Cederlof

If you are trying to use the IRQ interrupt line for any purpose, and also DOS, you may have run across this problem before. Apparently at random, for no good reason, you may get the NO BUFFERS AVAILABLE message.

The reason is that both DOS and the IRQ interrupt code are trying to use the same page zero location: $0045. DOS uses this location as part of a pointer address when looking for the next available buffer. (See the code at $A2CB-A2CF and $A764-A780.) DOS also uses $0045 when printing the catalog (see $ADB9, $AE09, and $AE53).

The IRQ interrupt code in the Apple Monitor ROM (at $FA86) uses $0045 to save the contents of the A-register. If an interrupt occurs while DOS is in the process of looking for a buffer, POW!

One solution is to turn off interrupts whenever DOS may be active, using the SEI opcode. A better solution would be quite difficult: look through all of DOS and modify every reference to $0045 (or to $0044 and $0045 as a pair) to use some other location in page zero. A third possible solution for those who can do it is to modify the Apple Monitor ROM to use some other location to save the A-register.

In case you ARE using interrupts and DOS together, you should also know that RWTS does inhibit interrupts while it is active. After a call to RWTS is complete, the interrupt-inhibit status is restored to whatever it was before the call. Interrupts cannot be allowed during RWTS, because of the critical software timing code involved in reading and writing the disk.

```
=============================================================================
DOCUMENT :AAL-8201:Articles:StepTraceCorrex.txt
=============================================================================
```

A Correction to "Step-Trace Utility"....Bob Sander-Cederlof

"Step-Trace Utility", published in the July 1981 issue of AAL (pages 17-20), has a bug. Three or four of you ran into the problem and called me about it, but I was never able to duplicate the problem. Finally Bob Leedom managed to pinpoint the bug, and I found out how to fix it.

If you have used Step-Trace, you might have noticed that it sometimes will hang-up or go crazy after a relative branch instruction. The problem is that if the 6502 was in decimal mode, the calculations are all incorrect. This affects the branch target, and also messes up screen output. To fix it, insert the following line:

```
     2095        CLD          SELECT bINARY MODE
```

But how did the 6502 get into decimal mode, when I wasn't ever setting it? The contents of SAVE.P were random on initial start-up. Sometimes the contents managed to switch on decimal mode! Perhaps you should also insert the following two lines, to be certain of the initial status of the program you are tracing:

```
     1455        LDA #0       CLEAR INITIAL STATUS
     1456        STA SAVE.P
```

Future copies of Quarterly Disk #4 already have these two patches installed.

```
==============================================================================
DOCUMENT :AAL-8201:DOS3.3:AS.Copy.FW.txt
==============================================================================

d∑WRITE EXEC FILE8n∑TO COPY SLOT 4 FIRMWARE CARD[x∑CONTENTS INTO SLOT
0 RAM CARDz∑D$»‰(4):F$»"COPY
FIRMWARE"©†∑D$"OPEN"F$:∑D$"DELETE"F$∑®∑D$"OPEN"F$:∑D$"WRITE"F$∫∞∑"CALL
-151"Ï∑∑"C0C0":∑TURN ON SLOT 4 FIRMWARE CARD
     ∑∑"1000<D000.FFFFM":∑COPY CONTENTS TO RAM@    ∑∑"C0C1":∑TURN OFF
FIRMWARE CARDh ∑∑"C081 C081":∑WRITE ENABLE RAM CARD¢
     ¿∑"D000<1000.3FFFM":∑LOAD RAM CARDµ ∑"3D0G":∑BACK TO DOS∑
     ’∑D$"CLOSE"d
```

```
=====================================================================
DOCUMENT :AAL-8201:DOS3.3:AS.MAKE.LANGASM.txt
=====================================================================

´100)
≤ PRINT BY AS 2 HEX DIGITSSD1–”(BYÀ16):D2–BY…D1 16:≠D1œ9ƒD1–
D1»7e-≠D2œ9ƒD2–D2»7}(∫Á(D1»48);Á(D2»48);É2±ùd≤MAKE EXEC FILE
WHICH√n≤CREATES LANGASM FROM ASMDISK 4.0·xD$–Á(4):F$–"MAKE LANGASM")
     }∫D$"MONCIO":∫D$"BLOAD ASMDISK 4.0,A$4000":∫D$"BLOAD
LANGASM,A$6000"`
     Ç∫D$"OPEN"F$:∫D$"DELETE"F$:∫D$"OPEN"F$:∫D$"WRITE"F$Å
     å∫"INT":∫"MONCIO":∫"CALL-151"í   ñ∫"C083 C083"≤  †∫"BLOAD ASMDISK
4.0,A$E000"‘   ™A–8192:B–40960:ÅI–16384¡21755Ë     ¥C–,(I):D–,(I»A)ı
     æ≠C–Dƒ300
»E–I»B:≤PRINT EEEE:DD2
“BY–”(EÀ256):∞10:BY–E…BY 256:∞10<
‹∫":";K
ÊBY–D:∞10:∫Q
,Çt
6∫"BSAVE LANGASM,A$E000,L$2000"Å
;∫"E000G"ê
@∫D$"CLOSE"ñ
JÄ
```

================================================================================
DOCUMENT :AAL-8201:DOS3.3:ASM.txt
================================================================================

```
CALL-151
C081
C0C1
A5B8:80
A5C0:81
3D3G
```

```
==============================================================================
DOCUMENT :AAL-8201:DOS3.3:COPY.FIRMWARE.txt
==============================================================================

CALL-151
C0C0
1000<D000.FFFFM
C0C1
C081 C081
D000<1000.3FFFM
3D0G
```

==============================================================================
DOCUMENT :AAL-8201:DOS3.3:INT.txt
==============================================================================

```
CALL-151
C081
C0C1
A5B8:C0
A5C0:C1
3D3G
```

================================================================================
DOCUMENT :AAL-8201:DOS3.3:LOAD.ASM.txt
================================================================================

```
CALL-151
C081 C081
BLOAD LANGASM
A5B8:80
A5C0:81
3D0G
```

```
=============================================================================
DOCUMENT :AAL-8201:DOS3.3:MAKE.LANGASM.txt
=============================================================================

INT
MONCIO
CALL-151
C083 C083
BLOAD ASMDISK 4.0,A$E000
E002:E2
E005:E0
E008:E0
E00B:E0
E00E:E8
E011:10
E01E:EF
E025:E1
E030:ED
E031:E2
E036:E2
E042:E2
E04F:E6
E054:E1
E057:E1
E05A:E8
E05D:E0
E062:E7
E065:E1
E07A:E5
E07D:E0
E087:E2
E0B0:E4
E0B8:E4
E0C0:E4
E0D2:E5
E127:E0
E12C:E7
E131:E2
E13C:E1
E14C:E2
E15D:E2
E183:E1
E188:E4
E195:E1
E1AF:E4
E1BC:E7
E1BF:E7
E1C2:EB
E1C7:E7
E1CA:E2
E1CD:E2
E1D2:E2
E1D7:E7
```

```
E1DC:E7
E1DF:E2
E1E9:60
E1F4:E2
E1F9:E2
E1FE:EB
E20A:EB
E213:EB
E216:E2
E21B:E2
E227:E7
E22C:E5
E22D:4D
E22E:4E
E22F:54
E230:68
E231:FF
E236:E5
E23B:E1
E240:E4
E245:E1
E24A:E6
E24F:E3
E254:E2
E259:EB
E25E:E3
E263:E0
E268:E5
E26D:F1
E272:E6
E273:FF
E274:FF
E275:FF
E277:E1
E27C:E0
E281:EB
E285:E2
E2BF:E2
E2D6:10
E2DF:EC
E2E1:83
E2E2:C0
E2E3:AD
E2E4:83
E2E5:C0
E2E6:A9
E2E7:00
E2E8:85
E2E9:D9
E2EA:4C
E2EB:08
E2EC:E3
E2ED:AD
E2EE:83
```

```
E2EF:C0
E2F0:AD
E2F1:83
E2F2:C0
E2F3:4C
E2F4:75
E2F5:E3
E323:E3
E32D:EB
E332:E7
E334:E0
E336:E0
E338:E0
E33A:E3
E33C:E0
E33F:F1
E342:E2
E345:E0
E34A:E7
E360:E7
E374:E7
E379:E7
E38D:E0
E3BB:E3
E3F1:E4
E3FA:E3
E41B:E4
E41E:E4
E423:E4
E426:E4
E42D:E7
E436:E3
E437:99
E438:07
E44D:E3
E462:E3
E46F:E4
E474:E4
E478:E4
E4A3:E4
E4C6:E4
E4EA:E4
E4EF:E4
E4F3:E0
E54B:E4
E558:E4
E586:E0
E589:E1
E590:E5
E59A:E4
E5D0:E7
E5DB:E2
E5F6:E2
E5F9:E2
```

```
E5FC:E2
E601:E2
E624:E2
E646:E1
E649:E5
E64F:E6
E654:E6
E685:E6
E694:E6
E69C:E6
E6EC:E7
E6EF:E1
E6FF:E6
E707:E6
E733:E0
E740:E7
E745:E7
E74A:E7
E751:E7
E754:EA
E757:E0
E75E:E7
E761:E7
E770:E7
E774:EB
E783:E7
E789:E7
E7B1:F1
E7BA:EC
E7BD:F1
E7CA:F2
E7E9:E2
E7EC:E2
E7F1:E2
E7FA:F2
E7FF:E7
E806:E2
E81A:E2
E825:EA
E828:E2
E831:E2
E836:E2
E83E:F2
E84B:E6
E850:E6
E855:E6
E85A:E6
E85F:E6
E864:E6
E869:E6
E870:E7
E873:E8
E882:E2
E887:E2
```

```
E88D:E9
E890:EA
E893:E7
E89C:E2
E8A5:E9
E8A8:EA
E8BE:E9
E8C3:E8
E8C6:E9
E8C9:E8
E8CC:E2
E8D0:E2
E8D7:EA
E8DD:E9
E8E0:E8
E8E3:E9
E8E6:E2
E8ED:EA
E8F0:E9
E8F3:E2
E8FA:E2
E90F:EA
E920:E2
E929:E2
E936:E8
E93F:E8
E94C:E9
E951:E8
E956:E9
E961:E9
E96C:E9
E975:E2
E980:E2
E987:E2
E990:E9
E997:E2
E9A0:E2
E9B1:E9
E9B6:E9
E9BB:E8
E9BE:E9
E9D0:E2
E9DA:E2
E9DD:EA
EA01:F2
EA10:E7
EA13:EA
EA16:E7
EA3C:F2
EA53:E8
EA57:EF
EA5A:EA
EA65:F2
EA72:F2
```

```
EA7D:EA
EA8B:E1
EA94:EC
EA99:ED
EA9E:ED
EAB5:E8
EAB8:E9
EABF:E2
EAC4:E2
EAD5:E2
EAF2:E2
EAFA:E2
EB11:EC
EB16:ED
EB21:F0
EB24:EB
EB3C:EA
EB4C:EA
EB53:E1
EB56:E2
EB5D:E2
EB6A:E9
EB73:EB
EB78:EB
EB7E:E2
EB8D:E2
EBA4:EA
EBAA:EA
EBB7:E7
ECCF:E2
ECDA:E1
ECF7:E2
ED10:E2
ED5C:ED
EDC1:E8
EE0E:E1
EE4D:E8
EE6D:EA
EE70:E7
EE7E:E7
EE8B:E7
EEBD:E7
EEC0:E7
EEC3:EE
EEC6:E7
EECF:E7
EEDD:E7
EEE2:E7
EEFC:EF
EF0F:E7
EF14:EF
EF17:ED
EF2B:EF
EF36:EF
```

```
EF41:EF
EF46:EF
EF4B:EF
EF59:EF
EF5E:EF
EF63:EF
EF68:EF
EF6B:E8
EFB1:EF
EFB4:ED
EFB7:ED
EFCA:EF
EFCD:EF
F001:E7
F006:E7
F012:E2
F01A:E7
F01D:EA
F02A:F0
F02D:EA
F030:EF
F03B:E8
F042:E2
F04F:E2
F05A:E9
F05D:F0
F060:E2
F065:E2
F068:EB
F06D:EB
F074:E9
F077:F0
F07A:E2
F07F:E9
F084:E8
F096:EA
F0A3:EA
F0A6:E8
F0B4:E8
F0B7:E2
F0C6:E2
F0C9:EA
F0D6:E9
F0D9:E9
F0DC:E9
F0DF:F1
F0E4:F1
F0E9:F1
F0EE:F1
F0F3:E1
F0F8:F1
F0FD:F1
F102:F1
F107:F1
```

```
F11F:EF
F122:F1
F127:F1
F132:EA
F135:F1
F144:E7
F149:E7
F161:EA
F164:F1
F169:EF
F172:E7
F175:F2
F17C:E7
F17F:EE
F183:F1
F186:E7
F189:F1
F190:EB
F19B:F1
F1A0:F1
F1A5:F1
F1AA:F1
F1AF:F1
F1B7:F2
F1BE:E7
F1C1:E7
F1C4:EA
F1C9:EA
F1FC:EF
F1FF:F1
F202:E8
F205:F0
F208:E2
F216:F2
F219:E7
F21C:E8
F21D:80
F225:F0
F22A:F1
F22F:F0
F234:F1
F239:F0
F23E:F0
F243:F0
F248:F2
F24D:F0
F252:F1
F257:F0
F25C:EE
F261:E0
F4FB:9D
BSAVE LANGASM,A$E000,L$2000
E000G
```

```
================================================================================
DOCUMENT :AAL-8201:DOS3.3:READ.EXEC.FILE.txt
================================================================================
```

```
d≤READ EXEC FILE3ÇD$–Á(4):Ñ"FILE NAME:
";F$]å∫D$"NOMONCIO":∫D$"OPEN"F$:∫D$"READ"F$gñ•´220r†å64874|™I–511†¥I  –
I»1:C–,(I):∫Á(C);:≠C—œ141ƒ180©æ´160ø‹å…3288:∫D$"CLOSE"d
```

```
=======================================================================
DOCUMENT :AAL-8201:DOS3.3:S.HiresScrnClr.txt
=======================================================================

 1000  *--------------------------------
 1010  *       HI-RES SCRN FUNCTION WITH COLOR
 1020  *
 1030  *       BY DAVID DOUDNA, FERGUSON, MISSOURI
 1040  *       NOVEMBER 30, 1981
 1050  *--------------------------------
 1060  HBASL  .EQ $26      BASE ADDRESS
 1070  HBASH  .EQ $27
 1080  HMASK  .EQ $30      BIT MASK
 1090  *--------------------------------
 1100  X0L    .EQ $320     X-COORDINATE
 1110  X0H    .EQ $321
 1120  Y0     .EQ $322     Y-COORDINATE
 1130  HCOLOR.BYTE .EQ $324
 1140  HPAGE  .EQ $326     HI-RES PAGE ($20 OR $40)
 1150  *--------------------------------
 1160  HSCRN  LDA Y0       GET (A)=Y-COORDINATE
 1170         LDX X0L      GET (Y,X)=X-COORD.
 1180         LDY X0H
 1190         PHA          Y-COORD BITS LABELED ABCDEFGH
 1200         AND #$C0     CALCULATE BASE ADDRESS FOR Y-COORD
 1210         STA HBASL         IN HBASL,HBASH FOR
 1220         LSR               ACCESSING SCREEN MEMORY
 1230         LSR               VIA (HBASL),Y
 1240         ORA HBASL    HBASH = PPPFGHCD
 1250         STA HBASL    HBASL = EABAB000
 1260         PLA          WHERE PPP=001 FOR $2000-3FFF
 1270         STA HBASH      AND PPP=010 FOR $4000-5FFF
 1280         ASL
 1290         ASL
 1300         ASL
 1310         ROL HBASH
 1320         ASL
 1330         ROL HBASH
 1340         ASL
 1350         ROR HBASL
 1360         LDA HBASH
 1370         AND #$1F
 1380         ORA HPAGE
 1390         STA HBASH
 1400  *--------------------------------
 1410         TXA          DIVIDE X-COORD BY 7 (7 DOTS PER BYTE)
 1420         CPY #0       IS X-COORD > 255?
 1430         BEQ .2       NO, ENTER SUBTRACTION LOOP
 1440         LDY #35      YES: 256 = 7*36 + 4
 1450         ADC #4       CARRY WAS SET, SO ADDS 5
 1460  *                   ALSO CLEARS CARRY, SO SBC #7 BELOW
 1470  *                   ACTUALLY SUBTRACTS 8
 1480  .1     INY          INCREASE QUOTIENT
```

```
1490  .2     SBC #7       SUBTRACT 7 (OR 8 IF CARRY CLEAR)
1500         BCS .1       STILL MORE 7'S
1510         TAX          REMAINDER IS BIT POSITION
1520         LDA MSKTBL-249,X
1530         STA HMASK
1540  *--------------------------------
1550         LDA (HBASL),Y     GET BYTE WHICH HAS OUR SPOT
1560         AND #$80     ISOLATE HALF-DOT SHIFT BIT
1570         STA HIBIT
1580         LDA (HBASL),Y     GET BYTE AGAIN
1590         AND HMASK    ISOLATE OUR SPOT
1600         BEQ .9       COLOR IS BLACK (0 OR 4)
1610         LDA X0L      NOT BLACK
1620         LDX #1
1630         LSR          ODD OR EVEN X-COORD.?
1640         BCS .3       ODD, COLOR=1 OR 5
1650         INX          EVEN, COLOR=2 OR 6
1660  *--------------------------------
1670  .3     LDA HMASK    LOOK AT NEIGHBOR BIT ON LEFT
1680         LSR          BITS ARE IN BYTE BACKWARDS
1690         BCC .4       NEIGHBOR IN SAME BYTE
1700         TYA          NEIGHBOR IN DIFFERENT BYTE
1710         BEQ .5       NO BYTE LEFT OF THIS ONE
1720         DEY
1730         LDA (HBASL),Y
1740         AND #$40
1750         BNE .7       WHITE
1760         INY          RESTORE Y
1770         BNE .5       ...ALWAYS
1780  .4     AND (HBASL),Y
1790         BNE .7       WHITE
1800  *--------------------------------
1810  .5     LDA HMASK    LOOK AT NEIGHBOR BIT ON RIGHT
1820         ASL
1830         BPL .6       NEIGHBOR IS IN SAME BYTE
1840         CPY #39      ALREADY AT RIGHT END?
1850         BCS .8       YES, NOT WHITE THEN
1860         INY
1870         LDA (HBASL),Y
1880         AND #1
1890         BNE .7       WHITE
1900         BEQ .8       ...ALWAYS (NOT WHITE)
1910  .6     AND (HBASL),Y
1920         BEQ .8       NOT WHITE
1930  *--------------------------------
1940  .7     LDX #3       COLOR IS WHITE (3 OR 7)
1950  *--------------------------------
1960  .8     TXA          COLOR TO A-REG
1970  *--------------------------------
1980  .9     BIT HIBIT    SEE IF HALF DOT SHIFT
1990         BPL .10      NO
2000         CLC
2010         ADC #4       YES
2020  .10    STA HCOLOR
```

```
2030        TAX          USE COLOR # (0-7) TO GET COLOR BYTE
2040        LDA COLOR.TABLE,X
2050        STA HCOLOR.BYTE
2060        RTS
2070 *--------------------------------
2080 MSKTBL .HS 01020408102040
2090 *--------------------------------
2100 COLOR.TABLE .HS 002A557F80AAD5FF
2110 *--------------------------------
2120 HIBIT  .BS 1        MSB
2130 HCOLOR .BS 1        COLOR INDEX 0-7
```

```
=====================================================================
DOCUMENT :AAL-8201:DOS3.3:S.RELOCATE.txt
=====================================================================

 1000  *--------------------------------
 1010  *      6502 RELOCATION SUBROUTINE
 1020  *--------------------------------
 1030  *      MAY BE LOADED ANYWHERE, AS IT IS SELF-RELOCATABLE
 1040  *--------------------------------
 1050  *      ADAPTED FROM SIMILAR PROGRAM IN PROGRAMMERS AID #1
 1060  *      ORIGINAL PROGRAM BY WOZ, 11-10-77
 1070  *      ADAPTED BY BOB SANDER-CEDERLOF, 12-30-81
 1080  *      (ELIMINATED USAGE OF SWEET-16)
 1090  *--------------------------------
 1100  MON.YSAV   .EQ $34  COMMAND BUFFER POINTER
 1110  MON.LENGTH .EQ $2F  # BYTES IN INSTRUCTION - 1
 1120  MON.INSDS2 .EQ $F88E  DISASSEMBLE (FIND LENGTH OF OPCODE)
 1130  MON.NXTA4  .EQ $FCB4     UPDATE POINTERS, TEST FOR END
 1140  MON.RETURN .EQ $FF58
 1150  STACK      .EQ $0100     SYSTEM STACK
 1160  INBUF      .EQ $0200     COMMAND INPUT BUFFER
 1170  *--------------------------------
 1180  A1     .EQ $3C,3D
 1190  A2     .EQ $3E,3F
 1200  A4     .EQ $42,43
 1210  R1     .EQ $02,03
 1220  R2     .EQ $04,05
 1230  R4     .EQ $08,09
 1240  INST   .EQ $0A,0B,0C
 1250  *--------------------------------
 1260  START  LDA #$4C     JMP OPCODE
 1270         STA $3F8     BUILD CONTROL-Y VECTOR
 1280         JSR MON.RETURN    FIND OUT WHERE I AM FIRST
 1290  START1 TSX
 1300         DEX          POINT AT LOW BYTE
 1310         SEC          +1
 1320         LDA STACK,X  LOW BYTE OF START1-1
 1330         ADC #RELOC-START1
 1340         STA $3F9
 1350         LDA STACK+1,X   HIGH BYTE OF START1-1
 1360         ADC /RELOC-START1
 1370         STA $3FA
 1380         RTS
 1390  *--------------------------------
 1400  RELOC  LDY MON.YSAV COMMAND BUFFER POINTER
 1410         LDA INBUF,Y  GET CHAR AFTER CONTROL-Y
 1420         CMP #$AA     IS IT "*"?
 1430         BNE RELOC2   NO, RELOCATE A BLOCK
 1440         INC MON.YSAV YES, GET BLOCK DEFINITION
 1450         LDX #7       COPY A1, A2, AND A4
 1460  .1     LDA A1,X
 1470         STA R1,X
 1480         DEX
```

```
1490        BPL .1
1500        RTS
1510 *--------------------------------
1520 RELOC2 LDY #2       COPY NEXT 3 BYTES FOR MY USE
1530 .1     LDA (A1),Y
1540        STA INST,Y
1550        DEY
1560        BPL .1
1570        JSR MON.INSDS2  GET LENGTH OF INSTRUCTION
1580        LDX MON.LENGTH  0=1 BYTE, 1=2 BYTES, 2=3 BYTES
1590        BEQ .3       1-BYTE OPCODE
1600        DEX
1610        BNE .2       3-BYTE OPCODE
1620        LDA INST     2-BYTE OPCODE
1630        AND #$0D     SEE IF ZERO-PAGE MODE
1640        BEQ .3       NO (X0 OR X2 OPCODE)
1650        AND #$08
1660        BNE .3       NO (80-FF OPCODE)
1670        STA INST+2   CLEAR HIGH BYTE OF ADDRESS FIELD
1680 *--------------------------------
1690 .2     LDA R2       COMPARE ADDR TO END OF SOURCE BLOCK
1700        CMP INST+1
1710        LDA R2+1
1720        SBC INST+2
1730        BCC .3       ADDR > SRCEND
1740        SEC          COMPARE ADDR TO BEGINNING OF SRC
1750        LDA INST+1
1760        SBC R1
1770        TAY
1780        LDA INST+2
1790        SBC R1+1
1800        BCC .3       ADDR < SRCBEG
1810        TAX
1820        TYA          ADDR = ADDR-SRCBEG+DESTBEG
1830        CLC
1840        ADC R4
1850        STA INST+1
1860        TXA
1870        ADC R4+1
1880        STA INST+2
1890 *--------------------------------
1900 .3     LDX #0       COPY MODIFIED INSTRUCTION TO DESTINATION
1910        LDY #0
1920 .4     LDA INST,X   NEXT BYTE OF THIS INSTRUCTION
1930        STA (A4),Y
1940        INX
1950        JSR MON.NXTA4    ADVANCE A1 AND A4, TEST FOR END
1960        DEC MON.LENGTH  TEST FOR END OF THIS INSTRUCTION
1970        BPL .4       MORE IN THIS INSTRUCTION
1980        BCC RELOC2   END OF SOURCE BLOCK
1990        RTS
```

```
=====================================================================
DOCUMENT :AAL-8201:DOS3.3:WRITE.EXEC.FILE.txt
=====================================================================

d≤WRITE EXEC FILE4ÇD$–Á(4):Ñ"FILE NAME:
";F$Rå∫D$"OPEN"F$:∫D$"DELETE"F$oñ∫D$"OPEN"F$:∫D$"WRITE"F$û†∞500:≠,(512
)–175Õ,(513)–170Õ,(514)–141ƒ220®™I–511Ã¥I–I»1:C–,(I):∫Á(C);:≠C —
œ141ƒ180’æ´160‰‹∫D$"CLOSE"ÍÊÄ  Ù≤INPUT A LINE WITHOUT DOS KNOWING)
     ˘ã0:ä0:å64874:å1002:±d
```

==================================================================
DOCUMENT :AAL-8202:Articles:BMA.VERSES.txt
==================================================================

Wisdom for Daily Living

Assignment 3

The Search for Wisdom -- The Responsibility of Man

--------------------------------------------------------

Proverbs 4:5-9

Get wisdom, get understanding:  forget it not;
neither decline from the words of my mouth.

Forsake her not, and she shall preserve thee;
Love her, and she shall keep thee.

Wisdom is the principal thing; therefore get wisdom:
and with all thy getting, get understanding.

Exalt her, and she shall promote thee:
she shall bring thee to honor, when thou dost embrace her.

She shall give to thine head an ornament of grace:
a crown of glory shall she deliver to thee.

Proverbs 4:5-9

----------------------------------------------------------

Proverbs 16:16

How much better is it to get wisdom than gold!
and to get understanding rather to be chosen than silver!

Proverbs 16:16

----------------------------------------------------------

James 1:5

If any of you lack wisdom, let him ask of God,
that giveth to all men liberally, and upbraideth not;
and it shall be given him.

James 1:5

----------------------------------------------------------

The Search for Wisdom -- The Responsibility of Man

Assignment 3

Wisdom for Daily Living

=============================================================================
DOCUMENT :AAL-8202:Articles:DOS.Error.Trap.txt
=============================================================================

DOS Error Trapping from Machine Language........Lee Meador

I have been working on a text editor program for about three years now at the World Bible Translation Center.  It allows us to edit in any two of the following languages:  English, Russian, Greek, Hebrew, and Arabic.  Hebrew and Arabic move from right to left across the screen, as they should.

Recently we have been making some enhancements to this multi-lingual text editor (called ALPHONSE)  which include support of two disk drives (a program disk in drive 1 and a data disk in drive 2).  But we didn't want to require the use of two drives.  That means a routine must look on the various disks to see if the data is there.  We can do this very handily by RENAMEing a certain file -- call it FILE -- and assuming that a DOS error means that the data isn't on that disk. Then we can look on other drives and finally, if it isn't found anywhere, we can prompt the user to put in the data disk.  Then we look again -- and so on.

A problem with this is that I need to trap from assembly language any DOS errorswhich occur, but I want to return to the program if the user accidentally types the RESET key (with ALPHONSE it will always be accidentally).  A second use for DOS error trapping came up because I/O errors in a disk file print the error message but do not change from the HIRES page to the text page.  That makes it rather difficult to see what the error is -- especially for the less advanced user, who has no idea what is happening.

Here is a program listing of ALPHONSE with all the insides removed. Where the real program would have large sections of code, I have instead comments that look like this:

```
*->->->->->->->->->->->->->->->->->->->->
* DO SOME ACTION WHICH IS NOT SHOWN
*->->->->->->->->->->->->->->->->->->->->
```

(Of course, ALPHONSE is about 8K long and the listing is nearly 1/2-inch thick, so this isn't the whole listing.)

MAIN PROGRAM OUTLINE

Here is an outline of the main program:

```
MACH:   GLOBAL INITIALIZATION;
REENT:  LOCAL INITIALIZATION;
        REPEAT
          READ EDITOR COMMAND;
          PROCESS EDITOR COMMAND;
        UNTIL EDITOR COMMAND = QUIT;
```

END.

In the global initialization we have to do four things related to error-trapping:

1. Call SETUP.DOS.TABLE to copy my addresses into the table at $9D56 of DOS. This makes DOS come back to my program when any soft entry of a funny DOS command occurs. Just calling SETUP.DOS.TABLE will not really trap any errors, but it will keep DOS from terminating your program if a DOS error does occur (that usually means SYNTAX ERROR, I/O ERROR, or FILE NOT FOUND).

2. Call CLEAR.ERROR to initialize the ONERR trapping mechanism in my program.

3. Call ON.ERROR with the address of the error-handling routine in the A and Y registers (LO, HI). This sets up the DOS error-handling capabilities as if Applesoft were running and ONERR were set.

4. After doing all the global initialization of files and such, we need to call OFF.ERROR to turn off the error handling that ON.ERROR set up. After calling OFF.ERROR any DOS error will beep and go to the soft entry point. (We have already set the soft entry point in step one to be MY.RESET.)

In the local initialization we take care of a few more things that have to be done every time the program is run -- not just the first time. The call to OFF.ERROR cleared any error trapping so we can call SETUP.DOS.TABLE and CLEAR.ERROR again without causing any problems.

Note that the call to LOOK.FOR.FILE changes the error address so we have to call ON.ERROR with MY.ERROR again to make sure that an error doesn't throw us off into never-never land. LOOK.FOR.FILE returns the carry clear if FILE is found. Carry set signals that the file isn't on any available drives; in that case, ALPHONSE would print a message like "INSERT DATA DISK AND HIT ANY KEY," then wait for a key to be pushed and call LOOK.FOR.FILE another time.

The main program loop is not really of interest here, but it is shown in the listing in skeleton form.

SUB-PROGRAMS

Now, how do the subroutines work? First, the one that you wouldn't use in your program: LOOK.FOR.FILE has to save the stack pointer. This is because we expect DOS errors to occur inside the routine. A DOS error will mess up the stack. Saving the stack lets us remember where we were. (By the way, DOS just adds things to the stack and never removes them when there is an error. The LOOK.FOR.FILE return addresses will not be messed up.)

LOOK.FOR.FILE sets its own DOS error trap address. Then the program looks through trying to find FILE on the various slots and drives. It does this by printing the DOS commands <CTRL-D>, RENAME FILE, FILE, Sx, Dy with x and y filled in. Appropriate values for x are six, five, and seven; y would be one or two. The order in which you try the slot/drive combinations will determine which of two disks are chosen if you put two data disks in at the same time. I used a table of six slot/drive combinations to choose the order and positions to try. Notice that before printing the DOS RENAME command, I had to check to see if there was a disk card in the slot. Choosing a slot without a disk card in it for a DOS command will cause DOS to hang when you try the next DOS command with a different slot. DOS is waiting for the last drive to quit running. Little does DOS know that an empty slot always seems to be running (to DOS at least).

If the DOS RENAME command fails or there is no disk card in the slot, LOOK.FOR.FILE will jump to LOOK.ERR to loop and try the next slot/drive. If it runs out of slot/drives the program returns with carry set to indicate FILE was not found. Carry clear indicates that the last-used drive has FILE on it.

There are several routines you might want to copy as is to your program. Calling them takes care of error trapping and reset trapping.

SETUP.DOS.TABLE: copies MY.TABLE into DOS to jump to my program on any DOS error or RESET. Unfortunately, at this point you can't tell them apart.

ON.ERROR: sets the error address to the value in the A, Y (LO, HI) registers. When a DOS error occurs after ON.ERROR has been called, DOS will jump to this address with the error number in the X register. All other registers will have been changed.

OFF.ERROR: turns off the error trapping and resets DOS to the state it was in before ON.ERROR was first called. SAVE.AAB6 is used to keep track of which BASIC language DOS thinks was active. Restoring AAB6 before exiting your program will help DOS keep things sorted out. Calling OFF.ERROR restores AAB6. (By the way, while ON.ERROR is active, DOS thinks that Applesoft is currently running a program and that there has been an ONERR statement. Zero page locations $D8, $76, and $33 are used for this.)

CLEAR.ERROR: call this the first thing in your program to set up the flags used by ON.ERROR and OFF.ERROR.

Note: MY.RESET just reenters the program loop if someone types the RESET key. That makes it a null key. MY.ERROR should be looked at to see how the DOS error message comes back to you. You can use the message to print various messages depending upon what is wrong. Or, you can take various actions depending upon the error message. Pages 114-115 of the DOS manual show what the various error numbers are that come back in the X register.

The program listing should show how most of these things are handled.

```
==============================================================================
DOCUMENT :AAL-8202:Articles:EvenFstrPrimes.txt
==============================================================================
```

Even Faster Primes.............................Charles Putney
[  Charlie is a long-time friend and subscriber in Ireland  ]

Bob, I wanted to answer your challenge in the Ocotber 1981 AAL for some time, but this is the first chance I had.  You sifted out the primes in 690 milliseconds, and challenged readers to beat your time. I did it!

I increased the speed by using a faster algorithm, and by using some self-modifying code in the loops.  I know self-modifying code is dangerous, and a NO-NO, but it amounts to about 50 milliseconds improvement.

The algorithm changes are an even greater factor.  The main ideas for the sieve are:

1.  Only check odd numbers
2.  Get next increment from the prime array.
    This means you only knock out primes.
3.  Start knocking out at P^2.  That is,
    if prime found is 3, start at 9.
4.  Increment the knock-out index by 2*P.
    This avoids knocking out even numbers.
5.  Stop at the square-root of the maximum number.

Your algorithm did all the above except 3 and 4.

With these routines, a generation takes 330 milliseconds.  This is over twice as fast as yours!

You could still shave a little time off by optimizing the square routine, and even including it inline since it is only called from one place.

I'll grant you that this is not the same algorithm, but the goal is to find primes fast.  I know throw down the glove for the next challenger!

==============================================================================
DOCUMENT :AAL-8202:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 2 -- Issue 5 February, 1982

In This Issue...



Renew Now, the Price is Going Up

If you renew your subscription before March 1, 1982, you can renew at the current rate of $12/year. Starting March 1st, the price will go up to $15/year (2nd class mail in the USA). Subscriptions sent First Class Mail to USA, Canada, and Mexico will be $18/year. Air Mail subscriptions to all other countries will be $28/year. The price for back issues will be $1.50 each (plus $1.00 postage outside of USA, Canada, and Mexico).

S-C MACRO Assembler II is almost here!

By the time you read this, I expect to be filling orders for the new MACRO version. This is what I have been calling Version 5.0, but I have decided to call it S-C MACRO Assembler II instead. Version 4.0 will still be sold at $55. The MACRO version will be $80. Owners of Version 4.0 can upgrade for only $27.50. There will be an all new manual, rather than the current 2-part manual.

The MACRO Assembler includes macros (of course!), conditional assembly, EDIT, COPY, global string replacement, and many more new features. And it assembles even faster than version 4.0!

================================================================
DOCUMENT :AAL-8202:Articles:Great.Free.Adv.txt
================================================================

Great Adventure................................Jeff Jacobsen

Have you ever played the ORIGINAL game?  Adventure game, that is? Adventure was originally developed by Willie Crowther and Don Woods in FORTRAN on a DEC PDP-10 computer.  It is the grandfather, or maybe great-grandfather by now, of the hundreds of Adventure games you see in the advertisements (you might even have bought some!).

I used the S-C Assembler II to write an Apple version of the original Adventure game.  By using text compression techniques, I was able to squeeze the entire game into 48K RAM.  The interaction is lightning fast, and nothing ever has to be found on the disk.  The whole game is in there:  over 130 rooms, 15 treasures, 40 useful objects, and 12 obstacles or opponents.

I will send you a copy FREE!  Just send me a blank diskette and postage.  Or send $5.00 and I will send the disk and pay the postage. Write to me, Jeff Jacobsen, at Frontier Computing Inc., P. O. Box 402, Logan, Utah 84321.

==========================================================================
DOCUMENT :AAL-8202:Articles:ImprvEpsonCard.txt
==========================================================================

Improving the Epson Controller.........Peter G. Bartlett, Jr.

[ I recently bought an NEC PC-8023 dot matrix printer, which has fabulous features. The store sold me an Epson controller to run it, assuring me it was all I needed. Naturally, they were wrong. To get all features, just as with the Epson printer, you need to be able to send 8-bit characters. I figured out how, and was just about to write an article about it, when the following one came from Peter Bartlett of Chicago, Illinois. (Bob Sander-Cederlof) ]

As you may know, the Epson MX-80 printer is somewhat hamstrung by the Epson Controller. Certain features, such as the "TRS-80" graphics character set, are not available. You can buy the Graftrax kit to enable dot graphics, but these built-in character graphics are still inaccessible.

The problem is in the card Epson makes to interface its line of printers with the Apple. (The problem is not present if you use a non-Epson card.) Hardware on the Epson controller card masks out the high-order bit, eliminating the ability to access the standard graphics characters and some of the dot graphics capabilities.

Epson's reasoning is that the Apple only sends characters with the high-bit set, so Epson has hardware on the card to mask out that bit. That way the normal characters print as they should.

If Epson had masked out the high bit in their printer driver routine instead, then machine language programmers like us could have accessed all the features of the printer. We could bypass the printer driver and work directly with the printer I/O port.

Fortunately, the card has jumpers that can be changed and the printer driver is on an EPROM that can be changed.

So you will need a soldering iron, an EPROM blaster, and an erased 2708 EPROM. [ The EPROM Blaster from Apparat can blow 2708's. I don't believe the Mountain Hardware ROMWRITER can. ]

On the Epson interface card, the jumper marked "P4" should be removed and installed on "M4" instead. This jumper are directly underneath the EPROM and are labelled. [ Ignore the three jumpers on the right side of the EPROM. ] This fix is not documented, although you can see it in the Schematic Drawing of the card. I called Epson on the phone, and they told me about it. With the jumper moved to M4, the high-bit is transmitted correctly.

BUT!!! Now normal characters do not print normally! Instead, you get the graphics characters! Okay, we need to modify the program inside the EPROM. At location $C120 (assuming the card is in slot 1) you will find an instruction "AND #$7F". This clears the high-bit for processing control codes only. We need to move this instruction to $C112, so that the hihg-bit is cleared for transmitted codes also. Here is a listing of the BEFORE and AFTER programs, with the moved instruction starred. (Note that the hex values for the BCC and BPL instructions changes too.)

<program here>

That fix in the program will clear the high-bit off every character sent via the printer driver to the printer. We are back where we started. Except that now the clever programmer can send characters directly to the printer, bypassing the EPROM resident driver. Here is how to send one character directly to the printer:

```
OUTPUT STA $C090   Assuming slot 1
.1     BIT #C1C1   Character picked up by printer?
       BMI .1      No, keep testing
```

I have tried everything above, and it all works perfectly. I hope it proves useful to lots of you AAL readers.

```
==================================================================
DOCUMENT :AAL-8202:Articles:My.Ad.txt
==================================================================

S-C MACRO ASSEMBLER II.......................................$80.00
S-C ASSEMBLER II Version 4.0.................................$55.00
Upgrade from Version 4.0 to MACRO............................$27.50
    Includes Manual, Diskette with Assembler and sample programs,
    and Quick Reference Card.  Call for more information.

Source code of Version 4.0 on disk...........................$95.00
    Fully commented, easy to understand and modify to your own tastes.

Cross Assembler Patches for 6809.............................$22.50
    Requires possession of Version 4.0.  Enables you to develop
    programs for the Motorola 6809 CPU.  (The MILL from Stellation,
    EXCEL-9 from ESD Laboratories, or the Radio Shack Color Computer.)

Cross Assembler for 6800.....................................$22.50
    Requires possession of Version 4.0.  Enables you to develop
    programs for the Motorola 6800, 6801, and 6802 CPUs.

AAL Quarterly Disks.....................................each $15.00
    Each disk contains all the source code from three
    issues of "Apple Assembly Line", to save you lots
    of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981

Double Precision Floating Point for Applesoft................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)..............$65.00
Source Code for FLASH! Runtime Package.......................$20.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00

Program Line Editor (Neil Konzen, Synergistic Software)......$40.00
    (Comes on DOS 3.2 disk.  I have added a second disk
    in DOS 3.3 format with a superior set of ESCAPE macros.)

S-C Games Disk (requires Integer BASIC)......................$15.00
S-C Games Disk (compiled by FLASH!, no need for Integer BASIC).....$25.00
    Includes 4x4x4 tic-tac-toe, lo-res space war, lo-res jig-saw
    puzzle, musical memory, pentominoes, and mastermind.

Blank Diskettes..................................package of 20 for $50.00
    With hub rings, bulk packaged, in plain white jackets.

Lower-Case Display Encoder ROM...............................$25.00
    Works only Revision level 7 Apples.  Replaces the encoder ROM.
    Comes with instructions.

Diskette Mailing Protectors........................10-99:  40 cents each
                                              100 or more:  25 cents each
    Corrugated folder specially designed for mailing mini-floppy
    diskettes.  Fits in standard 6x9-inch envelope.  (Envelopes
```

```
    5-cents each, if you need them.)

Zip-Lock Bags (2-mil, 6"x9")................................100 for $8.50
              (2-mil, 9"x12")..............................100 for $13.00

Books, Books, Books..........................compare our discount prices!
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "What's Where in the Apple", William Leubert.........($14.95)  $14.00
    "6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
    "Apple Assembly Language", Don & Kurt Inman..........($12.95)  $12.00

***We take Master Charge and VISA ***
```

================================================================================
DOCUMENT :AAL-8202:Articles:On.DivBy10.txt
================================================================================

On Dividing by Ten..............................Jim Church

Some time ago you asked readers to come up with subroutines to divide by ten (or multiply by one-tenth). I may have come up with the smallest one, although it is certainly not the fastest.

By using SWEET-16 to merely subtract 10 over and over, until the remainder is less than 10, and counting the number of subtractions, I can divide a 16-bit value by ten in a 10-byte subroutine!

In fact, you can divide by any 16-bit value. My program assumes the divisor is in $02,03 and the dividend is in $04,05. These are the Sweet-16 registers 1 and 2. The quotient will be left in $04,05; the remainder will be in $00,01.

I used a copy of Sweet-16 in RAM, from the source code on the S-C Assembler II disk. If you use the copy in the Integer BASIC ROM's, or in the RAM card Integer BASIC, change line 1130 to "SW16 .EQ $F689".

Here is the program listing:

========================================================================
DOCUMENT :AAL-8202:Articles:Overseas.Subs.txt
========================================================================

We are now sending AAL to over 800 subscribers.  Of course most of these are in the U.S.A., but an increasing number are subscribing from other countries.  We now have:

```
15 -- Canada            2 -- Hong Kong
5 -- Sweden            2 -- Ireland
5 -- New Zealand       1 -- Israel
4 -- France            1 -- Italy
4 -- Japan             1 -- Netherlands
3 -- Australia         1 -- Qatar
3 -- England           1 -- Saudi Arabia
3 -- West Germany      1 -- Spain
3 -- South Africa      1 -- Thailand
2 -- Argentina         1 -- Turkey
2 -- Belgium
```

And there are also at least a half dozen subscribers with APO addresses, who are stationed in strange exotic lands.

==============================================================================
DOCUMENT :AAL-8202:Articles:Patch.AW.PLE.txt
==============================================================================

Patches for Applewriter to Unhook PLE......Bob Sander-Cederlof

If you use Applewriter a lot, like I do.... And if you use Neil Konzen's Program Line Editor (PLE) a lot, like I do.... Then you probably have at least once tried to BRUN TEDITOR while PLE was still installed, like I have....

The result is maddening, to say the least. Everything seems fine. You can load a file into Applewriter, or enter a new one. You can edit to your hearts content. Then you try to SAVE it on disk. POW! What happened?!! Since PLE is still hooked into DOS, it needs to remain unmolested in memory. But Applewriter ignores its presence, and puts the text right over the top of it.

I thought I had finally learned my lesson, but then I did it again!

Finally, I decided to make Applewriter unhook everything that PLE might have hooked in, during initialization. It turned out to be surprisingly easy. Here are the patches. I haved moved them up high enough so that if you have the lower case patches installed there is no conflict. Now I do not need to reboot to get rid of PLE.

```
================================================================
DOCUMENT :AAL-8202:Articles:PrinterFIFOBuf.txt
================================================================
```

Printer Handler with FIFO Buffer................Jim Kassel
[  Jim Kassel is a subscriber from St. Paul, Minnesota.  ]

Before I get on with technical discussions, first let me say that I have had a ball using S-C Assembler II Version 4.0.  It definitely has earned a place on the list of "The Greatest Things Since Sliced Bread."  My current version incorporates the block move and copy feature described in the December '80 and January '81 issues of AAL which have been a welcome enhancement.

Now...on with the article, about a super simple programming technique that I have used extensively.  I am a hardware logic designer by trade and before the introduction of First In-First Out (FIFO) memory chips, designers had to implement that function using an input address up-counter, an output address up-counter, and an up/down counter to determine character count.  Now that the FIFO chips are available, they are still a bit expensive for home computer use.  By using the old counter method implemented in software, not only is the FIFO free but also extrememly expandable in size (within the bounds of the computer memory, of course).

I am going to give a little background into the necessity, in my case, for using this technique.  I feel that the problem I experienced may be interesting reading to others who may have had similar occurrences.

I was writing an assembly language program that would allow my Apple II to become a terminal, using the Hayes Micromodem II and Epson MX-80 printer/Orange Micro Grappler interface card.

For the sake of versatility I would have preferred to perform operations like JSR $Cx00 (x = slot number) when transferring data with these devices.  However, it became apparent that I would have to bypass the firmware on the other interface cards.  This was especially true with the printer interface card.  Because the printer takes 1-2 seconds to print out a line of characters, the interface becomes unavailable for storage.  Since the modem wants to supply characters at a rate of up to 30 cps, at least that many characters were being "dropped on the floor" while the printer interface card kept program control.

I finally had to get the schematics and/or firmware disassemblies of the other interface cards.  From them I figured out the addresses of, and the methods of communications for, the various control, data, and (most important) status registers.  This allowed me to check for printer busy, modem transmit register not yet empty and modem receive register not yet full.  Now I could do other things when no data could be transferred.  No longer would I have to be a slave to the equipment that is used for support!

The only other problem, then, was to be able to save the print characters in a FIFO print buffer so they would not be forgotten while the printer was busy printing the previous line of characters. In my version I allow a whole page of memory ($94) to be used for the buffer space. As long as there is not a horribly long burst of received carriage returns (the slowest printer operation), 256 locations is more than adequate because the MX-80 prints at least twice as fast as the modem data rate. Plus non-control characters are transferred into the printer line buffer much faster than incoming modem characters and the FIFO almost always stays empty because of this.

As characters arrive from the modem they are placed into the FIFO (by executing a JSR PRINT.FIFO.INPUT), then the input index (PBII) and the character counter (PBCC) are incremented. Whenever the program is in a wait loop (keyboard entries, modem data transfers, etc.) there are no less than 33 milliseconds (300 baud/30 cps) to do non-critical operations. This is more than enough time to execute a JSR PRINT.FIFO.OUTPUT.1 instruction during each "round trip" of the wait loops. If the printer is busy, the program is returned to with no data transferred; if the printer is not busy, the program is returned to after the next FIFO output character is sent, the output index (PBOI) is incremented, and the character counter (PBCC) is decremented. In any case, the program does not depend on the outcome of the subroutine results. The subroutines maintain their independence by correctly updating and monitoring the character counter (PBCC).

In my version, I must append a line feed (<LF>) character to every carriage return (<CR>) that is sent. I check every FIFO input character to see if it is a <CR>. If so, I store a <LF> into the next FIFO input location. Note that if I had decided to send the <LF> directly to the printer by monitoring for the <CR> in the FIFO output subroutine, I would again have been a slave to the printer while waiting for it to become unbusy with the <CR> operation.

By making PRINT.FIFO.OUTPUT.2 a separate subroutine, I could write it for any printer interface card with data and status registers and still not require any changes to subroutines PRINT.FIFO.INPUT and PRINT.FIFO.OUTPUT.1. This provided some versatility for converting the program for some friends with different interfaces.

==============================================================================
DOCUMENT :AAL-8202:Articles:Problem.QD5.txt
==============================================================================

Problem with QD#5

The first 14 copies that I sent out of Quarterly Disk #5 were incomplete.  I forgot to include PMD and FPSUBS.  If you have one of those with serial #1 thru #14, send it back; I will add the programs and return it.  I'm sorry!

==============================================================================
DOCUMENT :AAL-8202:DOS3.3:AW.Patch4PLE.txt
==============================================================================

```
 1000  *--------------------------------
 1010  * APPLEWRITER PATCH TO UNHOOK PLE
 1020  *--------------------------------
 1030         .OR $803
 1040         .TF AW.1
 1050         JSR PATCH    REPLACES "JSR $10F8"
 1060  *--------------------------------
 1070         .OR $1873    SAFE PATCH AREA
 1080         .TF AW.2
 1090  PATCH  JSR $FE89    SET INPUT TO KEYBOARD
 1100         JSR $FE93    SET OUTPUT TO SCREEN
 1110         LDA #$9C
 1120         STA $9D01    RESTORE NORMAL DOS BUFFERS
 1130         LDA #3
 1140         STA $AA57    MAXFILES=3
 1150         JSR $A7D4
 1160         LDX #$2F
 1170  .1     LDA $9E51,X  RESTORE PAGE 3 POINTERS
 1180         STA $3D0,X
 1190         DEX
 1200         BPL .1
 1210         JSR $3EA     RE-HOOK DOS
 1220         JMP $10F8    DO WHAT THE "JSR PATCH" COVERED
```

```
==============================================================================
DOCUMENT :AAL-8202:DOS3.3:PutneyPrimeDrvr.txt
==============================================================================

\
â:ó:∫"CHARLES PUTNEY'S FASTER PRIME GENERATOR --------------------- -
----------------"y¢10:ñ15:∫"LOADING . . ."Å -ë:â™(D$–Á(4):∫D$"BLOAD
B.PUTNEY'S PRIMES"–2ó:¢10:ñ10:∫"HIT ANY KEY TO
START"Í<π49168,0:æA$:π49168,0  Pπ49232,0:π49239,0
     Zå327686  _â:ÅA–8195¡24576«2:≠,(A)–0ƒ∫A…8192;" ";<  bÇP  d≤  PRIME
TESTERi   n≤  CHARLES H. PUTNEY     x≤  18 QUINNS ROADè  Ç≤  SHANKILL°
     å≤  CO. DUBLIN∞      ñ≤  IRELAND÷    †≤  TIME FOR 100 RUNS = 42
SECONDS
```

```
=========================================================================
DOCUMENT :AAL-8202:DOS3.3:S.DIVIDE.BY.TEN.txt
=========================================================================

 1000  *---------------------------------
 1010  * DIVIDE ANY NUMBER BY 10 OR BY *
 1020  * ANYTHING ELSE FOR THAT MATTER *
 1030  *                               *
 1040  * DIVIDEND - REGISTER 0  $00.01 *
 1050  * DIVISOR  - REGISTER 1  $02.03 *
 1060  * QUOTIENT - REGISTER 2  $04.05 *
 1070  *                               *
 1080  * EXAMPLE  - DIVIDE 65534 BY 10 *
 1090  * 00:FE FF 0A 00 00 00 N 300G   *
 1100  *                               *
 1110  *         JIM CHURCH            *
 1120  *---------------------------------
 1130         .OR $300
 1140  SW16   .EQ $9B89   (SWEET-16 ADDRESS IN RAM)
 1150  *---------------------------------
 1160  GO     JSR SW16
 1170  STILL.GREATER
 1180         SUB 1        DEDUCT DIVISOR FROM DIVIDEND
 1190         INR 2        ADD 1 TO QUOTIENT
 1200         CPR 1        DIVIDEND > DIVISOR?
 1210         BC  STILL.GREATER
 1220         RTN          LEAVE SWEET-16
 1230         RTS
 1240  LENGTH .EQ *-GO
 1250  *---------------------------------
 1260  * LOOK IN $00.01 FOR REMAINDER  *
 1270  *---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8202:DOS3.3:S.DOSOnErrXmpl.txt
==============================================================================
```

```
 1000         .OR $803
 1010  *--------------------------------
 1020  *
 1030  * ALPHONSE - MULTI-LINGUAL TEXT EDITOR
 1040  *
 1050  * Chopped up to show ERROR trapping
 1060  *  ala Applesoft ONERR command.
 1070  *  NOTE: There is no RESUME but
 1080  *  you are able to easily pick
 1090  *  up DOS errors and handle them
 1100  *  while disabling the RESET
 1110  *  (on AutoStart ROM).
 1120  *
 1130  * by Lee Meador
 1140  *
 1150  *       MACH - Main program entry
 1160  *       REENT- Program re-entry
 1170  *       ULOOP- Main program loop
 1180  *       MY.RESET- handle RESET key pushed
 1190  *       MY.ERROR- default error handler
 1200  *       END  - Exit to BASIC
 1210  *       SETUP.DOS.TABLE- hook in RESET trapping
 1220  *       ON.ERROR - set error trap
 1230  *       OFF.ERROR- kill error trap
 1240  *       CLEAR.ERROR- init error flags
 1250  *       LOOK.FOR.FILE- find S,D of FILE
 1260  *       MY.TABLE - copied into DOS table
 1270  *
 1280  *--------------------------------
 1290  DOS.TABLE .EQ $9D56
 1300  HOME.TEXT .EQ $FC58
 1310  TMP1      .EQ 0     PAGE 0
 1320  *--------------------------------
 1330  *
 1340  * THIS IS THE MAIN ENTRY POINT
 1350  *  FOR ALPHONSE.
 1360  *
 1370  *--------------------------------
 1380  MACH   JSR SETUP.DOS.TABLE
 1390         JSR CLEAR.ERROR  NO ONERR
 1400         LDA #MY.ERROR    THEN SET IT
 1410         LDY /MY.ERROR    .. TO MY.ERROR
 1420         JSR ON.ERROR
 1430         JSR HOME.TEXT CLR TXT SCR
 1440  *->->->->->->->->->->->->->->->->
 1450  * DO INITIALIZE PROCESSING
 1460  *->->->->->->->->->->->->->->->->
 1470         JSR OFF.ERROR ON ERR TURNED OFF
 1480  *--------------------------------
```

```
1490  * RE-ENTRY POINT. NORMAL ENTRY COMES HERE TOO
1500  *--------------------------------
1510  REENT  JSR SETUP.DOS.TABLE
1520         JSR CLEAR.ERROR NO ON ERR
1530  .10    JSR LOOK.FOR.FILE
1540         BCC LOAD.FILE
1550         LDA #MY.ERROR     SET ERROR
1560         LDY /MY.ERROR     .. TO MY.ERROR
1570         JSR ON.ERROR
1580         JSR HOME.TEXT CLEAR SCRN
1590  *->->->->->->->->->->->->->->->->
1600  * PRINT "INSERT CORRECT DISK"
1610  *->->->->->->->->->->->->->->->->
1620         JMP .10       TRY AGAIN TO FIND TEXT.DIR
1630  *--------------------------------
1640  LOAD.FILE
1650         LDA #MY.ERROR FIX ERROR HANDLER
1660         LDY /MY.ERROR
1670         JSR ON.ERROR
1680  *->->->->->->->->->->->->->->->->
1690  * THE REST OF INITIALIZING
1700  *->->->->->->->->->->->->->->->->
1710
1720  ULOOP
1730  *->->->->->->->->->->->->->->->->
1740  * MAIN PROGRAM LOOP DOES EACH
1750  *  COMMAND TYPED
1760  * EXIT COMMAND JUMPS TO "END"
1770  *->->->->->->->->->->->->->->->->
1780         JMP ULOOP    .. LOOP IF UNDEF
1790  *--------------------------------
1800  *
1810  * ROUTINE TO HANDLE USER HITTING
1820  *  RESET. (HANDLED IN DOS--DOS
1830  *  FIXES IT ON $3D3 EXIT.)
1840  * NO HOOKS TO CHANGE AND FIX BACK
1850  *
1860  *--------------------------------
1870
1880  MY.RESET
1890  *->->->->->->->->->->->->->->->->
1900  * RESET POINTERS AND HIRES PAGE2
1910  *->->->->->->->->->->->->->->->->
1920         JMP ULOOP
1930
1940  *--------------------------------
1950  *
1960  * MY GENERAL ERROR HANDLER JUST
1970  *  PRINTS "ERROR NUMBER " AND
1980  *  THE NUMBER FOR THE ERROR THEN
1990  *  EXITS TO WHATEVER BASIC WAS
2000  *  RUNNING BEFORE.
2010  *
2020  *--------------------------------
```

```
2030  MY.ERROR
2040         TXA          SAVE ERR NUM
2050         PHA
2060  *->->->->->->->->->->->->->->->->->
2070  * HOME SCREEN AND PRINT THE
2080  *  MESSAGE "ERROR NUMBER "
2090  *->->->->->->->->->->->->->->->->->
2100         PLA          ERR NUMBER
2110  *->->->->->->->->->->->->->->->->->
2120  * PRINT ACC AS DECIMAL NUMBER
2130  *  FOLLOWED BY A <RETURN>
2140  *->->->->->->->->->->->->->->->->->
2150  END    JSR OFF.ERROR FIX UP $AAB6
2160         JMP $3D3      HARD EXIT RESTORS DOS.TABLE
2170  *--------------------------------
2180  *
2190  * COPY MY ADDRESSES INTO THE DOS
2200  *  TABLE OF JUMPS (AT $9D56).
2210  *
2220  *--------------------------------
2230  SETUP.DOS.TABLE
2240         LDX #12      12 BYTES
2250  .10    LDA MY.TABLE-1,X
2260         STA DOS.TABLE-1,X
2270         DEX
2280         BNE .10
2290         RTS
2300  *--------------------------------
2310  *
2320  * DOS ERROR SETUP/RESET
2330  *
2340  *  CALL CLEAR.ERROR AT START OF
2350  *   PROGRAM TO SET UP FLAG
2360  *   (ITS ALSO OK AFTER OFF.ERROR)
2370  *  CALL ON.ERROR WITH A,Y HOLDING
2380  *   THE ADDRESS YOU WANT TO JUMP
2390  *   TO IF A DOS ERROR OCCURS.
2400  *  CALL OFF.ERROR TO CANCEL ERROR
2410  *   TRAPPING AND REVERT TO NORMAL
2420  *   ERROR MSG AND JUMP TO BASIC
2430  *
2440  *  WHEN THE ERROR ROUTINE IS
2450  *   CALLED (ON AN ERROR) THE X
2460  *   REGISTER HOLDS THE ERROR
2470  *   NUMBER AS LISTED P 114-115 OF
2480  *   THE DOS MANUAL.
2490  *
2500  *  AN ERROR WILL CAUSE THE STACK
2510  *   TO BE MESSED UP. SO, SAVE IT
2520  *   WHEN YOU EXPECT ERRORS.
2530  *
2540  *--------------------------------
2550  ON.ERROR
2560         STA DOS.TABLE+4
```

```
2570        STY DOS.TABLE+5
2580        LDA SAVE.AAB6     48K ONLY
2590        BNE .10
2600        LDX $AAB6         48K ONLY !!!!!
2610        DEX
2620        STX SAVE.AAB6     48K ONLY
2630  .10   LDA #$40     PRETEND AS([])
2640        STA $AAB6         48K ONLY
2650        ASL          $80
2660        STA $D8      ONERR ACTIVE
2670        ASL          $00
2680        STA $76      AS([]) RUNNING
2690        STA $33       (REALLY)
2700        RTS
2710  OFF.ERROR
2720        LDX SAVE.AAB6
2730        BEQ CLEAR.ERROR   ZERO->NEVER SET
2740        INX
2750        STX $AAB6         48K ONLY
2760  CLEAR.ERROR
2770        LDA #0            CLEAR FLAGS
2780        STA SAVE.AAB6
2790        STA $D8           CLEAR ONERR FLAG
2800        RTS
2810  *---------------------------------
2820  SAVE.AAB6 .HS 00    FLAG
2830  *---------------------------------
2840  *
2850  * LOOK FOR FILE ON VARIOUS DRIVES
2860  *
2870  *  RETURNS CARRY CLEAR IF FOUND
2880  *   AND SET IF NOT. USES RENAME
2890  *   FILE,FILE TO SEE IF FILE EXISTS
2900  *
2910  *---------------------------------
2920  LOOK.FOR.FILE
2930        TSX          SAVE STACK
2940        STX LOOK.STACK
2950        LDA #LOOK.ERR
2960        LDY /LOOK.ERR
2970        JSR ON.ERROR
2980        LDA #0       TABLE OFFSET
2990        STA LOOK.CNT
3000  *---------------------------------
3010  LOOK.LOOP
3020        LDX LOOK.CNT
3030        CPX LOOK.MAX (# OF TRYS)
3040        BCS .99      FAIL EXIT
3050  *->->->->->->->->->->->->->->->->->
3060  * CHECK FOR DISK CARD IN SLOT
3070  *  SO THINGS WON'T HANG. FIRST,
3080  *  LOAD THE ACC WITH THE SLOT
3090  *  THEN ...
3100  *->->->->->->->->->->->->->->->->->
```

```
3110
3120        AND #$07     SLOT #
3130        ORA #$C0
3140        STA TMP1+1
3150        LDA #0
3160        STA TMP1     TMP1=CS00
3170  * TMP1 IS SLOT ADDRESS
3180  * CHECK BYTES 7,5,3,1 FOR MATCH
3190  *  AS AUTO MONITOR DOES
3200        LDY #$07     SAME AS MONITOR ($FABA AUTO)
3210  .10   LDA (TMP1),Y FETCH SLOT BYTE
3220        CMP DISKID-1,Y  IS IT DISK?
3230        BNE LOOK.ERR NOPE...
3240        DEY          DOWN TWO
3250        DEY
3260        BPL .10      AND LOOP
3270  * THERE IS A DISK CARD THERE
3280  *---------------------------------
3290  *->->->->->->->->->->->->->->->->->
3300  * PRINT OUT CTRL-D THEN "RENAME
3310  *  FILE,FILE,SX,DX" FILLING IN
3320  *  X ACCORDING TO THE VALUE OF
3330  *  LOOK.CNT
3340  *->->->->->->->->->->->->->->->->->
3350        CLC          FOUND IT
3360  .99   LDX LOOK.STACK
3370        TXS          RESTORE STACK
3380        RTS
3390  *---------------------------------
3400  * COME HERE IF DOS COMMAND FAILS
3410  *---------------------------------
3420  LOOK.ERR
3430        INC LOOK.CNT
3440        JMP LOOK.LOOP
3450  *---------------------------------
3460  LOOK.MAX   .DA #6
3470  LOOK.CNT   .BS 1
3480  LOOK.STACK .BS 1
3490  *---------------------------------
3500  DISKID     .HS 20FF00FF03FF3C MATCHES DISK CARD+1 TO 7
3510  *---------------------------------
3520  *
3530  * TABLE OF ERR/RESET ADDRESSES
3540  *
3550  *---------------------------------
3560  MY.TABLE .DA MY.RESET
3570           .DA MY.RESET
3580           .DA MY.ERROR
3590           .DA $E000
3600           .DA MY.RESET
3610           .DA MY.RESET
3620        .EN
```

```
=====================================================================
DOCUMENT :AAL-8202:DOS3.3:S.EpsonROMChng.txt
=====================================================================

 1000  *--------------------------------
 1010  *      CHANGES TO EPSON CONTROLLER 2708 ROM
 1020  *--------------------------------
 1030  *---AS IT NOW IS-----------------
 1040         .OR $C111
 1050         .TA $811
 1060         PLA
 1070         TAY
 1080         DEX
 1090         TXS
 1100         PLA
 1110         PLP
 1120         TAX
 1130         BCC $C152
 1140         LDA $5B8,X
 1150         BPL $C138
 1160         TYA
 1170         AND #$7F     STRIP OFF SIGN
 1180         EOR #$30
 1190  *---AS IT NEEDS TO BE------------
 1200         .OR $C111
 1210         .TA $811
 1220         PLA
 1230         AND #$7F     STRIP OFF SIGN BIT
 1240         TAY
 1250         DEX
 1260         TXS
 1270         PLA
 1280         PLP
 1290         TAX
 1300         BCC $C152
 1310         LDA $5B8,X
 1320         BPL $C138
 1330         TYA
 1340         EOR #$30
 1350         .LIF
```

```
==============================================================================
DOCUMENT :AAL-8202:DOS3.3:S.FIFOPrntHndlr.txt
==============================================================================

 1000  *---------------------------------
 1010  *          PRINTER HANDLER
 1020  * USED SO THAT PROGRAM DOESN'T HANG
 1030  *       WHEN PRINTER IS BUSY
 1040  *
 1050  *            JIM KASSEL
 1060  *        1161 GOODRICH AVE.
 1070  *        ST. PAUL, MN 55105
 1080  *---------------------------------
 1090  PRINT.SLOT.SHIFTED  .EQ $10
 1100  *                   PRINTER SLOT # SHIFTED LEFT BY 4
 1110  PBII   .EQ $CE      PRINT BUFF INPUT INDEX
 1120  PBOI   .EQ $CF      PRINT BUFF OUTPUT INDEX
 1130  PBCC   .EQ $1F      PRINT BUFF CHAR COUNT
 1140  PBUFF  .EQ $9400    PRINT BUFF BASE ADDRESS
 1150  CR     .EQ $D       CARRIAGE RETURN WITH MSB CLR
 1160  LF     .EQ $A       LINE FEED WITH MSB CLR
 1170  *---------------------------------
 1180  START  .EQ $800
 1190         .OR START
 1200  *---------------------------------
 1210  * PRINT BUFF INPUT SUBROUTINE
 1220  *---------------------------------
 1230  PRINT.FIFO.INPUT
 1240         PHA
 1250         AND #$7F     CLEAR BIT 7
 1260
 1270  .1     LDY PBII
 1280         STA PBUFF,Y  STORE CHAR IN PRINT BUFF
 1290         INC PBII     INCREMENT INPUT INDEX
 1300         INC PBCC     INCREMENT CHAR COUNT
 1310
 1320         CMP #CR      CARRIAGE RETURN?
 1330         BNE .2       NO
 1340         LDA #LF      YES
 1350         BNE .1       SEND <LF>
 1360
 1370  .2     PLA          RESTORE CHAR
 1380         RTS
 1390  *---------------------------------
 1400  * PRINTER OUTPUT SUBROUTINE
 1410  *---------------------------------
 1420  PRINT.FIFO.OUTPUT.1
 1430         LDA PBCC     PRINT BUFF EMPTY?
 1440         BEQ .1       YES
 1450
 1460         LDY PBOI     NO
 1470         LDA PBUFF,Y  GET PRINT CHAR
 1480         JSR PRINT.FIFO.OUTPUT.2
```

```
1490  *                     HANDLER OF SPECIFIC INTERFACE
1500         BCS .1       DON'T UPDATE IF PRINTER WAS BUSY
1510
1520         INC PBOI     ELSE, INCREMENT OUTPUT INDEX
1530         DEC PBCC     AND DECREMENT CHAR COUNT
1540
1550  .1     RTS
1560  *---------------------------------
1570  *   HANDLER FOR THE GRAPPLER (+)
1580  *         INTERFACE CARD
1590  *      AND MX-80 PRINTER(++)
1600
1610  * PRINT CHAR MUST BE IN THE A-REG
1620  *   CARRY SET IF CHAR NOT SENT
1630  *   CARRY CLEARED IF CHAR SENT
1640
1650  PSTAT  .EQ $C081    PRINTER STATUS REG
1660  PREG   .EQ $C081    PRINTER DATA REG
1670  PSTRBL .EQ $C082    PRINTER STROBE LOW
1680  PSTRBH .EQ $C084    PRINTER STROBE HIGH
1690  *---------------------------------
1700  PRINT.FIFO.OUTPUT.2
1710         TAX          SAVE PRINT CHAR
1720         LDY #PRINT.SLOT.SHIFTED
1730         LDA PSTAT,Y  GET PRINTER STATUS
1740         AND #$A      MASK
1750         EOR #$2      PRINTER SELECTED AND NOT BUSY?
1760         BNE .1       NO, EXIT
1770
1780         TXA          YES, RESTORE PRINT CHAR
1790         STA PREG,Y   LOAD PRINTER OUTPUT REG
1800         STA PSTRBL,Y SET STROBE
1810         STA PSTRBH,Y CLR STROBE
1820         CLC          CLEAR CARRY
1830         BCC .2       EXIT
1850  .1     SEC          SET CARRY
1860
1870  .2     RTS
1880  *---------------------------------
1890  END
1900  SIZE   .EQ END-START
1910  *---------------------------------
1920  * NOTE:
1930  * (+) : TRADEMARK OF ORANGE MICRO, INC.
1940  * (++): TRADEMARK OF EPSON AMERICA, INC.
1950  *---------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8202:DOS3.3:S.Putney.Primes.txt
=============================================================================

 1000         .OR $8000    SAFELY OUT OF WAY
 1010         .TF B.PUTNEY'S PRIMES
 1020  *--------------------------------
 1030  BASE   .EQ $2000    BASE OF PRIME ARRAY
 1040  BEEP   .EQ $FF3A    BEEP THE SPEAKER
 1050  *--------------------------------
 1060  *      MAIN CALLING ROUTINE
 1070  *
 1080  MAIN   LDA #100     DO 100 TIMES SO WE CAN MEASURE
 1090         STA COUNT    THE TIME IT TAKES
 1100         JSR BEEP     ANNOUNCE START
 1110  .1     JSR ZERO     CLEAR ARRAY
 1120         LDA #$03
 1130         STA START    SET STARTING VALUE
 1140         JSR PRIME
 1150         DEC COUNT    CHECK COUNT
 1160         BNE .1       DONE ?
 1170         JSR BEEP     SAY WE'RE DONE
 1180         RTS
 1190  *--------------------------------
 1200  *      ROUTINE TO ZERO MEMORY
 1210  *      FROM $2000 TO $6000
 1220  *
 1230  ZERO   LDA #BASE+1  START AT $2001
 1240         STA .1+1     MODIFY OUR STORE
 1250         LDA /BASE+1
 1260         STA .1+2
 1270         LDA #$00     GET A ZERO
 1280         TAX          SET INDEX
 1290         LDY #$40     NUMBER OF PAGES
 1300  .1     STA $FFFF,X  MODIFIED AS WE GO
 1310         INX          EVERY ODD LOCATION
 1320         INX
 1330         BNE .1       NOT DONE
 1340         INC .1+2     NEXT PAGE
 1350         DEY
 1360         BNE .1       NOT YET
 1370         RTS
 1380  *--------------------------------
 1390  *      PRIME ROUTINE
 1400  *      SETS ARRAY STARTING AT BASE
 1410  *      TO $FF IF NUMBER IS NOT PRIME
 1420  *      CHECKS ONLY ODD NUMBERS > 3
 1430  *      INC = INCREMENT OF KNOCKOUT
 1440  *      N = KNOCKOUT VARIABLE
 1450  *
 1460  PRIME  LDA START
 1470         ASL          INC = START * 2
 1480         STA INC
```

```
1490        JSR SQUARE   SET N = N * N
1500        CLC          ADD BASE TO N
1510        LDA N+1
1520        ADC #BASE
1530        TAX          KEEP LOW ORDER PART IN X
1540        LDA #0       N+1 TO ZERO
1550        STA N+1
1560        LDA N+2
1570        ADC /BASE
1580        STA N+2
1590        TAY
1600 LOOP   LDA #$FF     FLAG AS NOT PRIME
1610 N      STA $FFFF,X  REMEMBER THAT N IS REALLY AT N+1
1620        CLC          N = N + INC
1630        TXA          N=N+INC
1640        ADC INC
1650        TAX
1660        BCC LOOP     DONT'T BOTHER TO ADD, NO CARRY
1670        INY          INC HIGH ORDER
1680        STY N+2
1690        CPY /BASE+$4000  IF IS GREATER THAN $6000
1700        BCC LOOP     NO, REPEAT
1710        LDX START    GET OUR NEXT KNOCKOUT
1720 .1     INX
1730        INX          START = START + 2
1740        BMI .2       WE'RE DONE IF X>$7F
1750        LDA BASE,X   GET A POSSIBLE PRIME
1760        BNE .1       THIS ONE HAS BEEN KNOCKED OUT
1770        STX START
1780        BEQ PRIME    ...ALWAYS
1790 .2     RTS
1800 *--------------------------------
1810 *      SQUARE ROUTINE
1820 *      TAKES SQUARE OF NUMBER
1830 *      IN START (ONE BYTE) AND
1840 *      PUTS RESULT IN N+1 (LOW)
1850 *      AND N+2 (HIGH)
1860 *
1870 SQUARE LDA #$00
1880        STA N+1      CLEAR N
1890        STA N+2
1900        STA MULT+1   AND MULTIPLIER HIGH
1910        LDA START
1920        STA MULT     MULT LOW = START
1930        STA SHCNT    SHIFT COUNTER
1940        LDX #$08     EIGHT SHIFTS
1950 .1     ROR SHCNT    GET LS BIT IN CARRY
1960        BCC .2       DON'T ADD THIS TIME
1970        CLC          N = N + MULT
1980        LDA N+1
1990        ADC MULT
2000        STA N+1
2010        LDA N+2
2020        ADC MULT+1
```

```
2030          STA N+2
2040  .2      CLC          SHIFT MULT (BOTH BYTES)
2050          ROL MULT
2060          ROL MULT+1
2070          DEX
2080          BNE .1       MORE BITS ?
2090          RTS
2100  START   .DA #*-*     STARTING KNOCKOUT
2110  INC     .DA #*-*     INCREMENT FOR KNOCKOUT
2120  COUNT   .DA #*-*     COUNT FOR 100 TIMES LOOP
2130  MULT    .DA *-*      MULTIPIER
2140  SHCNT   .DA #*-*     SHIFT COUNT MULTIPLIER
```

```
=============================================================================
DOCUMENT :AAL-8203:Articles:Code.Alwys.Skip.txt
=============================================================================
```

Tricky Code that Always Skips..............Bob Sander-Cederlof

All microprocessors have an instruction which does nothing, usually called "NOP". The 6502 is no exception.

In spite of appearances, an instruction which does nothing can be quite useful. However, this article is about another kind of instruction, which does ALMOST nothing.

Some microprocessors have this kind, which do nothing except skip over one or more bytes. That is, they act like a very short forward jump. The advantage over using an actual jump or branch instruction is in memory saved. Relative branches on the 6502 take two bytes of memory; jumps take three. A skip-one or skip-two instruction would take only one byte, IF the 6502 had such.

IF? Well, you certainly do not see an instruction like this among the 56 in any of the books, do you?

However, if you disassemble things like DOS, Applesoft ROMs, and printer interface ROMs, you will find tricky ways to skip with only one byte. For example, in many Apple printer interfaces, the first three bytes look like this:

```
C100-  18        CLC
C101-  B0 38     BCS $C13B
```

Now isn't that silly: to clear carry, and then to use BCS to branch if it is not clear!? No, the BCS is just being used to skip over the $38 stored in $C102. If you enter the code at $C102, that $38 is a SEC instruction. Thus, depending on whether you entered at $C100 or $C102, carry is clear or set respectively. The BCS opcode byte is being used as a skip-one opcode.

Another kind of skip is found in various places inside your Apple. You might find the BIT instruction used this way. In fact, it seems to me that I run across BIT being used as a skip-one or skip-two instruction more often than I see it used to test bits! Here is an example from Applesoft ROMs:

```
E196-  A2 78     LDX #$6B      "BAD SUBSCRIPT" MSG
E198-  2C A2 35  BIT $35A2     TRICK: BIT SKIPS OVER 2
E19B-  4C 12 D4  JMP $D412     PRINT ERROR MESSAGE
```

The code should really look like this:

```
E196-  A2 78     LDX #$6B      "BAD SUBSCRIPT" MSG
E198-  2C        .HS 2C        SKIP NEXT TWO BYTES
E199-  A2 35     LDX #$35      "ILLEGAL QUANTITY" MSG
```

```
E19B-  4C 12 D4  JMP $D412     PRINT ERROR MESSAGE
```

You have to be a little careful about what you skip over. The BIT instruction is actually executed, and so status flags Z, N, and V are possibly changed. Also, the two bytes skipped over represent a memory address to the BIT opcode; that memory location will be accessed. No problem, unless it just happens to be an address in the range of the I/O addresses (from $C000 to $CFFF). If it does, something strange might occur, like turning on a disk drive....

If you remember my article about the "So-Called Unused Opcodes", from about a year ago, there are some REAL skip-one and skip-two instructions. They do not modify any status bits, and they do not reference any memory addresses. I would recommend using ".HS 3C" rather than ".HS 2C" for this reason. "3C" is not a defined or supported opcode, but it apparently is built-in to all existing 6502's. (No guarantee here...test your own before you make a big commitment.)

If you want to skip only one byte, you can use the other BIT form ($24); it works on a zero page address, which will not bother any I/O addresses. If you don't want to modify any status bits, try ".HS 34".

==============================================================================
DOCUMENT :AAL-8203:Articles:Correx.2.FIFO.txt
==============================================================================

Correction to Kassel's FIFO Handler..............Bill Morgan

Ever the experimenter, I started playing with Jim Kassel's FIFO Buffered Printer Handler as soon as I read about it. I learned a lot, but maybe I can spare you some difficulty with the following information.

1. Be aware that the three indices PBII, PBOI, and PBCC must be all cleared to zero before the first time you activate the handler.

2. Line 1720 was printed in AAL with a missing character. Change from

```
     1720       LDY PRINT.SLOT.SHIFTED

to   1720       LDY #PRINT.SLOT.SHIFTED
```

```
=====================================================================
DOCUMENT :AAL-8203:Articles:EPROM.Blstr.Def.txt
=====================================================================
```

EPROM Blaster Defined......................Bob Sander-Cederlof

Several readers have asked what an EPROM blaster is. This is a device, more commonly called an EPROM programmer or writer or burner, which writes data into an EPROM. The EPSON interface has an EPROM device on it, called a "2708", which can hold 1024 bytes of data or program. (Only the first 256 bytes are actually used by EPSON.) A company called Apparat advertises a card for the Apple II which will write (burn, program, blast,...) stuff into a 2708. They call their board the "Blaster".

Mountain Computer makes the ROMWRITER board for the Apple. This board can only burn single-voltage 2716 EPROMs, the Apparat board can burn 2708s, 2716s, and 2732s, whether single or multiple voltages. And ROMWRITER costs almost twice as much.

Maybe you are asking, "What on earth is an EPROM, anyway?" EPROM stands for "Erasable Programmable Read Only Memory". The "memory" part is easy: each EPROM can hold a large number of bytes of data or program. A 2708 holds 1024 bytes, 2716 holds 2048 bytes, and a 2732 holds 4096 bytes.

"Read Only" means that once the bytes are recorded, they cannot be changed. They are permanent, even if power is removed. "Programmable" means that you and I can, with a burner or blaster", record the bytes; the chip comes un-recorded from the factory. Non-programmable ROMs are recorded during manufacturing.

"Erasable" means that you can erase what you have recorded and re-use the chip. An ultraviolet lamp is used to erase the contents; I bought a $75 EPROM Eraser from Logical Devices in Florida for the job. Maintaining the level of confusion, still other letters can be added to the acronym: EEPROMs are "Electrically" erasable; EAROMs are too (I don't know the difference between the two, if any).

```
==============================================================================
DOCUMENT :AAL-8203:Articles:Front.Page.txt
==============================================================================
```

$1.50

Volume 2 -- Issue 6                              March, 1982

In This Issue...



Reading Two Paddles at the Same Time.......Bob Sander-Cederlof

You may have discovered by now that if you try to read both game paddles from BASIC, there is some interaction at certain ranges.  The problem is that there is only one trigger for both (really, all four) analog ports.  If one of them times out long enough before the other one, you will read the tail end of the count on the second timer.

I wrote a little subroutine (see back page) which reads both paddles at once, eliminating all interaction.  It also stretches the range, meaning you need a higher resistance than the standard paddles to get a full 0-255 counting range.  Programs which use both paddles will run faster using this subroutine, because you get two readings in the time of one.

Circulation and Advertising Rates

Now that circulation is over 1000 copies per month, it seems appropriate to charge more per page for advertising than I did when there were only 100 to 200 subscribers.  The new rate, effective immediately, is $30 for a full page, $15 for a half page.

================================================================================
DOCUMENT :AAL-8203:Articles:More.Epson.Intf.txt
================================================================================

More About the EPSON Interface................Peter Bartlett

Whoops! I left out something in my instructions for modifying the EPSON interface card!

The software driver on the interface card is $100 bytes long, and resides in the first 256 bytes of the 1024-byte EPROM. However, the folks at EPSON got a couple of the address lines mixed up. Burning a new EPROM is not as straightforward as it should be.

The problem is that chunks of the program are shuffled. To understand, consider the $100 bytes to be divided into 4 parts of $40 bytes each. Part 0 is $0 to $3F, part 1 is $40 to $7F, part 2 is $80 to $BF, and part 3 is $C0 to $FF. When blasting the EPROM, the sequence of these parts must be changed. Instead of 0,1,2,3, the sequence must be 1,0,3,2.

When you list the contents of the EPROM while it is in the EPSON card, the contents appear normal. But if you remove the EPROM from the card and read it with another device, it will be in its juggled format.

Another point worth emphasizing is that this fix does not allow characters with the high-bit set to pass through the normal software driver. This driver is only compatible with the Apple's normal ASCII output. However, both Applesoft and machine language programmers can send 8-bit characters by bypassing the card as described last month in my article.

================================================================
DOCUMENT :AAL-8203:Articles:New.SCAsm.Ad.txt
================================================================

S-C Software Corporation is pleased to introduce the S-C Macro Assembler, the latest version of our most popular product. The S-C Assembler II Version 4.0 already has the reputation of being the easiest editor/assembler to learn, to remember, and to use...now the S-C Macro Assembler provides a new level of power and performance for the beginner and professional programmer alike.

29 Commands, including a convenient EDIT command with 15 subcommands. COPY and REPLACE commands further simplify entry and modification of even the most complex programs.

20 Assembler Directives (Pseudo-Ops) provide all features necessary for professional software development, including conditional assembly and macro generation.

Operates in any Apple II or Apple II Plus with at least 32K RAM and one disk drive. Any additional memory or disk drives will be used as required. A Language Card version is also included.

A memory size of 48K allows source programs of over 24,000 bytes to be handled entirely within RAM. The Language Card version allows source programs of over 32,000 bytes. Much larger programs can be edited and assembled using the "INCLUDE" and "TARGET FILE" capabilities, up to the limit of on-line disk storage.

Programs can be edited, assembled, and tested entirely within the framework of the S-C Macro Assembler. The editor and assembler are co-resident, allowing rapid cycles of modification, re-assembly, and check-out. All DOS and Apple Monitor commands are active as well, providing a familiar interface to the standard Apple features.

Uses its own high-speed technique to store source files, but also can read or write standard TEXT files. You can EXEC in files from another assembler, use some other text editor to prepare files, keep a library of routines on disk to EXEC into any program, or use S-C Macro Assembler to prepare EXEC files for any purpose.

Price is only $80! Includes diskette with Macro Assembler and sample programs, a 100-page Reference Manual, and a Programmer Reference Card. (Registered Owners of S-C Assembler II Version 4.0 may purchase the upgrade package for only $27.50)

Already well-known for excellent support, S-C Software Corporation pledges to continue development of new features, and to help owners gain the maximum benefit from the S-C Macro Assembler. In addition to telephone consultation for registered owners, a monthly newsletter is available by subscription (currently $15/year). The "Apple Assembly Line" covers items of interest to assembly language programmers at all levels, and has helped many to advance their programming skills.

```
Commands

       Source:  NEW, LOAD, SAVE,
                TEXT, HIDE, MERGE

      Editing:  LIST, FIND, EDIT,
                DELETE, REPLACE,
                COPY, RENUMBER

 List Control:  FAST, SLOW, PRT, "

       Object:  ASM, MGO, VAL,
                SYMBOLS

Miscellaneous:  AUTO, MANUAL,
                INCREMENT, MEMORY,
                MNTR, RST, USR

All Apple Monitor Commands

All Apple DOS Commands

Assembler Directives

.OR   Origin
.TA   Target Address
.TF   Target File
.IN   Include File
.EN   End of Program
.EQ   Equate
.DA   1- or 2-byte Data
.HS   Hex String
.AS   ASCII String
.AT   ASCII Terminated

.BS   Block Storage
.TI   Title
.LIST Listing Options
.PG   Page Eject
.DO    Conditional
.ELSE   Assembly
.FIN
.MA   Macro Definition
.EM   End of Macro
.US   User Directive

Apple is a trademark of Apple Computer

We take Master Card and Visa
```

==============================================================================
DOCUMENT :AAL-8203:Articles:OtherEpsonMan.txt
==============================================================================

The Other EPSON Manual -- A Review.........Bob Sander-Cederlof

If you have an EPSON MX-80 printer tied to your Apple (who doesn't), you probably share the frustration of trying to learn how to use it with a manual aimed at Radio Shack TRS-80 owners. Bill Parker decided to do something about it.

Bill studied, analyzed, experimented, and perspired; then he wrote the key facts down in Apple-oriented English. With description and sample program listings he shows you how to:

1. Use all 12 print modes (emphasized, double width, etc.).
2. Underline.
3. Use subscripts and superscripts.
4. Set half-spacing, double-spacing, etc.
5. Do formfeeds, vertical tabs, etc.
6. Use horizontal tabs.
7. Use the printer commands inside a word processor.
8. Do some special tricks through the parallel interface card (true underlining, single-word emphasis, etc.).

The book(let) is 8-1/2 by 11, 17 pages, bound with a plastic comb. Not elegant, but sufficient; and anyway, it is the information he is selling. The price is $4.98, postpaid, from Bill Parker, Cut The Bull Software, P. O. Box 82761, San Diego, CA 92138.

==============================================================================
DOCUMENT :AAL-8203:Articles:Rvw.6502.Subs.txt
==============================================================================

Leventhal's 6502 Subroutines

6502 Assembly Language Subroutines, by Lance Leventhal and Winthrop Saville, is a book all of you will want.  Specs:  550 pages, 7-1/2 by 9-1/4 inches, paperback, $12.99 from Osborne/McGraw-Hill.  I'll send you a copy for $12 plus $2 shipping (it weighs two pounds!). Naturally, shipping will be more if you live outside the USA.

Quoting from the back cover:

"If you want to use a specific assembly language routine, learn assembly language quickly, or improve your programming skills, 6502 Assembly Language Programming is for you.  It provides code for more than 40 common 6502 subroutines, including code conversion, array manipulation, arithmetic, bit manipulation, string processing, input/output, and interrupts.  It describes general 6502 programming methods (including a quick summary for experienced programmers), and tells how to add instructions and addressing modes [using several instructions in sequence, subroutines, or macros].  It even discusses common 6502 assembly language programming errors."

All of the subroutines are thoroughly documented, making it easy to understand how they work, and how to use them.  The subroutines are useful in the Apple with no changes, other than those required to interface to your own programs.  Some of the subroutines even reference the Apple monitor ROMs!

The first five copies I bought were gone within three hours of their arrival, so I ordered 20 more.  Want one?

==============================================================================
DOCUMENT :AAL-8203:Articles:Rvw.AmperMagic.txt
==============================================================================

A Review of AMPER-MAGIC....................Bob Sander-Cederlof

AMPER-MAGIC is a utility program which makes it easy to add machine language subroutines to Applesoft programs and thereby extend the capabilities of Applesoft BASIC. It was written by Bob Nacon, one of our subscribers from New Jersey. For $75, you get a 51-page reference manual; an administrative program; and a collection of 23 subroutines, to be added to your programs.

Why We Need It

Here are some common problems that we have all had in developing machine language routines for Applesoft:

* Where do you put it? You don't want to clobber Applesoft or DOS, and you don't want either of them to clobber your routines.
* How do you get to it? CALL? Ampersand? USR?
* What do you do when you want to add a second routine?
* How do you pass data to the subroutine, and get answers back?

Most of the time we have put all of our routines at location $300-$3CF because that is a free area. It works great until you need the same space for a second or third routine. We also have been using the POKE technique of placing the machine language routine at location $300 and then calling it with CALL 768 or using the Ampersand command. This is fine for 1 or 2 routines, but you lose the full advantage of the speed of these routines waiting for them to be POKEd into memory. AMPER-MAGIC solves all of the above problems nicely.

AMPER-MAGIC hides your subroutines "underneath" your Applesoft program so that they are loaded automatically along with the Applesoft program. AMPER-MAGIC can handle 255 different subroutines of varying lengths. You can use as much space as necessary, up to the limit of memory. That solves the problem finding space for your routines.

The Ampersand ("&") command of Applesoft followed by the name of your routine is used to gain access to your subroutines. More about subroutine names later. By pointing the Ampersand vector at $3F5-$3F7 to the proper place, AMPER-MAGIC decodes the name of the routine desired and then transfers control to it. That solves the problem of linkage to more than one subroutine, and in a way that is human readable!

There is one limitation to the subroutines which can be used within AMPER-MAGIC: they must be fully relocatable subroutines. Without any change or reassembly they must be able to work at a new address.

Why? Because they are located at the end of your Applesoft program. As you edit your program, even just a little, the subroutines will probably move to a different address.

A fully relocatable subroutine is one which does not make any direct references to any address WITHIN the subroutine. There can not be any JMP, JSR, LDA, STA, etc. to an address within the subroutine. Only relative addressing branch commands may be used within subroutines.

Many of the subroutines published within AAL this past year were not fully relocatable but they could be made so easily. Maybe I will spend some time in a future issue discussing techniques on how to make subroutines fully relocatable. Roger Wagner, in his "Assembly Lines" column in Softalk Magazine, explained many of the motives and methods involved.

AMPER-MAGIC lets you select any name you wish for your subroutines, even for the subroutines in the AMPER-MAGIC library.

Names may be up to 4 bytes long. That is bytes, not necessarily characters. Applesoft tokenizes every command name or function name into a one byte token. Thus you can call your subroutines PRINT, INPUT, GET, etc. which only take up one byte each. A name like CLEAREOL is a legal AMPER-MAGIC name and only takes up 4 bytes (one for CLEAR, three for EOL). This allows you to name your own subroutines very descriptively for future reference.

To call a subroutine from within your program you simply use the & (Ampersand) followed by your subroutine name, followed by a "," and then your variables. The comma is not needed if there are no variables. For example:   &GOTO,A*5  or &CLEAREOL:.

The AMPER-MAGIC administrative program is a smooth operating menu driven program which prompts you all along the way. Here is how you use it:

1. Load your Applesoft program.
2. Put the AMPER-MAGIC diskette in a drive and type EXEC AMPER-MAGIC. (Specify slot and drive if not the same as the last accessed one.)
3. Fill in information after the prompts, as required.

By following the menu and the well-written documentation, you can add, change, delete, and rename any subroutine in your program. You may add or delete any number of subroutines in one session.

You can load a subroutine directly from the keyboard in either decimal or hex. Thus many of the routines published in AAL can just be typed directly into AMPER-MAGIC.

If you have subroutines already assembled on disk, you simply tell AMPER-MAGIC the file names watch it work. AMPER-MAGIC makes room in the subroutine table at the end of your program, and loads the subroutine into your program. Really neat! Everything is handled automatically except for the subroutine name, which you must supply.

There isn't enough room here to describe all the other functions available, but suffice to say that AMPER-MAGIC gives you all the administrative functions you need to selectively add or delete any subroutines from your program easily and quickly.

Once you have finished with AMPER-MAGIC you simply EXIT via the menu. AMPER-MAGIC returns all your program pointers to their previous state, and clears itself out. Your program has now been modified and you can run it to check out the new subroutine. If you need to make further changes, just EXEC AMPER-MAGIC again.

The AMPER-MAGIC program alone is probably enough to justify its purchase, but you also get 23 ready to use subroutines. Some of these were originally published right here in AAL. Bob Nacon modified them wherever necessary to make them fully relocatable.

Here is a list of some of the subroutines on the disk:

```
&FIND,v$,v$,v      Find a substring in a string.
&DARY,v            Delete an array.
&GET,v,v           PEEK a two-byte value.
&GOSUB,v           GOSUB to a variable line.
&GOTO,v            GOTO to a variable line.
&INPUT,v$          Input a line containing even commas,
                   quotation marks, or colons.
```

The ones listed above only give you the flavor. Remember, there are 23!

One of the best features of all of these subroutines is that all information is passed to and from the subroutines via variables, just like regular commands. No peeking or poking to set up parameters. This is a very professional touch, and makes the subroutines truly useful.

Each subroutine is described in detail, with all the information and examples you need to use them effectively.

As you can probably tell, I like this program. It provides all of us an easy way to add all those neat routines we have been working on, or wanting to work on, and never had a good way of accessing them.

AMPER-MAGIC is available from your local dealer or from AURORA Systems Inc., Madison, WI 53704.

==============================================================================
DOCUMENT :AAL-8203:Articles:Rvw.TimeII.Card.txt
==============================================================================

Using the Applied Engineering Time II......Bob Sander-Cederlof

You have probably noticed Dan Pote's ad in this and previous issues of AAL.  I finally got one of his clock-calendar cards, and learned how to program it.

A disk full of sample programs comes with the board, but none of them were exactly what I wanted.  I wanted a simple short program to read the time and date and display it on the screen; and I wanted some patches to DOS 3.3 which would append the date in MM/DD/YY format to any files SAVEd or BSAVEd.

The clock already had the correct time and date set when it arrived in the mail.  The onboard rechargeable battery keeps the circuit running even when you remove the card from your computer!  A couple of times I stopped the clock when I was working on my programs, so I just used one of the time-setting programs on the disk to correct the time.

How do you read the time and date?  There are 13 registers on the board.  Each register holds one digit of the time and date information.  To read a particular register, you store the register number into the clock input port, and then read the clock output port.

In order to avoid reading the time or date while it is being changed, you momentarily stop the clock before reading, and restart it when you are finished.  You don't want to keep the clock stopped for more than one second, or it will lose time.  After stopping the clock, you have to wait at least 150 microseconds before reading it.  If the clock was updating when you stopped it, the delay allows the update in progress to complete.

The following program reads the time and date and writes it on the bottom line of the screen.

<program here>

My Time II card is in slot 5.  It will work in any slot from 1 to 7. Change line 1040 if you use a different slot.  There are two addresses used to talk to the Time II card:  $C081+slot*16, and $C082+slot*16. For slot 5, these are $C0D1 and $C0D2.  Line 1070 loads "slot*16" into the X-register, so that loads and stores into the Time II registers will be directed to the proper slot.

Lines 1080,1090 stop the clock.  Storing any value at $C0D1 of the form xxx1xxxx will stop the clock.  If bit 4 is a zero the clock will be started again, as in lines 1270,1280.

Lines 1100-1260 read the date and time and store them on the screen. The reading is under the control of a format map, line 1340.  The format map contains three kinds of bytes:  00, meaning the end of the map; 2x, register addresses; and ASCII characters with the high bit set.  The Y-register indexes access to the map, and also the corresponding position on the screen line.

Lines 1110-1130 get the next map byte, and analyze it.  If it is 00, the time and date have been read; then lines 1270-1320 restart the clock and test if you want to keep reading or not.  If the byte is negative, then it is an ASCII character; Line 1240 stores the character on the screen line, and reading continues.  If neither zero nor negative, the byte is a register address.  Line 1140 selects the register by storing its address at $C0D2.

Lines 1150-1230 read the selected register.  If the register was the tens-digit of the hour, then the flag bits are removed.  These flag bits indicate whether you are using 12-hour or 24-hour format in the Time II, and AM/PM status.  I didn't care, so I just mask them out.  I also replaced a leading zero digit with blank here.  Line 1230 converts the digit to an ASCII character.

Lines 1290-1320 test whether you have pressed any key on the keyboard. If not, reading continues.  If you did, the storbe is cleared and the program terminates after printing a carriage return.

Here is a summary of the clock register addresses:

| | tens | units | |
|---|---|---|---|
| Seconds | 21 | 20 | |
| Minutes | 23 | 22 | |
| Hours | 25 | 24 | with 12/24 and AM/PM flags |
| Day of Week | | 26 | |
| Day of Month | 28 | 27 | |
| Month | 2A | 29 | |
| Year | 2C | 2B | |

The second program I wrote only reads the date.  The actual reading is very similar to the first program, but the purpose is different. Instead of displaying it on the screen, I store in in the last 8 positions of the primary file name buffer inside DOS 3.3.  The patches in lines 1040-1140 set up SAVE and BSAVE to call my program before opening the file.  My program then modifies the file name to include the current date as the last 8 characters.

I located the program inside a hole in DOS 3.3 ($B6B3-$B6FD).  If you are already using a modified DOS, this hole may already have some code in it, so be careful.  For example, the DOS on Applied Engineering's disk IS modified, and the modification uses this same space.

When you assemble this program, the four .TF directives write four short little binary files (B.1, B.2, B.3, and B.4).  I wrote a four line EXEC file to BLOAD these four binary files, installing the patches.

<program here>

The program saves the contents of the A-register at line 1220, and restores A at line 1390. Lines 1230,1240 stop the clock so we can read it. Lines 1250-1270 delay for about 150 microseconds in case the clock was updating when I stopped it.

Lines 1280-1360 read the date under control of a format map in line 1420, almost the same way the first program did. This time I used the known length of 8 bytes to terminate the loop, rather than a final 00 byte. Line 1340 stores inside the DOS primary file name buffer ($AA75-$AA92).

Lines 1370-1380 turn the clock back on. Line 1390 restores the A-register, and line 1400 continues with the normal DOS 3.3 code.

Before arriving at the above technique, I tried several others. I had one working which patched the DOS File Manager instead of the SAVE and BSAVE commands. This version appended the date to the name of any and all new files created. It worked exactly as it should, but it would have caused many problems with existing programs. Many Applesoft and Integer BASIC programs using TEXT files use an OPEN-DELETE-OPEN-WRITE sequence to make sure that a new file is used for output. If my patch to the file manager was installed, this sequence would not work correctly anymore. Therefore I elected to go the more direct route, only dating SAVE and BSAVE files.

If you want to use the date on TEXT file names, you could append it to the file name using normal string concatenation techniques.

I have not used any other of the clock/calendar cards available for the Apple, but I am convinced the Time II from Applied Engineering is a good one. (It may also be the least expensive.) The circuit card is professionally done; the components are highest quality; it works when you plug it in. There are other features, such as interrupt capability, which I have not yet explored. If you have any use for a clock/calendar, I recommend this one.

```
==============================================================================
DOCUMENT :AAL-8203:Articles:SCAsm.Ready.txt
==============================================================================
```

S-C Macro Assembler........................Bob Sander-Cederlof

The printer has delivered the manuals (five days early!), the bugs are exterminated, the UPS driver went back to the depot and got a bigger truck, and we are now shipping S-C Macro Assembler.

Here is a brief summary of the new features the S-C Macro Assembler has that S-C Assembler II Version 4.0 did not.  The highlights are of course macros, conditional assembly and the new commands EDIT, COPY and REPLACE.  But they are not all!

Commands

There are 10 new commands:

```
EDIT         Select a line, a range of lines, or a range
             of lines that contain a particular string.
             Edit the lines using some of the 15
             convenient sub-commands.

TEXT         Write source program to disk, as a TEXT
             file, with or without line numbers.

REPLACE      Global search and replace. Your search
             string can include wildcards; you can limit
             the search to a line, a range of lines,
             or search the entire program.  The search
             can be made sensitive or insensitive to
             upper/lower case distinctions.  And you can
             select Auto or Verify mode for replacement.

COPY         Copy one or more lines from one place to
             another in the source code.  Rearrange your
             code as you please!

AUTO         Generate automatic line numbers after
             every carriage return.  Allows ordinary
             TEXT files to be EXECed into S-C Macro
             Assembler!  You still can use the Version
             4.0 form of automatic line numbers.
             Now you have a choice!

MANUAL       Turn off automatic line numbering.

SYMBOLS      Print out the symbol table, in case you
             missed it the first time.

MNTR         Enter the system monitor (just like CALL
             -151 in BASIC).  Of course all the
```

```
               Monitor commands can be executed within
               S-C Macro Assembler, but if you really
               WANT to leave....

RST            Change the Autostart Monitor RESET vector
               to the specified address.

"              Send setup control strings to your printer.
```

There are also improvements in some of the older commands.

The spelling of commands is now checked. In older versions, only the first three characters were tested. The first three are still all that are necessary, but any additional letters you type must be correct. For example, LIS will list your program, and so will LIST. But, LISX will give a syntax error.

LIST and FIND now have the same syntax (in fact, they are processed by the same routine.) They may now specify either a line range, a search string, or both. The search string now requires a delimiter.

Line ranges in the LIST, FIND, COPY, EDIT, and DELETE commands may be written with a leading or trailing comma (as in Applesoft):

```
     LIST ,2500      List from beginning through 2500.
     LIST 2500,      List from 2500 through end.
```

The NEW command now restarts the automatic line numbering at 1000, rather than continuing from the last line number you entered.

The SLOW and FAST commands no longer use the Monitor output hooks at $36 and $37.

To leave the Macro Assembler, type FP or INT. You no longer have to also type PR#0.

After using the PR#slot command to run your printer, use PR#0 to turn it off. FAST won't do it anymore.

Directives

There are 7 new directives:

```
.MA and .EM    For macro definition.

.DO expr       Start conditional block.
.ELSE          Toggle condition flag.
.FIN           End conditional block.

.TI num,title  Title and number each page of the
               assembly listing.

.AT string     Like .AS, but the last character has the
               high-bit set opposite from the rest.
```

The .DA directive may now have a list of expressions.

The .EQ directive may now be used with local labels.

The .LIST directive has new options to control listing of macro expansions.

Source Entry

Control-O (Override) will allow any control character to be typed into a source line in the normal input mode or in edit mode. The control character will appear in inverse video.

The editor no longer double spaces after each line is entered.

The escape-L comment line produces one less dash, so that the line lists on the screen without a blank line after it.

Operand expressions can now include * and / as operators, as well as + and -. The relational operators (<, =, and >) may also be used.

The tab routine has been changed to include up to five tab stops. The stop values are kept in a user-modifiable list starting at $1010. These are the actual column numbers (not 3 less, as in version 4.0). You may use any values up to column 248.

The tab character (control-I, $89) is kept at $100F now, so you can change it if you like some other character better.

Any sequence of the same character repeated 4 or more times in the source code is replaced by a token $C0, the character code, and the repeat count. (multiple blanks are still replaced by a single byte between $80 and $BF.) This reduces both the memory requirements and disk file size for your source programs.

If you want to shrink your source file a little, and if you have been using the Escape-L to generate comment lines that have all those dashes in them, type "EDIT" and hold down the RETURN and REPEAT keys until the entire program has been scanned. Type MEM before you do it, and after it is finished; you will probably notice a significant saving!

A parameter at location $1017 allows the extra compression to be turned on or off. If the contents of $1017 is $04, compression is on. If it is $FF, compression is off. You can experiment with this parameter to see what effect it has on program size.

Reference Manual

The S-C Macro Assembler comes with an all-new, 100-page manual. (At last! All the information in one place!) The manual includes chapters on source program format, commands, directives, operand expressions, macros, 6502 programming, SWEET-16, and a tutorial on using the Macro Assembler.

Assembly

Older versions of the assembler terminated assembly after finding one error. The S-C Macro Assembler keeps going, but rings the bell and prints an error message, so you know about it. If any errors are found during pass one, assembly terminates before doing pass two. At the end of assembly, the number of errors found is printed.

Typing the RETURN key during assembly will abort the assembly (even if the listing has been turned off with .LIST OFF directive).

Believe it or not, the new version assembles slightly faster than version 4.0! I measured about a 10% improvement on a large program.

All previous versions had difficulty handling forward references to variables which turned out to be in page zero. (Described on page 22 of the old blue manual.) That problem has been solved, so with S-C Macro Assembler it does not matter where you put your page-zero definitions.

Memory Usage

All page zero variables used by the assembler have been concentrated, so $00 through $1F are completely free for the user.

The standard version of the S-C Macro Assembler now occupies $1000 through $31FF. The symbol table starts at $3200 and grows upward; the source code still starts at $9600 and grows downward.

Included on the disk with the Macro Assembler is a Language Card version and a short EXEC file to load the card. This version fills the 16K RAM card from $D000 through about $F300. The symbol table begins at $1000 rather than $3200. The EXEC file configures things so that the language card contents appear to DOS as the opposite language to the one on the mother board. For example, if Applesoft is on the mother board, you type INT to get into the S-C Macro Assembler.

There are no variables within the body of the assembler. The Language Card version could be burned into ROM and placed on a firmware card, if you so desire.

Ordering

You can order the S-C Macro Assembler by phone or mail. We accept cash, checks, money orders, Visa, Mastercard, or COD. The price of the Macro Assembler is $80.00. Registered owners of S-C Assembler II Version 4.0 may upgrade to the S-C Macro Assembler for only $27.50.

```
==============================================================================
DOCUMENT :AAL-8203:DOS3.3:Inst.DOS.Patch.txt
==============================================================================

BLOAD B.1
BLOAD B.2
BLOAD B.3
BLOAD B.4
```

```
==============================================================================
DOCUMENT :AAL-8203:DOS3.3:S.DATE.FILES.txt
==============================================================================

 1000  *SAVE S.DATE.FILES
 1010  *--------------------------------
 1020  *      PUT DATE ON ALL NEW FILES
 1030  *--------------------------------
 1040         .OR $A33F    IN BSAVE COMMAND
 1050         .TF B.1
 1060         JSR PATCH
 1070  *--------------------------------
 1080         .OR $A3A5    IN SAVE COMMAND
 1090         .TF B.2
 1100         JSR PATCH
 1110  *--------------------------------
 1120         .OR $A3BE    IN SAVE COMMAND
 1130         .TF B.3
 1140         JSR PATCH
 1150  *--------------------------------
 1160  SLOT   .EQ 5
 1170  CLOCK  .EQ SLOT*16+$C080
 1180  *--------------------------------
 1190         .OR $B6B3
 1200         .TF B.4
 1210  PATCH
 1220         PHA
 1230         LDA #$10     HOLD CLOCK
 1240         STA CLOCK+1
 1250         LDY #32      DELAY 150 MICROSECONDS
 1260  .1     DEY          WHILE HOLD TAKES EFFECT
 1270         BNE .1
 1280         LDY #7       MOVE 8 CHARS
 1290  .2     LDA MAP,Y    NEXT BYTE FROM MAP
 1300         BMI .3       COPY CHARACTER
 1310         STA CLOCK+2       SELECT REGISTER
 1320         LDA CLOCK+2       READ REGISTER
 1330         ORA #$B0     CONVERT TO ASCII
 1340  .3     STA $AA8B,Y  IN LAST 8 CHARS OF PRIMARY FNB
 1350         DEY
 1360         BPL .2       LOOP UNTIL ALL 8 CHARS MOVED
 1370         LDA #0       RELEASE CLOCK
 1380         STA CLOCK+1
 1390         PLA
 1400         JMP $A3D5    CONTINUE AFTER PATCH
 1410  *--------------------------------
 1420  MAP    .HS 2A29AF2827AF2C2B
 1430  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8203:DOS3.3:S.DISPLAY.TIME.txt
==============================================================================

 1000  *SAVE S.DISPLAY TIME
 1010  *---------------------------------
 1020  *      READ DATE FROM CLOCK II
 1030  *---------------------------------
 1040  SLOT   .EQ $50      SLOT# * 16
 1050  CLOCK  .EQ $C080
 1060  *---------------------------------
 1070  READ   LDX #SLOT
 1080         LDA #$10     HOLD CLOCK
 1090         STA CLOCK+1,X
 1100         LDY #0       BEGINNING OF MAP
 1110  .1     LDA MAP,Y    NEXT BYTE FROM MAP
 1120         BEQ .3       END OF MAP
 1130         BMI .2       COPY CHARACTER
 1140         STA CLOCK+2,X     SELECT REGISTER
 1150         CMP #$25     IS IT HOUR:TENS?
 1160         BNE .4       NO
 1170         LDA CLOCK+2,X     YES
 1180         AND #3       STRIP OFF FLAGS
 1190         BNE .5
 1200         LDA #$A0
 1210         BNE .2       ...ALWAYS
 1220  .4     LDA CLOCK+2,X     READ REGISTER
 1230  .5     ORA #$B0     CONVERT TO ASCII
 1240  .2     STA BUFFER,Y
 1250         INY
 1260         BNE .1       ...ALWAYS
 1270  .3     LDA #0       RELEASE CLOCK
 1280         STA CLOCK+1,X
 1290         LDA $C000    SEE IF KEY PRESSED
 1300         BPL READ     NO, KEEP READING
 1310         STA $C010    YES, CLEAR STROBE
 1320         JMP $FD8E    LINEFEED AND RETURN
 1330  *---------------------------------
 1340  MAP    .HS 2A29AF2827AF2C2BA0A02524BA2322BA212000
 1350  *---------------------------------
 1360  BUFFER .EQ $7D0
```

```
==============================================================================
DOCUMENT :AAL-8203:DOS3.3:S.PADDLES.txt
==============================================================================

 1000  *--------------------------------
 1010  *      READ BOTH GAME PADDLES AT THE SAME TIME
 1020  *--------------------------------
 1030  MON.CH .EQ $24
 1040  PDL0   .EQ $C064
 1050  PDL1   .EQ $C065
 1060  PDL.S  .EQ $C070
 1070  KEYBOARD .EQ $C000
 1080  *--------------------------------
 1090  TEST   JSR READ.BOTH.PADDLES
 1100         TYA           (Y) = PDL 1 SETTING
 1110         JSR $FDDA     PRINT IN HEX ON SCREEN
 1120         INC MON.CH    SPACE BETWEEN VALUES
 1130         TXA           (X) = PDL 0 SETTING
 1140         JSR $FDDA     PRINT IN HEX ON SCREEN
 1150         LDA #0        HTAB 1
 1160         STA MON.CH
 1170         LDA KEYBOARD SEE IF ANY KEY PRESSED
 1180         BPL TEST      NO KEYPRESS, KEEP READING PADDLES
 1190         STA KEYBOARD+16   CLEAR KEYBOARD STROBE
 1200         RTS           RETURN
 1210  *--------------------------------
 1220  READ.BOTH.PADDLES
 1230         LDX #0        PADDLE 0 COUNT
 1240         LDY #0        PADDLE 1 COUNT
 1250         LDA PDL.S     START THE PADDLE TIMERS
 1260  .1     LDA PDL0      CHECK PADDLE 0 TIMER
 1270         BPL .2        TIMED OUT
 1280         INX           COUNT PDL0
 1290         LDA PDL1      CHECK PADDLE 1
 1300         BPL .4        TIMED OUT
 1310         INY           COUNT PDL1
 1320         BNE .1        AGAIN
 1330         LDX #255      MAX TIME FOR BOTH PADDLES
 1340         BNE .3        ...ALWAYS
 1350  *---PADDLE 0 TIMED OUT, KEEP LOOKING AT PADDLE 1
 1360  .2     LDA PDL1      CHECK PADDLE 1
 1370         BPL .5        TIMED OUT
 1380         INY           COUNT PDL1
 1390         NOP           EQUALIZE TIMING
 1400         NOP
 1410         NOP
 1420         NOP
 1430         BNE .2
 1440  .3     LDY #255      MAX TIME FOR PDL1
 1450         BNE .5        ...ALWAYS
 1460  *---PADDLE 1 TIMED OUT, KEEP LOOKING AT PADDLE 0
 1470  .4     LDA PDL0      CHECK PADDLE 0
 1480         BPL .5        TIMED OUT
```

```
1490          INX          COUNT PDL0
1500          NOP          EQUALIZE TIMING
1510          NOP
1520          NOP
1530          NOP
1540          BNE .4       KEEP CHECKING
1550          LDX #255     MAX TIME FOR PDL0
1560   .5     RTS          RETURN TO CALLER
```

================================================================
DOCUMENT :AAL-8204:Articles:Add.AutoSave.txt
================================================================

Adding Auto-SAVE to S-C Macro Assembler........Greg H. Anders

[ Greg is a subscriber from Albuquerque, New Mexico. ]

One of the nice features of the new S-C Macro Assembler is the title directive (.TI). This directive causes a title and page number to be printed at the top of each page of an assembly listing. The title directive gave me the idea for the Automatic Save command program which follows.

I felt the need for an Auto Save command because of my own carelessness. After extensive editing of a rather lengthy program, I decided it was a good time to save the program before I proceeded. The file names I use are usually descriptive and forgettable, so to save a file, I list the Catalog, then use the cursor controls to copy the file name. After the file name appeared on the screen, I zipped the cursor next to the name I wanted to save the file under and, succumbing to temporary insanity, typed an "L". The word "LOAD" flashed on the screen and my mouth dropped open in disbelief. The only sounds that could be heard were the whirr of the disk drive and the screams of my new code byting the dust cover!

I decided to try to simplify the task of saving a program, giving myself less chance of making an error. From this came the Auto Save command. With this command, typing SAVE does not save your program on cassette. Instead, the SAVE command searches your source program for a title. If a title is found and it is a valid DOS name, the source program is automatically saved, using the title as the file name. In addition, if you end your title with a version number in the form N.N, Auto Save automatically increments the version number in the source program and saves the program using the new version number. The version number option does not erase your old file, which means your old file is a back-up. Be careful, though. A few saves and your disk is full of back-up files. You'll need to go back and delete a file or two every once in a while.

The version number goes up to a maximum of 9.9, after which it starts back at 0.0. If the version number option is not desired, don't put a number in the form N.N at the end of your title.

Leading and trailing blanks are ignored by Auto Save. If there is more than one consecutive blank in a title, the blanks are compressed to one. Thus, the title ".TI 56,TI       TLE" generates a SAVE to the file named "TI TLE". Also, any commas in your title are changed to dashes so as not to confuse DOS.

To use the Auto Save command, the vector address of the SAVE command must be changed. The address must be one less than the actual start of the Auto Save command. For example, if Auto Save is assembled at

$800, the address would be changed in the table inside the S-C Macro Assembler to $07FF.

For the version of the S-C Macro Assembler which loads at $1000, change the contents of address $1679 to $07 and $1678 to $FF. Shown as a monitor command, this would be:

```
:$1678:FF 07
```

For the Language Card version of the S-C Macro Assembler, change the content of address $D679 to $07 and $D678 to $FF. You have to write-enable the card first:

```
:$C083 C083 D678:FF 07
```

I like to keep Auto Save behind the Language Card version of the Macro Assembler. I put the program at $F320 and the changes are:

```
:$C083 C083 D678:1F F3
```

One thing you'll have to look out for. If you type an illegal DOS SAVE command such as "SAVE 1 4 THE ROAD", DOS ignores this command and the Auto Save goes into effect; the "1 4 THE ROAD" is ignored. Also note that the save is performed on the drive that is active. Since commas are changed to dashes, there is currently no way to specify which drive you want the save to be performed on. Perhaps you would like to try to implement this enhancement yourself.

After you've installed the Auto Save program, type in this program:

```
1000 *    A TEST OF AUTO SAVE
1010      .TI 54, TITLE TEST VER. 0.9
```

Then type SAVE, and CATALOG. See how the file was saved? List the file and notice the change in line 1010. Voila!

For those of you who haven't updated to the Macro Assembler yet, Auto Save can be implemented with S-C Version 4.0 by using the .US command for the title. The changes which are necessary are outlined below.

1. The following lines must be deleted: 1490-1540, 2090-2150, 2460-2470, 2560-2930.

2. The following lines must be added:

```
1210        .US S-C VER. 4.0 AUTO SAVE 1.0
1600        BNE .2       ...ALWAYS
1920 *   CHECK THE OP CODE FOR .US
2170        BCS TITLE
2480 .1     CMP #$80
3480 OPS    .AS /.US/
3510 NO.TTL .AS /*** NO TITLE ERRO/
3515        .AS  -/R/
3520        .AS /*** ILLEGAL TITLE FIRST CHARACTE/
```

```
3525       .AS  -/R/
```

3. Change the SAVE vector address. For an origin of $800, that would be

```
     :$1271:FF 07
```

4. To use the command, put the title you want to use for the file name like so:

```
     .US MY TITLE VER. 1.0
```

================================================================================
DOCUMENT :AAL-8204:Articles:Ashby.Shift.Mod.txt
================================================================================

Ashby's Easy Shift-Key Modifier...........Bob Sander-Cederlof

How many times have you read or heard about a way to modify your Apple so that the shift-key would function like a normal typewriter? It is a relatively safe and easy thing to do, but the directions can really be frightening.

Words like "solder", "wire", "take the bottom off your Apple", and so on.

If you have an Apple with the piggy-back board hanging down under your keyboard (Revision 7 or newer), take heart! There is a little device you can pick up for only $15 postpaid, called Ashby's Shift-Key Modifier, which hooks up the modification without any tools or trouble. And it only takes a minute or so! (In fact, only a few seconds if you have done it a few times like I have.)

The Modifier consists of a piece of wire fitted with a plug for the game connector on one end, and with a clip on the other end. The plug is devised so that you still have an empty game socket on top, for attaching paddles or whatever.

To install the Modifier, all you have to do is insert the plug into the game socket, and clip the other end onto the connector from the keyboard to the piggy-back board at the second wire from the right (the RESET key side).

I have installed them on all my Apples, except for my oldest one. (That one is serial #219, bought in August of 1977, and is so old it doesn't even have ventilation slots on the case! Yes, I installed the open-case-and-solder-a-wire modification in the old one.)

Now I can use the shift-key the way I was taught in typing class when I am using Data Capture 4.0, SuperText II, Apple Pie 2.0, the S-C Macro Assembler, or the Word Handler. And more and more programs are being created to take advantage of a REAL shift key on an Apple.

The normal retail price of the Ashby Shift-Key Modifier is $18. I have bought a bunch of them, and you can have them for only $15 each. They come complete with directions for installation.

==================================================================
DOCUMENT :AAL-8204:Articles:Front.Page.txt
==================================================================

$1.50

Volume 2 -- Issue 7                              April, 1982

In This Issue...



Another New Book:  Bag of Tricks

The authors of Beneath Apple DOS (Don Worth and Pieter Lechner) have done it again!  This time you get a diskette with four powerful disk utilities on it, and a book expaining their use.  The retail price is $39.95, but I will have them for only $36.

The utilities are TRAX, INIT, ZAP, and FIXCAT.  TRAX examines any track on a disk, reading it in as pure nibbles and displaying in a partially analyzed form.  INIT reformats any track or tracks, optionally retaining existing data in whatever readable sectors are in the track.  You can reorder the sectors, change the volume number, and more.  ZAP is a general purpose disk utility: sectors may be read, written, displayed, modified with a powerful assortment of over 50 commands.  It works with 13- and 16-sector DOS, as well as Pascal and CP/M diskettes.  You can even "program" in ZAP, with labels, loops, and macro-commands.  FIXCAT can automatically repair or reconstruct a catalog track by analyzing the rest of the disk.

Beyond the utilities themselves, there is about 40 pages of advanced tutorial material which starts where "Beneath Apple DOS" ends.

Unless you are fully satisfied with your present collection of disk utilities, you ought to get this set.

```
==============================================================================
DOCUMENT :AAL-8204:Articles:Pot.Tymac.Troub.txt
==============================================================================
```

Potential Trouble in TYMAC...................Robert H. Bernard

[ Bob is a subscriber in Westport, Connecticut. ]

The article by Peter Bartlett on improving the Epson Controller Card (which appeared in the February 1982 issue of AAL) has prompted me to write to bring to the attention of fellow AAL readers that the TYMAC controller card, which is a lower-cost alternative to the official Epson card, has a potentially serious problem.

To achieve slot independence, controller card ROM programs JSR to an RTS instruction in the Monitor. Then they extract the slot from the return address the JSR put on the stack. The Apple II Reference Manual details the process on page 81-82.

Most controller cards use the Apple technique verbatim, JSR'ing to $FF58, which is an RTS instruction in the Monitor ROM. However, the TYMAC card JSR's to $FDFF. That location also contains an RTS, so there is no problem using the TYMAC card as long as the Monitor ROM is enabled.

The problem occurs when the TYMAC card is used with Pascal. While Apple Computer has specifically guaranteed an RTS instruction at $FF58 in the Pascal Basic Input/Output System (BIOS), no RTS exists at $FDFF. Therefore TYMAC loses control and causes a Pascal crash as soon as it is called.

If any of you have TYMAC cards, and plan to make the Peter Bartlett modification (or perhaps even if you don't plan to), you should also change the JSR instruction at $0A relative to the beginning of the ROM from 20FFFD to 2058FF.

==============================================================================
DOCUMENT :AAL-8204:Articles:Recursive.Macro.txt
==============================================================================

Recursive Macro Example............................Lee Meador

[ Lee is a subscriber from Arlington, Texas.  He wrote the original code for the .TF directive and REPLACE command in the S-C Assemblers. ]

Here is short example of a useful macro that uses a recursive definition.  By recursive I mean that the definition calls itself.

Most large computers have a shift instruction which can shift any number of bits; the 6502 shifts only shift one bit at a time.  The LSR macro shown here accepts a shift count as the first parameter, and generates one LSR opcode for each bit shift you want.

The second parameter is optional.  If there is no second parameter, the A-register will be shifted.  If you specify a variable for the second parameter, that memory location will be shifted.  Both cases are shown in the example below.

How does it work?  The definiton says to test the first parameter; if it is greater than zero, generate the LSR with the optional second parameter as the address field, and call on the LSR macro with the first parameter decremented by one.  If the first parameter is zero (and it eventually will be), no code is generated.  Read the listing carefully, noting the indentation, and you should be able to follow it.

================================================================================
DOCUMENT :AAL-8204:Articles:Review.AED.II.txt
================================================================================

AED -- A New Applesoft Program Editor.............Reviewed by
Bob Sander-Cederlof

One of the joys of putting the Apple Assembly Line out each month has been the knowledge that a lot of readers are putting making good use out of what I print.  A case in point:  William Linn, of Lithonia, Georgia, was inspired by a combination of several articles to produce a new software product we all can use!

He calls it AED, which stands for Applesoft EDitor.  AED combines in one easy-to-use package:

Line Editing as in PLE and the S-C Macro Assembler
Automatic Line Numbering
Global Search and Replace (with wildcard matching)
Controlled LISTing (Page- or Line-at-a-time, and Slow Scroll)
Display of Variables after execution
Quick entry of DOS commands from a mini-menu
And a lot more.

I said it is easy to use.  Why?  Here are a few reasons:

The screen is split, with the line being entered at the bottom 6 lines and two possiblities for the top 18 lines.  The top 18 lines are used for listing or for display of the most frequently used commands and edit controls.

The commands and edit controls are single letters or control-letters, with mnemonic value.

An inverse letter appears before the prompt character indicating which of six special modes you are in, so you don't get lost.

Clicks and tones provide pleasant feedback at appropriate times.

One very unusual feature, which I have grown to love in a very short time, is a new kind of cursor.  Rather than the flashing cursor of the standard Apple input routines, AED alternates the underline character with the character already on the screen.  This alternation is done at the same rate as the Apple's flashing mode, but doesn't tire the eyes.

AED loads into memory from $8500 through $95FF, and uses a 256-byte buffer from $8400 to $84FF.  HIMEM is set to $83FF.

AED is normally in charge of all input, until the Control-Q command (QUIT) is typed.  If you type a letter A, C, E, F, L, M, R, S, or V the rest of the AED command starting with that letter will be displayed.  If the command requires no additional information, it is immediately executed.  Otherwise, it waits for you to finish the command and type a carriage return. The period is also a command: call it "dot", and think of "DOS", because its purpose is to call up the DOS Command Mini-Menu. If you type a line beginning with a non-command character, it is passed on to Applesoft. Thus you can enter numbered lines, or type immediate mode commands such as NEW or PRINT X(3) or PR#1.If you do leave AED control, typing "&" will enter AED again. If you have the Autostart Monitor, hitting RESET will re-enter AED.

It is important to realize that you are always in an editing mode. Even commands can be edited using the edit control keys.

Here is a list of the commands:

Letter Commands

```
A AUTO line #,increment
C CHANGE /string1/string2/A
E EDIT line #
F FILE = filename to use in
         DOS commands
L LIST [ line #,line # ]
M MANUAL line numbering
R Repeat last LIST command
S SEARCH /string/
V Variable display
. DOS Mini-Menu
```

Control Commands

```
^A  Assistance
^C  Clear Scroll Area
^Q  Quit
^X  Clear Edit Area
ESC Edit Next Line
```

Editing Commands

```
^B   Cursor to beginning
^D   Delete a character
^E   Cursor to end of line
^Fx  Cursor to next "x"
^I   Begin Insert mode
^M   (RETURN) Submit line
^N   Cursor to end of line
^R   Recall last line edited
^Tx  Delete through next "x"
^T^T Delete to end of line
^V   Next character verbatim
^W   Enter word cursor mode
```

AED does not have user-defined keyboard macros. The keyboard macros in PLE are a big selling point; however, the ones you actually end up using in PLE are built-in to AED as actual commands or as part of the

DOS Mini-Menu. Of course, PLE words with both Integer BASIC and Applesoft; AED is only for Applesoft.

If you use Applesoft, are not already firmly addicted to PLE, and if you do not use Integer BASIC, then you should consider picking up a copy of AED. It is only $40 (same price as PLE), and packs a lot of usefulness for the dollars.

```
==============================================================================
DOCUMENT :AAL-8204:Articles:Sftwr.Cnfg.Ctrl.txt
==============================================================================
```

Controlling Software Configuration................Don Taylor

Paul Schlyter's article on moving the S-C Assembler into the language card (AAL January 1982) couldn't have come at a better time for me. I was working on a project that had just outgrown the available memory space, and LANGASM came to the rescue. Long live LANGASM!

LANGASM and the extensions to the S-C Assembler that have appeared in the AAL bring to the fore an important subject: controlling the configuration of your copy of someone else's software.

How do I know that a particular "patched" copy I have of the assembler is compatible with another extension that will appear in next month's AAL? What kind of documentation must I keep somewhere to keep track of patched object code for which I have no source code? And how many different patched source code versions (to which I have given different names) of the S-C Assembler am I willing to keep track of?

For my use, I've chosen to keep track of only two modified copies of the assembler; I call them ASM II.1 and LANGASM.1. These two versions are simply the "standard issue" S-C Assembler Version 4.0 and LANGASM, each augmented with the listed .DA directive patch described by Bob in the December, 1980 issue of AAL. (I chose this configuration because the extension was written by Bob himself, and because other AAL articles have used the listed .DA directive. The feature is upward compatible, and listed .DAs presented to unmodified copies of the assembler will cause invisible errors by seemingly accepting those directives, while generating no code for items betond the comma.)

To add the extensions I want, I first load in ASM II.1 or LANGASM.1, and then modify the copy in memory with a configuration file before using it.

The source listing of LANGASM.1 EXT.SRC shows the method I use to add HOME, COPY and EDIT commands to my copy of LANGASM.1. This particular routine is .OR'd at the beginning of one of the 4K language card memory blocks located at $D000, which permits several extensions to be loaded in one contiguous area of memory, while leaving the main memory area free for the source file and symbol table.

Lines 1160-1570 install the patches in the memory-resident copy of LANGASM.1 and then return to a calling routine. Lines 1320-1400 patch the FAST command (disabled by the LANGASM patches) to render it a HOME command that works like Applesoft's does.

Lines 1260-1430 make similar modifications to LANGASM's command table entries, replacing LOAD with COPY and SAVE with EDIT, along with their assembled addresses (less one).

Lines 1440-1520 are the patches that were contained in Mike Laumer's source code for the EDIT command, found in the January, 1981 issue of AAL.

The source files for EDIT and COPY used within LANGASM.1 EXT.SRC in lines 1590 to the end of the file are identical to those written by Mike Laumer and Bob Sander-Cederlof, with a couple of exceptions. As stated above, the patch code for NML was moved to the modification area in lines 1440-1520. Second, all .OR and .TF directives were removed from both files. Third, a few redundant .EQ directives (internal assembler reference addresses) had to be removed to avoid any EXTRA DEFINITION errors. Finally, $D000 was added to all internal assembler references to make them compatible with LANGASM's $E000 origin.

To install these patches to LANGASM.1, I EXEC the following text file, which I call LANGASM:

```
CALL -151            (get into the monitor)
C0C1                 (turn off any firmware card)
C081 C081            (write enable the language card)
BLOAD LANGASM.1      (load LANGASM into the language card)
BLOAD MONITOR EXTENSIONS (load in page 3 extensions
                          from 10/81 issue of AAL)
BLOAD LANGASM.1 EXTENSIONS (load in the mods)
A5B8:80              (patch DOS to use the language card)
A5C0:81
300G                 (install monitor extensions)
C083                 (switch in Bank 2)
D000G                (install LANGASM mods)
3D3G                 (return to DOS and Applesoft)
INT                  (enter the assembler)
```

To use this method of in-memory configuration with ASM II.1 (where patches can't always be added in contiguous memory), I use a separate file for each command patch, each .OR'd at the proper address, and then install all patch routines within a single text file that is EXEC'd. Since I'm not dealing with the language card, and each of the commands added above are indepedent of one another, I can skip the EXEC and just BLOAD and install each command (or group of commands) I want to add with the monitor. The result is an easy configuration of the assembler, done at run time.

The use of configuration files to modify the assembler takes a few extra seconds (and a couple of extra files on my utility disk), but it is no more work thanks to the EXEC file. It permits me to keep only a single copy of the assembler (in a known configuration), while enabling me to fully document any modifications I make to the assembler with configuration files for which I have the source code. By creating different EXEC files, I can quickly and easily intermix configuration files to create (and document!) any version of the assembler I wish.

Even though I suppressed the listing of the EDIT and COPY commands to save newsletter space, the source code is on the Quarterly Disk (#7) which will include this program.

=====================================================================
DOCUMENT :AAL-8204:Articles:Using.Macros.txt
=====================================================================

Using Macros and Nested Macros.....................Art Schumer

[ Art is a subscriber in Manvel, North Dakota; he is the programming side of S&H Software. Art wrote the Universal Boot Initializer, The DOS Enhancer, and the AmperCat Utility. ]

The new S-C Macro Assembler is truly the best assembler around. With the addition of Macros, easier programming is limited only by your imagination. All you have to do is dream up some uses for Macros. Are Macros and Nested Macros really worth using? You bet! One of my source files was 104 sectors long, but after going back through it and implementing macros, the file shortened to only 96 sectors; it was also easier to read.

As Bob pointed out in the manual, nested macros are allowed in this new version, but he frowned on their use. I beg to differ with him, as I believe that nested macros can make your source files easier to read, as well as easier to write. They may seem complex at first, but after setting them up they become very easy to use.

In my example program, I've defined a macro called GOTO.XY that will take two variables and use them to position the cursor. Another defined macro called CLEAR.XY is a singly nested macro that uses GOTO.XY to position the cursor, and then clears from there to the end of screen. CLEAR.PRINT.XY positions the cursor (using GOTO.XY inside CLEAR.PRINT.XY), clears the rest of the screen, and prints a message. It may sound confusing, but after examining the source listing and th macro definitions, it should be easy to understand how this all works.

In all the macros, the first variable is the horizontal cursor position and the second variable is the vertical cursor position. CLEAR.PRINT.XY calls on a subroutine (JSR PRNT), which expects the message to follow the JSR instruction. The message is terminated by a 00 byte, and execution proceeds at the instruction which follows the message in memory.

The PRNT subroutine came from a Call A.P.P.L.E. article by Andy Hertzfeld.

Have fun with your new S-C Macro Assembler!

```
===============================================================================
DOCUMENT :AAL-8204:DOS3.3:Inst.LA.Taylor.txt
===============================================================================

CALL-151
C0C1 C081 C081
A5B8:80
A5C0:81
BLOAD LANGASM.1
BLOAD LANGASM.1 EXTENSIONS
BLOAD MONITOR EXTENSIONS
300G
C083
D000G
3D3G
INT
```

```
==============================================================================
DOCUMENT :AAL-8204:DOS3.3:S.Autosave.txt
==============================================================================

 1000  *---------------------------------
 1010  *       AUTOMATIC SAVE PROGRAM
 1020  *       THIS PROGRAM CHECK'S FOR A TITLE
 1030  *       AND IF ONE IS FOUND, THE CURRENT PROGRAM
 1040  *       IS SAVED UNDER THE TITLE
 1050  *       ALSO, IF THE VERSION NUMBER IS APPENDED
 1060  *       IT IS UPDATED BEFORE EACH SAVE
 1070  *---------------------------------
 1080  *       SYSTEM EQUATES
 1090  *---------------------------------
 1100  MON.COUT   .EQ $FDED
 1110  MON.CROUT  .EQ $FD8E
 1120  MON.BELL1  .EQ $FBDD
 1130  IN.BUF     .EQ $200
 1140  SRC.END    .EQ $4C,4D
 1150  SRC.START  .EQ $CA,CB
 1160  NEXT       .EQ $1D
 1170  SEARCH     .EQ $1E,1F
 1180  *---------------------------------
 1190         .OR $800
 1200         .TF AUTO.SAVE.OBJECT A$800
 1210  *---------------------------------
 1220  *       INITIALIZE SEARCH REGISTERS AND
 1230  *       DETERMINE IF AT END OF SOURCE PROGRAM
 1240  *---------------------------------
 1250  AUTO.SAVE
 1260         LDA SRC.START     GET START OF SOURCE PROGRAM ADDRESS
 1270         STA SEARCH        AND MOVE TO THE SEARCH ADDRESS
REGISTER
 1280         LDA SRC.START+1
 1290         STA SEARCH+1
 1300         CLD
 1310  ADDRESS.END.CMP
 1320         LDA SEARCH
 1330         CMP SRC.END       SEE IF AT END OF SOURCE PROGRAM
 1340         BNE .1
 1350         LDA SEARCH+1
 1360         CMP SRC.END+1
 1370         BEQ ERROR1        DIDN'T FIND TITLE
 1380  *---------------------------------
 1390  *       SEARCH LINE FOR OP CODE
 1400  *---------------------------------
 1410  .1     LDY #0       Y OFFSET FOR LINE EXAMINATION
 1420         LDA (SEARCH),Y    NEXT LINE OFFSET
 1430         STA NEXT
 1440         LDY #3       POINT TO CHARACTER AFTER LINE NUMBER
 1450         LDA (SEARCH),Y
 1460         CMP #'*      COMMENT LINE?
 1470         BEQ NEW.LINE      YEP
```

```
1480  .5     CMP #$C0     COMPRESSED CODE?
1490         BNE .2       NOPE
1500  .4     INY          MOVE OFFSET PAST COMPRESSED INFO
1510         INY
1520         CLV
1530         BVC .3       ...ALWAYS
1540  .2     CMP #$80     SPACE(S)?
1550         BCS OPCHK    YES, CHECK THE OP-CODE
1560  .3     INY
1570         LDA (SEARCH),Y
1580         BEQ NEW.LINE      END OF LINE (EOL) IS 0
1590         BNE .5       ...ALWAYS
1600  *---------------------------------
1610  *      CALCULATE ADDRESS OF NEXT LINE
1620  *---------------------------------
1630  NEW.LINE
1640         CLC
1650         LDA SEARCH   MOVE SEARCH ADDRESS TO NEXT LINE
1660         ADC NEXT
1670         STA SEARCH
1680         BCC ADDRESS.END.CMP
1690         INC SEARCH+1
1700         BNE ADDRESS.END.CMP   ...ALWAYS
1710  *---------------------------------
1720  *      ERROR ROUTINES
1730  *---------------------------------
1740  ERROR1
1750         LDY #0       POINT TO NO TITLE ERROR
1760  PRTERR LDA NO.TTL,Y
1770         BMI ERREND
1780         ORA #$80
1790         JSR MON.COUT
1800         INY
1810         BNE PRTERR
1820  ERREND JSR MON.COUT
1830         JSR MON.BELL1
1840         JSR MON.BELL1
1850         JSR MON.CROUT
1860         RTS
1870  ERROR2
1880         LDY #18      POINT TO ILLEGAL CHAR. ERROR
1890         BNE PRTERR   ...ALWAYS
1900  *---------------------------------
1910  *      CHECK THE OP CODE FOR .TI
1920  *---------------------------------
1930  OPCHK  LDX #0
1940  .1     INY
1950         LDA (SEARCH),Y
1960         BEQ NEW.LINE      EOL
1970         CMP OPS,X    COMPARE OP CODE
1980         BNE NEW.LINE THAT'S NOT IT
1990         INX
2000         CPX #3       IF ALL 3 COMPARE, FOUND OP CODE
2010         BNE .1
```

```
2020  *--------------------------------
2030  *       NOW LOOK FOR TITLE
2040  *--------------------------------
2050  TITLE   INY
2060          LDA (SEARCH),Y
2070          BEQ ERROR1    NO TITLE?
2080          CMP #',       LOOKING FOR COMMA (TITLE FOLLOWS)
2090          BNE TITLE
2100  .1      INY
2110          LDA (SEARCH),Y
2120          BEQ ERROR1    NO TITLE?
2130          CMP #$C0      COMPRESSED?
2140          BEQ COMP.CODE1
2150          CMP #$80      SPACE?
2160          BCS .1        YEP--SKIP
2170          CMP #'A       MAKE SURE 1ST CHAR. IS LETTER
2180          BCC ERROR2    NOT LETTER
2190          CMP #$5B      1 MORE THAN "Z"
2200          BCS ERROR2
2210  *--------------------------------
2220  *       TITLE FOUND
2230  *       OUTPUT CTRL-D, "SAVE" AND TITLE
2240  *--------------------------------
2250          PHA
2260          LDX #0
2270  .2      LDA SAVE,X
2280          JSR MON.COUT
2290          INX
2300          CPX #5
2310          BNE .2
2320          PLA
2330  NEXT.CHAR1
2340          ORA #$80
2350          JSR MON.COUT
2360          INX           X KEEPS TRACK OF INPUT BUFFER OFFSET
2370  NEXT.CHAR2
2380          INY
2390          LDA (SEARCH),Y
2400          BEQ GOT.TTL2 EOL--GOT THE TITLE
2410          CMP #',       NO COMMAS ALLOWED
2420          BNE .1
2430          LDA #'-       REPLACE COMMA WITH DASH
2440          BNE NEXT.CHAR1    ...ALWAYS
2450  .1      CMP #$C0
2460          BEQ COMP.CODE2
2470          CMP #$80
2480          BCC NEXT.CHAR1
2490          INY           CHECK FOR CHARACTER AFTER SPACE
2500          LDA (SEARCH),Y
2510          BEQ GOT.TTL1 DROP TRAILING SPACES
2520          DEY           MOVE POINTER BACK TO CORRECT POSITION
2530          LDA #$20      SPACE--SPACES IN TITLE COMPRESSED TO 1
2540          BNE NEXT.CHAR1    ...ALWAYS
2550  *--------------------------------
```

```
 2560  *      COMPRESSED CHARACTER ROUTINES
 2570  *-------------------------------
 2580  COMP.CODE1
 2590         INY
 2600         LDA (SEARCH),Y    THIS IS NUMBER OF CHARACTERS
COMPRESSED
 2610         STA NEXT
 2620         INY
 2630         LDA (SEARCH),Y    ACTUAL CHARACTER
 2640         CMP #'A      MAKE SURE IT'S A LETTER
 2650         BCC ERROR2
 2660         CMP #$5B
 2670         BCS ERROR2
 2680         PHA
 2690         LDX #0
 2700  .1     LDA SAVE,X
 2710         JSR MON.COUT
 2720         INX
 2730         CPX #5
 2740         BNE .1
 2750         PLA
 2760         BNE STORE    ...ALWAYS
 2770  COMP.CODE2
 2780         INY
 2790         LDA (SEARCH),Y
 2800         STA NEXT
 2810         INY
 2820         LDA (SEARCH),Y
 2830         CMP #',
 2840         BNE STORE
 2850         LDA #'-
 2860  STORE
 2870         ORA #$80
 2880         JSR MON.COUT
 2890         INX
 2900         DEC NEXT
 2910         BNE STORE
 2920         BEQ NEXT.CHAR2
 2930  *-------------------------------
 2940  *      SEARCH FOR VERSION NUMBER AND CHANGE IF FOUND
 2950  *-------------------------------
 2960  GOT.TTL1
 2970         DEY
 2980  GOT.TTL2
 2990         DEY          MOVE Y POINTER TO THIRD NON-BLANK
 3000         DEY          CHARACTER FROM THE END OF LINE
 3010         DEY
 3020         DEX
 3030         LDA (SEARCH),Y    THIRD CHAR. FROM END
 3040         CMP #'0
 3050         BCC DOS.OP
 3060         CMP #':      ASCII ":" IS 1 MORE THAN ASCII 9
 3070         BCS DOS.OP
 3080         INY
```

```
3090        LDA (SEARCH),Y    2ND CHAR. FROM END
3100        CMP #'.      SHOULD BE PERIOD
3110        BNE DOS.OP
3120        INY
3130        LDA (SEARCH),Y    LAST CHARACTER
3140        CMP #'0
3150        BCC DOS.OP
3160        CMP #':
3170        BCS DOS.OP
3180        ADC #1
3190        CMP #':
3200        BNE STORIT
3210        LDA #'0
3220        STA (SEARCH),Y    CHANGE DIGIT IN SOURCE CODE
3230        ORA #$80
3240        STA IN.BUF,X      CHANGE DIGIT IN DOS COMMAND
3250        DEX
3260        DEX
3270        DEY
3280        DEY
3290        LDA (SEARCH),Y
3300        CLC
3310        ADC #1
3320        CMP #':
3330        BNE STORIT
3340        LDA #'0
3350  STORIT STA (SEARCH),Y
3360        ORA #$80
3370        STA IN.BUF,X
3380  *--------------------------------
3390  *      CR OUTPUT CAUSES DOS TO PERFORM SAVE
3400  *      AFTERWARDS, RETURN TO ASSEMBLER
3410  *--------------------------------
3420  DOS.OP JSR MON.CROUT
3430  END    RTS
3440  *--------------------------------
3450  *      MESSAGES
3460  *--------------------------------
3470  OPS    .AS /.TI/
3480  SAVE   .HS 84       CTRL-D
3490         .AS -/SAVE/
3500  NO.TTL .AT /*** NO TITLE ERROR/
3510         .AT /*** ILLEGAL TITLE FIRST CHARACTER/
3520  ZZZEND .EQ *
3530  ZZZLEN .EQ ZZZEND-AUTO.SAVE
```

==============================================================================
DOCUMENT :AAL-8204:DOS3.3:S.FUNNY.NOISE.txt
==============================================================================

```
1000  *--------------------------------
1010  *      FUNNY NOISE
1020  *--------------------------------
1030  SPKR   .EQ $C030    SPEAKER TOGGLE ADDRESS
1040  KYBD   .EQ $C000    KEYBOARD INPUT
1050  STROBE .EQ $C010    KEYBOARD STROBE
1060  *--------------------------------
1070  PNTR   .EQ 0        ADDRESS OF CURRENT RANDOM VALUE
1080  *--------------------------------
1090  NOISE  JSR $FC58    CLEAR SCREEN, HOME CURSOR
1100  N0     LDY #0       POINT TO FIRST BYTE IN PAGE
1110         LDA #$D000   START AT $D000
1120         STA PNTR
1130         LDA /$D000
1140         STA PNTR+1
1150         JSR $FDDA    PRINT PAGE NUMBER
1160  N1     LDA SPKR     TOGGLE SPEAKER
1170         LDA (PNTR),Y GET HALF-CYCLE TIMER
1180         TAX
1190  N2     DEX          DELAY LOOP FOR HALF-CYCLE
1200         BNE N2
1210         INY          NEXT BYTE IN PAGE
1220         BNE N1
1230         INC PNTR+1   NEXT PAGE
1240         LDA PNTR+1   BYPASS I/O AREA
1250         CMP /$C000
1260         BEQ N0
1270         JSR $FDDA    PRINT PAGE NUMBER
1280         LDA KYBD     SEE IF ANY KEY PRESSED
1290         BPL N1       NO, KEEP MAKING NOISE
1300         STA STROBE   YES, CLEAR STROBE
1310         RTS          THAT'S ALL, FOLKS!
```

```
===========================================================================
DOCUMENT :AAL-8204:DOS3.3:S.LA.Ext.Taylor.txt
===========================================================================

 1000  *---------------------------------
 1010  *      INSTALL EXTENSIONS TO LANGASM
 1020  *
 1030  *      AUTHOR:  DON TAYLOR
 1040  *        DATE:  2/6/82, 4:00 PM
 1050  *
 1060  *---------------------------------
 1070         .OR $D000
 1080         .TF LANGASM.1 EXTENSIONS
 1090  *---------------------------------
 1100  DOS.REENTRY             .EQ $03D0
 1110  MON.HOME                .EQ $FC58
 1120  SCA.LOAD.CMD            .EQ $E246
 1130  SCA.SAVE.CMD            .EQ $E26E
 1140  SCA.SLOW.CMD            .EQ $E273
 1150  *---------------------------------
 1160  INSTALL.MODIFICATIONS
 1170         LDY #2            MODIFY ASSEMBLER
 1180  .1     LDA HOME.TABLE,Y  COMMAND JUMP
 1190         STA SCA.SLOW.CMD,Y
 1200         DEY
 1210         BPL .1
 1220         LDA #MON.HOME-1
 1230         STA SCA.SLOW.CMD+3
 1240         LDA /MON.HOME-1
 1250         STA SCA.SLOW.CMD+4
 1260         LDY #2
 1270  .2     LDA COPY.TABLE,Y
 1280         STA SCA.LOAD.CMD,Y
 1290         DEY
 1300         BPL .2
 1310         LDA #COPY-1
 1320         STA SCA.LOAD.CMD+3
 1330         LDA /COPY-1
 1340         STA SCA.LOAD.CMD+4
 1350         LDY #2
 1360  .3     LDA EDIT.TABLE,Y
 1370         STA SCA.SAVE.CMD,Y
 1380         DEY
 1390         BPL .3
 1400         LDA #EDIT-1
 1410         STA SCA.SAVE.CMD+3
 1420         LDA /EDIT-1
 1430         STA SCA.SAVE.CMD+4
 1440         LDA #$60       PATCH NML TO
 1450         STA $E125      MAKE IT A
 1460         LDA #$4C       SUBROUTINE
 1470         STA NML
 1480         STA $E078
```

```
1490        LDA #NEW.NML
1500        STA NML+1
1510        LDA /NEW.NML
1520        STA NML+2
1530        RTS
1540 *----------------------------------
1550 HOME.TABLE .AS ^HOM^
1560 COPY.TABLE .AS ^COP^
1570 EDIT.TABLE .AS ^EDI^
1580 *----------------------------------
1590 * COPY COMMAND FOR S-C ASSEMBLER
1600 * VERSION 4.0
1610 *
1620 * SOURCE: BOB SANDER-CEDERLOF 12/80
1630 *
1640 *----------------------------------
1650 *
1660 *
1670 * NOTE: COPY FUNCTION SOURCE IS
1680 *       ASSEMBLED HERE...
1690        .LIST OFF
1700 *
1710 *      COPY FUNCTION <COPY L1,L2,L3>
1720 *      L1= FIRST LINE OF RANGE TO COPY
1730 *      L2= LAST LINE OF RANGE TO COPY
1740 *      L3= LINE BEFORE WHICH TO INSERT COPY
1750 *
1760 *      ROUTINE BY BOB SANDER-CEDERLOF
1770 *      APPLE ASSEMBLY LINE 12/80
1780 *
1790 *----------------------------------
1800 SS     .EQ $00,01     START OF SOURCE BLOCK
1810 SE     .EQ $02,03     END OF SOURCE BLOCK
1820 SL     .EQ $04,05     LENGTH OF SOURCE BLOCK
1830 NEWPP  .EQ $06,07     NEW PROGRAM POINTER
1840 A0L    .EQ $3A,3B
1850 A0H    .EQ $3B
1860 A1L    .EQ $3C,3D
1870 A1H    .EQ $3D
1880 A2L    .EQ $3E
1890 A2H    .EQ $3F
1900 A4L    .EQ $42
1910 A4H    .EQ $43
1920 LOMEM  .EQ $4A,4B
1930 PP     .EQ $CA,CB
1940 *----------------------------------
1950 SYNX   .EQ $E05E
1960 MFER   .EQ $E128
1970 SCND   .EQ $E12D
1980 SERTXT .EQ $E4F6
1990 MON.MOVE .EQ $FE2C
2000 *----------------------------------
2010 ERR1   JMP SYNX
2020 ERR2   .EQ ERR1
```

```
ERR3   JMP MFER
ERR4   .EQ ERR1
*--------------------------------
COPY
       JSR SCND       GET THIRD PARAMETER
       CPX #6         BE SURE WE GOT THREE
       BCC ERR1       NOT ENOUGH PARAMS
       LDX #A0L       FIND BEGINNING OF SOURCE
       JSR SERTXT
       LDA $E4        SAVE POINTER
       STA SS
       LDA $E5
       STA SS+1
       LDX #A1L       FIND END OF SOURCE BLOCK
       JSR SERTXT
       SEC            SAVE POINTER AND COMPUTE
       LDA $E6        LENGTH
       STA SE
       SBC SS
       STA SL         SOURCE LENGTH
       LDA $E7
       STA SE+1
       SBC SS+1
       STA SL+1
       BCC ERR2       RANGE BACKWARD
       BNE .4
       LDA SL
       BEQ ERR2       NOTHING TO MOVE
*--------------------------------
.4     LDA PP         COMPUTE NEW PP POINTER
       SBC SL
       STA NEWPP
       LDA PP+1
       SBC SL+1
       STA NEWPP+1
*--------------------------------
       LDA NEWPP      SEE IF ROOM FOR THIS
       CMP LOMEM
       LDA NEWPP+1
       SBC LOMEM+1
       BCC ERR3       MEM FULL ERROR
*--------------------------------
       LDX #A2L       FIND TARGET LOCATION
       JSR SERTXT
       LDA SS         BE SURE NOT INSIDE SOURCE
       CMP $E4
       LDA SS+1
       SBC $E5
       BCS .1         BELOW SOURCE BLOCK
       LDA $E4
       CMP SE
       LDA $E5
       SBC SE+1
       BCC ERR4       INSIDE SOURCE BLOCK
```

```
2570  *  TARGET IS ABOVE SOURCE BLOCK, SO WE HAVE TO
2580  *  ADJUST SOURCE BLOCK POINTERS.
2590         SEC
2600         LDA SS
2610         SBC SL          SS=SS-SL
2620         STA SS
2630         LDA SS+1
2640         SBC SL+1
2650         STA SS+1
2660         SEC
2670         LDA SE
2680         SBC SL          SE=SE-SL
2690         STA SE
2700         LDA SE+1
2710         SBC SL+1
2720         STA SE+1
2730  *---------------------------------
2740  .1     LDA PP          SET UP MOVE TO MAKE HOLE
2750         STA A1L
2760         LDA PP+1
2770         STA A1H
2780         LDA NEWPP
2790         STA PP
2800         STA A4L
2810         LDA NEWPP+1
2820         STA PP+1
2830         STA A4H
2840         LDA $E5
2850         STA A2H
2860         LDA $E4
2870         STA A2L
2880         BNE .2
2890         DEC A2H
2900  .2     DEC A2L         A2=A2-1
2910         LDY #0
2920         LDA A2L
2930         CMP A1L
2940         LDA A2H
2950         SBC A1H
2960         BCC .5
2970         JSR MON.MOVE    A4<A1.A2M
2980  *---------------------------------
2990  .5     LDA SS          MOVE IN SOURCE BLOCK
3000         STA A1L         (MON.MOVE LEFT
3010         LDA SS+1        A4 POINTING AT FIRST
3020         STA A1H         BYTE OF THE HOLE)
3030         LDA SE+1
3040         STA A2H
3050         LDA SE
3060         STA A2L
3070         BNE .3
3080         DEC A2H         A2=A2-1
3090  .3     DEC A2L
3100         JSR MON.MOVE    A4<A1.A2
```

```
3110        RTS
3120        .LIST ON
3130 *
3140 *
3150 * NOTE: EDIT FUNCTION SOURCE IS
3160 *       ASSEMBLED HERE...
3170        .LIST OFF
3180 *
3190 *
3200 *--------------------------------
3210 * EDIT COMMAND FOR S-C ASSEMBLER
3220 * VERSION 4.0
3230 *
3240 * SOURCE: MIKE LAUMER 12/6/80
3250 *
3260 *--------------------------------
3270 *
3280 *  SYSTEM EQUATES
3290 *--------------------------------
3300 MON.COUT     .EQ $FDED
3310 MON.BELL     .EQ $FF3A
3320 MON.RDKEY    .EQ $FD0C
3330 MON.CLREOP   .EQ $FC42
3340 MON.VTAB     .EQ $FC22
3350 CH           .EQ $0024
3360 CV           .EQ $0025
3370 *--------------------------------
3380 *  ASSEMBLER EQUATES
3390 *--------------------------------
3400 GNL    .EQ $E026
3410 NML    .EQ $E063
3420 PLNO   .EQ $E779
3430 GNB    .EQ $E2C5
3440 DOIT   .EQ $E874
3450 SEARCH .EQ $E64B
3460 SERNXT .EQ $E4FE
3470 NTKN   .EQ $E2AF
3480 SRCP   .EQ $DD,DE
3490 WBUF   .EQ $0200
3500 CURRENT.LINE.NUMBER .EQ $D3,D4
3510 *--------------------------------
3520 * PATCH ROUTINES FOR ASSEMBLER
3530 *--------------------------------
3540 NEW.NML JSR MY.NML
3550        JMP GNL
3560 MY.NML LDY #0
3570        JSR $E28D
3580        JSR $E14A
3590        JMP $E066
3600 *--------------------------------
3610 * LOCAL VARIABLES FOR EDIT COMMAND
3620 *--------------------------------
3630 NEXT   .DA 0
3640 END    .DA 0
```

```
3650  CHAR   .DA #0
3660  EDPTR  .DA #0
3670  FKEY   .DA #0
3680  *---------------------------------
3690  EDIT   DEX
3700         DEX
3710         BMI .2          NO ARGUMENTS
3720         BEQ .4          1 ARGUMENT
3730         JSR .3          2 ARGUMENTS
3740         LDX #A1L        FIND END PTR
3750         JSR SERNXT
3760         LDA $E6
3770         STA END
3780         LDA $E7
3790         STA END+1
3800  .1     LDA NEXT+1
3810         STA SRCP+1
3820         PHA
3830         LDA NEXT
3840         STA SRCP
3850         CMP END
3860         PLA
3870         SBC END+1       PAST END LINE?
3880         BCS .2          YES, EXIT
3890         JSR E.LIST      NO, LIST AND EDIT
3900         JMP .1          TRY FOR NEXT LINE
3910  .3     LDX #A0L        FIND START PTR
3920         JSR SERTXT
3930         LDA $E4
3940         STA SRCP
3950         STA NEXT        SAVE NEXT LINE ADRS
3960         LDA $E5
3970         STA SRCP+1
3980         STA NEXT+1
3990  .2     RTS
4000  .4     JSR .3          SEARCH FOR LINE
4010         BCC .2          NOT FOUND EXIT
4020  E.LIST JSR E.POSN      POSITION FOR EDIT
4030         JSR MON.CLREOP  PREPARE DISPLAY
4040         JSR GNB         GET LINE SIZE
4050         CLC
4060         ADC NEXT        COMPUTE NEXT LINE ADRS
4070         STA NEXT
4080         TYA
4090         ADC NEXT+1
4100         STA NEXT+1
4110         JSR GNB         GET LINE # FOR DISPLAY
4120         STA CURRENT.LINE.NUMBER
4130         JSR GNB
4140         STA CURRENT.LINE.NUMBER+1
4150         SEC
4160         ROR $F8         STUFF WBUF FLAG
4170         JSR PLNO
4180         LSR $F8         TURN OFF FLAG
```

```
4190          LDA #$20       SPACE AFTER LINE #
4200          LDX #0
4210  .1      STX EDPTR
4220          ORA #$80       FORCE VIDEO BIT
4230          STA WBUF+4,X   STORE INTO INPUT BUFFER
4240          CMP #$A0       TEST FOR CONTROL CHAR
4250          BCS .2         OK, IF NOT
4260          AND #$7F       OUTPUT INVERSE ALPHA
4270  .2      JSR MON.COUT   PRINT CHAR
4280          JSR NTKN       GET NEXT TOKEN
4290          LDX EDPTR
4300          INX
4310          CMP #0         END TOKEN?
4320          BNE .1         NO,PRINT IT
4330          STA WBUF+4,X   YES,PUT IT IN TOO
4340  E.LINE LDX #0
4350  E.0     STX EDPTR
4360  E.1     JSR E.INPUT    GET INPUT CHAR
4370  E.2     LDA #EDTB
4380          STA $2
4390          LDA /EDTB
4400          STA $3
4410          LDA #CHAR
4420          STA $12
4430          LDA /CHAR
4440          STA $13
4450          JSR SEARCH     SEARCH EDIT COMMAND TABLE
4460          BNE .2         NOT IN TABLE
4470          LDX EDPTR
4480          JSR DOIT       EXECUTE COMMAND ROUTINE
4490          BCC E.0        NO DISPLAY ON RETURN
4500          BCS .5         DISPLAY ON RETURN
4510  .2      LDX EDPTR      MUST BE TYPE OVER
4520          LDA CHAR
4530          CMP #$A0
4540          BCS .4
4550  .3      JSR MON.BELL   ERR IF CONTROL KEY
4560          JMP E.1
4570  .4      LDA WBUF+5,X   SEE IT END OF LINE
4580          BNE .6         TYPE OVER IF NOT
4590          STA WBUF+6,X   SHIFT OVER END OF LINE
4600  .6      LDA CHAR       STUFF CHAR INTO BUFFER
4610          STA WBUF+5,X
4620          CPX #256-5-2   TEST BUFFER SIZE
4630          BEQ .5         TYPE OVER LAST CHAR IN BUFFER
4640          INX            INSTEAD OF BUFFER END
4650  .5      JSR E.DISP     DISPLAY LINE
4660          JMP E.0        GET NEXT EDIT COMMAND
4670  *---------------------------------
4680  E.POSN LDA #19        POSITION TO LINE 19,
4690          STA CV
4700          LDA #0         COLUMN 0
4710          STA CH
4720          JMP MON.VTAB
```

```
*--------------------------------
E.DISP STX EDPTR
       JSR E.POSN     POSITION DISPLAY
       LDX #$FF
.1     INX
       LDA WBUF,X     GET BUFFER CHAR
       BEQ .3         END OF BUFFER
       CMP #$A0       CONTROL CHAR?
       BCS .2         NO
       AND #$7F       PRINT INVERSE ALPHA
.2     JSR MON.COUT   PRINT CHAR
       JMP .1         NEXT CHAR
.3     JSR MON.CLREOP CLEAN ANY REMAINING SCREEN
       LDX EDPTR
       RTS
*--------------------------------
E.BEG  LDX #0         SET CURSOR TO BEGINNING OF LINE
       CLC
       RTS
*--------------------------------
E.DEL  LDA WBUF+5,X   IS THIS END
       BEQ .2
.1     INX
       LDA WBUF+5,X   SHIFT TO LOWER MEMORY
       STA WBUF+4,X   TO DELETE CHAR
       BNE .1
       LDX EDPTR
.2     SEC            RETURN WITH DISPLAY
       RTS
*--------------------------------
E.END  LDA WBUF+5,X   END OF BUFFER?
       BEQ .1         YES
       INX            NO
       BNE E.END      TRY END AGAIN
.1     CLC            RETURN NO DISPLAY
       RTS
*--------------------------------
E.FIND LDA WBUF+5,X   END OF BUFFER?
       BNE .2         NO
.1     STA FKEY       YES SO ERR
       JSR MON.BELL   RING BELL
       CLC            RETURN NO DISPLAY
       RTS
.2     JSR E.INPUT    GET 1 CHAR
       STA FKEY       SAVE KEY TO LOCATE
.3     INX
       LDA WBUF+5,X   TEST BUFFER
       BEQ .1         END OF BUFFER
       CMP FKEY       NO, SEE IF KEY
       BNE .3         NO, GO FORWARD
       JSR E.INPUT    TRY ANOTHER KEY
       CMP FKEY       SAME CHAR?
       BEQ .3         YES, SEARCH AGAIN
       PLA
```

```
5270        PLA
5280        STX EDPTR      NO, EXIT POINTING HERE
5290        JMP E.2
5300 *---------------------------------
5310 E.BKSP TXA            AT BEGINNING?
5320        BEQ .1         YES, STAY THERE
5330        DEX            BACKUP
5340 .1     CLC            RETURN NO DISPLAY
5350        RTS
5360 *---------------------------------
5370 E.OVR  JSR E.INPUT    READ CHAR
5380        JMP E.INS1     SKIP CONTROL CHECK
5390 *---------------------------------
5400 E.INS  JSR E.INPUT    READ CHAR
5410        CMP #$A0       CONTROL CHAR POPS USER OUT
5420        BCC E.INS2     OF INSERT
5430 E.INS1 CPX #256-5-2   END OF BLOCK
5440        BEQ .1         YES STAY THERE
5450        INX
5460 .1     STX EDPTR
5470 .2     PHA            CHAR TO INSERT
5480        LDA WBUF+4,X   SAVE CHAR TO MOVE
5490        TAY
5500        PLA            GET CHAR TO INSERT
5510        STA WBUF+4,X   PUT OVER SAVED CHAR
5520        INX
5530        TYA            INSERT SAVED CHAR
5540        BNE .2         IF NOT BUFFER END
5550        STA WBUF+4,X   STUFF END CODE
5560        STA WBUF+256-5-1 INSURE AN END CODE
5570        LDX EDPTR
5580        JSR E.DISP     DISPLAY LINE
5590        JMP E.INS      GET NEXT INSERT CHAR
5600 E.INS2 PLA            SEND CHAR TO
5610        PLA            COMMAND SEARCH
5620        LDX EDPTR
5630 *---------------------------------
5640        JMP E.2
5650 E.RETQ LDA #0         CLEAR REST OF LINE
5660        STA WBUF+5,X
5670        JSR E.DISP     DISPLAY LINE
5680 E.RET  LDX #$FF       SUBMIT LINE TO ASSEMBLER
5690 .1     INX            COMPUTE LINE SIZE
5700        LDA WBUF,X
5710        BNE .1
5720        DEX
5730 .2     STX $E1        SAVE SIZE
5740        PLA
5750        PLA
5760        JMP MY.NML     SUBMIT THE LINE
5770 *---------------------------------
5780 E.TAB  CPX #20        <COL 20?
5790        BCS .1         NO
5800        LDA WBUF+5,X   END OF BUFFER?
```

```
5810        BEQ .1         YES
5820        INX            MOVE FORWARD
5830        CPX #7         TAB MATCH?
5840        BEQ .1
5850        CPX #11        TAB MATCH?
5860        BNE E.TAB
5870 .1     CLC            RETURN WITHOUT DISPLAY
5880        RTS
5890 *---------------------------------
5900 E.RIT  LDA WBUF+5,X   END OF BUFFER?
5910        BNE .1         NO
5920        STA WBUF+6,X
5930        LDA #$A0       PUT A BLANK
5940        STA WBUF+5,X   TO EXTEND LINE
5950        CPX #256-5-2
5960        BEQ .2
5970 .1     INX            MOVE AHEAD
5980 .2     CLC            RETURN NO DISPLAY
5990        RTS
6000 *---------------------------------
6010 E.ABORT LDA #$DC      OUTPUT BACKSLASH
6020        STA WBUF+5
6030        LDA #0
6040        STA WBUF+6
6050        JSR E.DISP     SHOW CANCEL
6060        JMP GNL        GET NEXT COMMAND
6070 *---------------------------------
6080 E.INPUT LDA #19
6090        STA CV
6100        TXA            POSITION TO CURSOR
6110        CLC
6120        ADC #5
6130 .1     CMP #40        THIS LINE?
6140        BCC .2         YES
6150        SEC
6160        SBC #40
6170        INC CV         ON NEXT LINE
6180        BNE .1
6190 .2     STA CH
6200        JSR MON.VTAB   SET BASL
6210        JSR MON.RDKEY  INPUT A CHAR
6220        STA CHAR
6230        RTS
6240 *---------------------------------
6250 *  COMMAND TABLE
6260 *---------------------------------
6270 EDTB   .DA #3,#1      ITEM SIZE, KEY SIZE
6280        .DA #$82,E.BEG-1   ^B
6290        .DA #$84,E.DEL-1   ^D
6300        .DA #$8E,E.END-1   ^N
6310        .DA #$86,E.FIND-1  ^F
6320        .DA #$88,E.BKSP-1  ^H
6330        .DA #$89,E.INS-1   ^I
6340        .DA #$8D,E.RET-1   ^M
```

```
6350        .DA #$8F,E.OVR-1   ^O
6360        .DA #$91,E.RETQ-1  ^Q
6370        .DA #$94,E.TAB-1   ^T
6380        .DA #$95,E.RIT-1   ^U
6390        .DA #$98,E.ABORT-1 ^X
6400        .DA #0
6410        .EN
```

```
==============================================================================
DOCUMENT :AAL-8204:DOS3.3:S.Recurs.Macro.txt
==============================================================================

 1000        .MA LSR
 1010        .DO ]1>0
 1020        LSR ]2
 1030        >LSR ]1-1,]2
 1040        .FIN
 1050        .EM
 1060 *--------------------------------
 1070        >LSR 3,$12
 1100        >LSR 2
```

```
==================================================================
DOCUMENT :AAL-8204:DOS3.3:S.Schumer.Macro.txt
==================================================================

 1000  *---------------------------------
 1010  * USE OF MACROS & NESTED MACROS
 1020  * BY ART SCHUMER - 3/25/82
 1030  *---------------------------------
 1040  VTAB   .EQ $FB5B
 1050  CLREOP .EQ $FC42
 1060  HOME   .EQ $FC58
 1070  RDKEY  .EQ $FD0C
 1080  COUT   .EQ $FDED
 1090  *---------------------------------
 1100  PTR    .EQ $6
 1110  CH     .EQ $24
 1120  CV     .EQ $25
 1130  *---------------------------------
 1140  *      MACRO DEFINITIONS
 1150  *
 1160  * CLR.PRNT.XY AND GOTO.PRNT.XY
 1170  * ARE EXAMPLES OF NESTED MACROS
 1180  *---------------------------------
 1190         .MA GOTO.XY
 1200         LDA #]1
 1210         STA CH
 1220         LDA #]2
 1230         JSR VTAB
 1240         .EM
 1250  *---------------------------------
 1260         .MA CLEAR.XY
 1270         >GOTO.XY ]1,]2
 1280         JSR CLREOP
 1290         .EM
 1300  *---------------------------------
 1310         .MA CLEAR.PRNT.XY
 1320         >CLEAR.XY ]1,]2
 1330         JSR PRNT
 1340         .EM
 1350  *---------------------------------
 1360         .MA GOTO.PRNT.XY
 1370         >GOTO.XY ]1,]2
 1380         JSR PRNT
 1390         .EM
 1400  *---------------------------------
 1410         .MA READ.XY
 1420         >GOTO.XY ]1,]2
 1430         JSR RDKEY
 1440         .EM
 1450  *---------------------------------
 1460  *      THE PROGRAM . . . .
 1470  *---------------------------------
 1480  START  JSR HOME
```

```
1490        >GOTO.PRNT.XY 4,12
1500        .AS -/THIS EXAMPLE USES NESTED MACROS/
1510        .HS 00
1520        >READ.XY 36,12
1530        >CLEAR.PRNT.XY 4,12
1540        .AS -/AND THIS ONE ALSO!/
1550        .HS 00
1560        >READ.XY 22,12
1570        RTS
1580 *--------------------------------
1590 * ANDY HERTZFELD'S PRINT ROUTINE
1600 *--------------------------------
1610 PRNT   PLA
1620        STA PTR
1630        PLA
1640        STA PTR+1
1650        LDY #0
1660 .1     INC PTR
1670        BNE .2
1680        INC PTR+1
1690 .2     LDA (PTR),Y
1700        BEQ .3
1710        JSR COUT
1720        JMP .1
1730 .3     LDA PTR+1
1740        PHA
1750        LDA PTR
1760        PHA
1770        RTS
1780 *--------------------------------
```

=====================================================================
DOCUMENT :AAL-8205:Articles:Anthr.Recur.Mac.txt
=====================================================================

Another Recursive Macro...........................Lee Meador

Last month I sent Bob a recursive macro definition that he put in the AAL for everyone to see. In case you have forgotten what recursive means, let me explain it somewhat. If you have a macro thatcalls itself under certain conditions, that macro is called 'recursive'. It's kind of like the plastic cup I had when I was little. There was a picture on the cup of a little bear sitting in a high-chair. The bear was holding a plastic cup and on the cup was a picture of a little bear in a high-chair holding a plastic cup with a picture of a bear in a high-chair holding a cup with a picture of a bear......

I always used to wonder how many bears there were and how big the littlest one was. Now, when we are using recursion in our macros we want to be sure we know that there is a last little bear -- a last call -- a way of leaving the lowest level of recursion. Otherwise (since each recursive call uses up some memory/stack space) we will soon run out of room to store the return information and BOOM goes the assembly.

My macro uses the principle of 'divide and conquer' to allocate a chosen number of bytes all of which hold the value we want them to have. We might use this macro with a table of 128 bytes. All non-alphabetic characters (codes $0 to $40, plus assorted others) will have the value $FF in their corresponding bytes in the table. All alphabetics will have a number indicating their relative frequency in English text. We could set up the table with lines and lines of hex strings for all the non-alphabetics. Or we could let the program filter out the alphabetics and use a shorter table. But for the sake of the example let us assume we need the program to be as fast as possible and memory space is no object.
Here is the macro definition I came up with:

```
.MA DB          Macro name is "DB"
.DO ]1<2        If only one left,
.DA ]2             generate a data byte
.ELSE           If more than one left,
>DB ]1/2,]2        call DB for half of them,
>DB ]1+1/2,]2      and call DB again for the other half
.FIN
.EM
```

Here is the table I talked about:

```
>DB $41,#$FF   65 bytes filled with $FF
.HS 00000000   upper case (fill in your own frequencies)
......
>DB 5,#$FF     5 bytes filled with $FF
.HS 00000000   lower case
```

```
......
>DB 5,#$FF     5 bytes filled with $FF
```

Here is a sample program to use such a table:

```
LDA CHARACTER.BYTE    get character
TAY
LDA TABLE,Y           get frequency
......                and then you have to use it
```

The macro calls itself to set up half of the area desired and then calls itself again to set up the other half. The adding of one to the second call makes sure that both odd and even values for the first parameter will work. If the call only needs one byte to be set up then a .DA is used to take care of it. That provides the end of the little bears -- when the first parameter is one.

When I needed a macro like this my first idea was to have each recursive call take care of one byte and then call itself to take care of the rest. If the macro was called with zero repetitions then nothing would be done except end the macro. The problem with that method is the amount of stack space used as the recursion goes to very deep levels. The method used in the example will only recurse, for example, 8 levels to generate 127 bytes of data.

By the way, notice that you must put the pound sign (#) on the second parameter if you want to generate single bytes. Leaving it off will generate two-byte values of data. I chose that method to make the macro more flexible. You might want to put the pound sign (#) inside the macro to make it safer in case you always want to generate single bytes of data. Also, you can use calculated values like #'F+$80 to generate tables of some character value.

The assembly of recursive macros produces quite a few extra lines in the listing, so after checking it out you will probably want to turn off the listing of the macro expansion with ".LIST MOFF". Here is a sample listing with the macro expansion listing on:

<listing here>

==============================================================================
DOCUMENT :AAL-8205:Articles:BlkMv.Benchmrk.txt
==============================================================================

Benchmarking Block MOVES..................William R. Savoie

While working on my new soon-to-be-released data file system, I came to see the importance of a speedy 6502 block move routine. I have resisted "moves" like the coming of winter. I have a super two pass sort routine that is very fast. Providing a person has less than 2000 files there is no need to do much file moving, since only pointers need move. (My system has a 64K card, giving me a 112K system.)

Unfortunately, the real world needs some 10,000 or more files, and these of course must be sorted too. By physically moving the files as directed by the sorted pointers, and then moving all this to disk, it is possible to use a merge sort to get the whole job done in the least amount of time. With this preamble behind us, let's get on the move!

I have benchmarked three approaches to moving blocks: the monitor move down (located at $FE2C) which I'm sure you all have used, and its variation, a similar move up routine. Next is the Applesoft block move, and third is a self modifying move which I call "Quick Move".

To ease such a tedious undertaking, I have included a BASIC connection to pass variables and determine the benchmark precision. To help further, I have added a hex converter, a memory dump routine, and an automatic 3D0G vector using the ctrl-Y command from the monitor. To help with the problem of what block of memory was moved where, I wrote a memory fill routine. This acts to place the memory address back into memory, on two-byte boundaries. You can easily read memory to see where it came from.

My first benchmark was a block move of 10,000 bytes made 100 times. The next was a move of 10,240 bytes, again made 100 times. Here are the conditions and resulting times:

| | | mon up | mon dn | AS | QM | |
|---|---|---|---|---|---|---|
| Case I | Lo=18674=$48F0 | 47 | 53 | 17.2 | 15.3 | seconds |
| Case II | Lo=18432=$4800 | 48.7 | 54.7 | 16.7 | 14.7 | |

Note: 0.5 seconds of these values due to BASIC overhead.

As you can see, the old monitor move is not made for high speed moves. For one thing, a two-byte subtraction is carried out for each byte that is moved. It is much more efficient to do the subtraction only once, before you start. A closer look shows that it is faster to move more data, providing you move a whole number of memory pages! The time needed to move the "extra" 240 bytes was negative 0.5 seconds for the Applesoft block move and negative 0.6 seconds for the "Quick Move". There was no sensitivity to start and destination boundaries. "Quick Move" was 3.7 times faster than the monitor move!

I tried putting the first half of Quick Move on page zero at $A0, but the speed improvement was only 0.7 seconds (about 5%) over the time it took when located at $3000.

As a further note, each move routine requires its own parameter organization.  If files are to be moved and not lost, attention must be paid to exact specification of end points and lengths.

================================================================================
DOCUMENT :AAL-8205:Articles:Branch.MacLIb.txt
================================================================================

Macro Branch Library........................................R. F. O'Brien

When I received my copy of the S-C Macro Assembler, my first task was to make up a set of branch macro definitions to use in all my programs. This set will finally eliminate the need to check usage of BCC, BCS, etc., and generally make the programs more readable.

There are six branch-on-tests: >BLT, >BLE, >BGE, >BGT, >BEQ, AND >BNE. All of these would normally be used with two parameters, e.g. >BGT P1,P2...reads: if contents of accumulator is greater than P1 then branch to P2. The first four of these can be gainfully used with only one parameter, after a comparison. Sample program:

```
LDA #$8D
CMP #$8E
>BLT THERE ... results in a branch to THERE
```

While >BEQ and >BNE are defined so as to work with only one parameter, there is no reason to so use them - it is easier to just use BEQ and BNE.

The macro >BRA (BRanch Always) is used with one parameter - any others are ignored. >BRA LABEL causes a jump to LABEL, up to +- 127 bytes away. To overcome this limitation I decided to put the macro facility to good use to provide for easy branching to any part of a program - this is necessary for writing relocatable code. I settled for a two-paramter code >JMP:

>JMP P1,P2 ... where P1 is the intermediate or final label you wish to branch to and P2 is the label for this instruction.

Instructions such as the foregoing are inserted anywhere you wish in the program (within 127 bytes of each other) to allow for unlimited branching whilst retaining relocatable code. The following is an example of how you might use the >JMP code:

```
1000 A    etc.
          (more code....)
2000      >JMP A,B
          (more code....)
3000      >BRA B
```

When a program designed as the above is run it will simulate an absolute jump to A. The >BRA B will branch to label B, which contains a >BRA A. This sequence of instructions is transparent to the rest of the program, as the first instruction in the >JMP is to skip around the the >BRA within the definition.

This use of macro definitions can easily be extended to the X and Y registers; simply substitute CPX's or CPY's for the CMP's.

Following are some examples of these macros at work:

```
>BLT #3,THERE....if (A) is less than 3 then go THERE
>BGT $40,THIS....if (A) is greater than ($40) then go THIS
>BEQ #'A,THAT....if (A) is equal to $41 then go THAT
```

To use these macros in all your programs, place the command .IN MACRO.BRANCH.LIBRARY at the beginning of your source program.

==============================================================================
DOCUMENT :AAL-8205:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 2 -- Issue 8                              May, 1982

In This Issue...



Macro Assembler in EPROM

A number of you have asked about the possiblility of getting the S-C Macro Assembler in EPROM to put on an Apple Firmware card, or a CCS 12K Eprom card.  If you want to do it, and know how to modify the firmware card to accept EPROMs, I will send a set of 5 EPROMs containing the assembler for $64.  If you also need the monitor in EPROM, add another $16.  The assumption will be that you have Applesoft on the mother board, and that you already own the S-C Macro Assembler on disk.

Advertising in AAL

Due to the increased costs of printing more than 1600 copies per month, and with the desire to limit the percentage of advertising pages to less than 30% each month, I have decided to raise the page rate again.

For the June 1982 issue, the price will be $50 for a full page, $30 for a half page.  So-called "classified" ads, of up to forty words, will be $5.

==============================================================================
DOCUMENT :AAL-8205:Articles:Game.Buttons.txt
==============================================================================

Reading the Game Buttons...........................Jim Kassel

Recently I was asked to come up with a maching language subroutine that involved using the Apple game buttons. Fortunately for me at the time, I forgot that my paddles were not plugged in. I was in for a rude awakening because when I tested the program, it said that all of the buttons were constantly being pushed!

I needed some additional programming to check whether the buttons were even plugged in. The problem occurs because the Apple button logic returns the same value for a pushed button as for a missing button. To get technical: in TTL logic, when an input pin of an IC chip is left unconnected, the chip thinks the pin is at a logic "1". The Apple buttons supply a logic "1" when they are plugged in and pushed. Hence, the hardware cannot tell the difference between a plugged-in-pushed-down button and a missing button.

What the hardware does know for sure is when a button is plugged in and not pushed. This is the only case in which a logic "0" is developed. I had to use this knowledge to write a program which could tell what a logic "1" really means. Since an installed unpushed button does unambiguously announce its presence by a "0" in bit 7 of the input byte, I could make a mask indicating which buttons appear to be installed.

I started by writing the GET.BUTTON.STATUS subroutine. It reads each of the three buttons, and packs the three bit-7's into one byte. The way I wrote it, bit 7 of the returned byte represents button 1, bit 6 is button 2, and bit 5 is button 3. If a button is installed and not pushed, the corresponding bit will be "1"; otherwise it will be "0".

Look at the listing, lines 1250-1350, and I'll describe how GET.BUTTON.STATUS works. I used an indexed loop, where X goes from 2 to 0, step -1, addressing all three of the buttons. Only bit 7 of a button byte is significant. I invert this bit (line 1280), and shift it into the carry status bit (line 1290). Then line 1300 rolls the bit into GB.PUSH. After all three have been read and rolled, I pick up GB.PUSH and zero out the lower five bits at line 1340.

Now lets look at the other three subroutines. GAME.BUTTON.INITIALIZE simply clears out GB.STAT. We have to start with GB.STAT = 0, so that as we discover each installed button we can set its bit. Call this subroutine once at the beginning of your overall run.

GAME.BUTTON.INSTALLED reads the current button status; remember that a "1" here indicates an installed but unpushed button. So line 1140 merges all "1" bits into GB.STAT. You need to call this subroutine several times, at time intervals of several seconds at least, to be sure that every installed button is noticed at least once. (The first few times you call it, you might be pushing down an installed button; then finally you let go, so this subroutine sees the button.)

GAME.BUTTON.PUSHED reads the current button status, and with a little boolean logic comes out with the final result: a "1" indicating an installed and pushed button, and a "0" meaning either a missing button or an unpushed button. The result is in the A-register, and also in GB.PUSH.

Here is a truth table of the logic involved:

| GB.STAT | CURRENT READING | EOR STAT | AND STAT |
|---|---|---|---|
| 0 (no button) | 0 (none or pushed) | 0 | 0 |
| 0 (no button) | 1 (unpushed) | 1 | 0 |
| 1 (button) | 0 (none or pushed) | 1 | 1 |
| 1 (button) | 1 (unpushed) | 0 | 0 |

There are other possible complications in reading buttons, which I have not handled here. You might want to "debounce" the buttons, so that you don't get false indications of multiple pushes when the button begins to make or break contact. And, you might want to guarantee that any action which happens from a pushed button only happens once per push.

Lines 1400-1910 demonstrate the usage of the subroutines. After clearing the screen, the buttons are continuously monitored until you press any key on the keyboard. The status of each button will be displayed on the screen: button 1 on the to line, button 2 on the second line, and button 3 on the third line. If you hold a button pushed and then start the program, it will say "not installed" until you release the button; from then on it will track the button properly.

If you have the shift key mod installed in button 3, it will say "not installed" until you press the shift key; from then on it will say "pushed" when you are not pushing, and "not pushed" when you are. This is because the sense of the shift key is the opposite of the normal game paddle buttons.

================================================================
DOCUMENT :AAL-8205:Articles:NewAEDFeatures.txt
================================================================

New AED Features............................Bob Sander-Cederlof

Bill Linn, author of AED, stopped by the other day.  Bill is a Vice President at Cullinane Corporation, and was in Dallas for a user convention.  Since it was Sunday, and we are both earnest Christians, he spent the morning with Becky and me and the kids at church.  Later we all went out for an excellent lunch at the local Harvey House.

He brought the latest version of AED along, and showed me all the new features.  AED now has keyboard macros!  They are user definable, but he has predefined quite a few for you.  If you type two escapes in a row, the top 18 lines of the screen are used to display a menu of all the currently defined escape macros.  Escape followed by some other character inserts the corresponding text string at the current cursor position.

There is a utility program on the disk for use in defining your own macro strings, and it is very easy to use.  In fact, you use AED editing to modify simple DATA statements within the utility itself. When you are through with your changes, the utility modifies the macro table within AED and on the disk.

Again I say, if you are not fully satisfied with your current stable of Applesoft programming aids, you owe it to yourself to buy AED.  It will save you countless hours of frustrating retyping as you create and edit and restructure and debug and modify Applesoft programs.

==============================================================================
DOCUMENT :AAL-8205:Articles:NewOpcodes.txt
==============================================================================

Implementing New Opcodes Using 'BRK'.......Bob Sander-Cederlof

If you have the Autostart ROM, you can control what happens when a BRK instruction is executed. If you do nothing, a BRK will cause entry into the Apple Monitor, and the register contents will be displayed. But (if you have the Autostart Monitor) by a small amount of programming you can make the BRK do marvelous things.

Like simulate neat instructions from the 6809, which are not in the 6502. I am thinking particularly of the LEAX instruction, which loads the effective address into a 16-bit register; of the BSR, which enters a subroutine like JSR, but with a relative address; and BRA, which is a relatively addressed JMP. With these three instructions you can write position-independent programs (programs that execute properly without any modification regardless of where they are loaded in memory).

I am thinking of these because of an article by A. Sato in "Lab Letters" (a publication of ESD Laboratories in Tokyo, JAPAN) Volume 6 No. 1, pages 91-93. It is all written in Japanese (see example below), but I think I deciphered what he is saying.

When a BRK instruction is executed, the program is interrupted as though a Non-Maskable Interrupt (NMI) occurred. The B bit in the status register is set, so the Apple can tell that the interrupt was caused by BRK rather than some external event. After making this determination, the Autostart Monitor performs a "JMP ($3F0)" instruction. This means that you can get control by placing the address of your own program into $3F0 and $3F1. The monitor initialization process puts the address $FA59 there.

By the time the monitor branches to the BRK processor (its own or yours) all the registers have been saved. The address of the BRK instruction plus 2 (PC) has been saved at $3A and $3B; the registers A, X, Y, P (status), and S (stack pointer) have been saved in $45 through $49, respectively.

In the program below, lines 1180-1230 will set up the BRK-vector at $3F0 and $3F1 to point to your own BRK processor. Lines 1250-1320 back up the PC value by one, to point at the byte immediately following the BRK instruction. At this point I can decide what to do about the BRK.

Since I want to simulate the operation of LEAX, BSR, and BRA, I will use the BRK instruction to introduce a pseudo instruction of three bytes. I decided to copy A. Sato on this. LEAX is a BRK instruction followed by LDX from an absolute address. This is $AE in hexadecimal, followed by a 16-bit value representing a relative address. BSR is

BRK followed by a JSR instruction ($20) and a relative address; BRA is BRK followed by a JMP instruction ($4C) and a relative address.

Looking back at the program, lines 1310 and 1320 store the address of the secondary opcode byte into PNTR and PNTR+1. These two bytes are inside an instruction at line 1760. I didn't want to use any page-zero space, so I had to resort to this kind of self-modifying code. While we are here, lines 1750-1780 pick up the byte whose address is in PNTR. Lines 1710-1740 increment PNTR. If we call GET.THIS.BYTE, it just picks up the byte currently pointed at. If we call GET.NEXT.BYTE, it increments the pointer and gets the next byte.

Lines 1330-1370 pick up the three bytes which follow the BRK. The opcode byte is saved in the Y-register. Lines 1380-1450 compute the effective address, by adding the actual address of the instruction to the relative address inside the instruction.

Lines 1470-1540 classify the opcode; if it is one of the three we have implemented, it branches to the appropriate code. If not, it jumps back into the monitor and processes the BRK in the normal monitor way.

Lines 1560-1690 implement the three opcodes BSR, BRA, and LEAX.

================================================================
DOCUMENT :AAL-8205:Articles:Printers.4Sale.txt
================================================================

Good Price on the NEC printers

I can ship you an NEC PC-8023A-C dot matrix printer for only $595. I believe the normal list price is $795, but mail order prices are generally less. I also have the Grappler interface card and cable, configured for the NEC printer, for only $150 (normally $175, I think). And if you want both printer and interface at the same time, the combined price is only $695.

I have two of these printers, and like them better than my Epson MX-80. Why? Faster: 100cps instead of 80cps. Fully equipped: standard features include graphics, tractor feed, and friction feed. Handier: the friction feed is just like a typewriter, platen and all; and option switches, should you wish to change them, are accessible without removing any screws. I run one of them with an Epson parallel interface, and the other with the Grappler.

If you would rather have a Spinwriter (that is what this newsletter is printed on), call me for prices.

Vinyl Diskette Pages for your S-C Assembler Binder

I am having 1000 special pages manufactured. They will fit the binder that comes with the Macro Assembler, and will hold one diskette each and a 3x5 index card. For $6 I'll send you ten of them. For $12 I'll send them in a binder. For $36 you can have a binder with ten blank diskettes in vinyl pages. The binder is also just right for storing back issues of AAL.

The Best Book So Far for Beginners

Roger Wagner's book for beginners wanting to learn assembly language programming is now out, at $19.95. (My price is only $18.) Called "Assembly Lines: The Book", it began as simply a reprint of the series Roger writes for Softalk Magazine. But there is a lot more material in the book, and 100 pages of Appendices. Appendix B, 70 pages, is a very lucid description of every 6502 opcode. If you rank yourself as a beginning assembly language programmer, this book will be a tremendous help.

```
==============================================================================
DOCUMENT :AAL-8205:Articles:RWTS.Caller.txt
==============================================================================
```

RWTS CALLER ...................................by Bill Morgan

Here is a routine to directly call RWTS (Read/Write Track/Sector), the subroutine in DOS that actually reads fro or writes to the disk. Many programs use a routine like this to handle disk I/O, without all the time-consuming overhead of the DOS file manager.

All you need to do to use RWTS directly is to place certain information into an Input/Output control Block (IOB), and tell RWTS where the IOB is. Following is an explanation of the IOB (the addresses are those of DOS's own IOB; you can use it yourself, or build your own wherever is convenient):

```
Address        Description

$B7E8      Table type, always $01
$B7E9      Slot number times 16, usually $60
$B7EA      Drive number, $01 or $02
$B7EB      Volume number expected, $00 matches anything
$B7EC      Track number, $00 through $22
$B7ED      Sector number, $00 through $0F
$B7EE-F    Address of Device Characteristics Table,
           $B7FB for DOS's own DCT
$B7F0-1    Address of buffer, wherever you want
$B7F2      Not used
$B7F3      Byte count if partial sector, $00 normally
$B7F4      Command     $00 = SEEK
                       $01 = READ
                       $02 = WRITE
                       $04 = FORMAT
$B7F5      Error Code  $00 = No errors
                       $08 = Error in initialization
                       $10 = Write protect error
                       $20 = Volume mismatch
                       $40 = Drive error
$B7F6      Last volume number
$B7F7      Last slot number
$B7F8      Last drive number
```

The Device Characteristics Table (whose address is at $B7EE,EF) is a four-byte block containing information about the disk drive. For a standard Apple Disk II this block always contains $00 01 EF D8.

For our purposes, the most important items in the IOB are track, sector, buffer address, and command. By manipulating these, you can read any sector of the disk into any area of memory. All you need to do is set up the IOB, load the A- and Y-registers with the address of the IOB, and JSR $3D9. And if you decide to use the file manager's IOB, you can even set up the A- and Y-registers by a simple JSR $3E3.

RWTS will read the track and sector you choose into your 256-byte buffer. If there was a disk error, RWTS will return with the carry bit set and an error code stored in the IOB. It is then up to the user's program to decide what to do about the error. If there was no error, carry will be clear.

This month we'll set up a short program using the RWTS Caller to read an entire track of the disk into 16 consecutive pages of memory. DOS stores information on a track starting at sector $0F and working back to sector $00, so we must read a sector into the buffer, decrement the sector in the IOB, and increment the buffer pointer.

RWTS.CALLER:

Lines 1680-1850 set up the IOB, transferring values from the program variables.

Lines 1870-1890 load the address of the IOB and call RWTS.

Lines 1900-1910 are necessary to avoid confusing the system monitor. (RWTS and the monitor both use location $48.)

Lines 1960-2030 ring a warning if a disk error occurred, and display the error code.

TRACK READ:

Lines 1260-1350 initialize the variables and call input routines.

Lines 1370-1440 read the sectors from $0f through $00 into the buffer. Line 1410 will end the program if an error occurred.

Line 1460 will become a display routine. (Or, whatever processing you want to do on the buffer.)

Lines 1500-1650 will become input routines; right now they just set the track, buffer and command variables.

CAUTIONS:

1) These routines have very little error-checking. It is very easy to make a trivial error and lose information from a diskette. Always test an RWTS-calling program on a diskette yhou don't care about.

2) If you store information on a blank area of a diskette using these techniques, DOS doesn't know you have taken some space. Unless you modify the VTOC to show that sectors are used, DOS can overwrite your data. (What's a VTOC?, you say. Volume Table of Contents. We'll go into that another time.)

There is more about RWTS on pages 94-98 of Apple DOS Manual, and a goldmine of information in Beneath Apple DOS, by Don Worth and Pieter Lechner. (Quality Software, 1981.)

==============================================================================
DOCUMENT :AAL-8205:Articles:SCMacro.patches.txt
==============================================================================

Some Patches for the S-C Macro Assembler....Bob Sander-Cederlof

1. Loading the Language Card version: When you type "EXEC LOAD LCASM", the language card is loaded with a copy of the monitor from the Apple mother board, and with the file S-C.ASM.MACRO.LC. The EXEC file also makes a small modification to the memory image depending on which language you have on the mother board.

If you have serial number M-5275 or earlier, the EXEC file does not do the final step of turning on the Assembler. I expected you to type the DOS command "INT" (if Applesoft is on the mother board) or "FP" (if Integer BASIC is on the mother board) to enter the Assembler.

You can make a minor change to the EXEC file to make it automatically turn on the assembler after loading. The next to the last line of the EXEC file is now "C082"; change it to "C080" for automatic turn-on.

Here is a step-by-step procedure for the change. Try it on a COPY of the master disk, in case you make a mistake.

1. Get into the S-C Macro Assembler, either regular of language card version, it doesn't matter which).

2. Type "AUTO". The Assembler will print ":1000 " and wait for input.

3. Type five (5) backspaces (left arrow) to position the cursor right after the prompt.

4. Type "EXEC LOAD LCASM", and the Assembler will load the EXEC file into memory. You will see a list of line numbers on the screen.

5. Type five backspaces and the word "MANUAL" to turn off the auto-line-number mode.

6. LIST the lines (type "LIST").

7. See line 1090? It should be "C082". Type "1090 C080" to change it.

8. Type "TEXT LOAD LCASM" to save the modified version.

9. That's all there is to it!

By the way, Bob Potts (from the Bank Of Louisville) was here last week. He brought along a Corvus 5-meg drive, so we put the Language Card version of the Assembler on it. For some reason which we can't explain, the EXEC file hangs up after the BLOAD (but only if the language card has not been loaded since power up). We changed the

EXEC file slightly, and it always worked. Instead of doing the BLOAD while in the monitor, we did it from Applesoft. Here is the new version of the file:

```
REM LOAD S-C MACRO ASSEMBLER
REM INTO THE LANGUAGE CARD
CALL-151
C081 C081
F800<F800.FFFFM
BLOAD S-C.ASM.MACRO.LC
300:A9 4C CD 00 E0 F0 12 8D 00 E0 A9 00 8D 01 E0 A9 D0 8D 02 E0 A9 CB
8D D1 03 60
300G
C080
3D3G
```

You may also want to change the HELLO file to include an option to EXEC LOAD LCASM automatically.

2. If you have a Language Card, and your Assembler has a serial number of about M-5030 or earlier, the memory limits are not set up properly in all cases.

Check your copy of S-C.ASM.MACRO.LC by loading it into the language card and typing "$D2C6" from inside the assembler. If you don't have $A0 there, then you need to install this patch:

1. Type in these monitor commands:

```
:$D2C6:A0 0C
:$D2C8:20 1E D3 A9 00 91 58 85
:$D2D0:D9 AD 00 E0 C9 4C F0 08
```

2. Type "BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F".

3. If you have serial number M-5287 or earlier, a more difficult to apply patch is needed to correct a problem in printing the symbol table. If you are using the .TI directive, and if you have several lines of local labels in the symbol table listing, and if the page break comes between two such lines, the listing is messed up in a disastrous way.

To fix the S-C.ASM.MACRO, do the following (very carefully):

1. Get into the assembler by typing "BRUN S-C.ASM.MACRO".

2. Type the following monitor commands:

```
:$2AF<26AF.26D5M
:$26B1<2AF.2D5M
:$26AF:84 2F
:$26C1:CA
```

3. Type "BSAVE S-C.ASM.MACRO,A$1000,L$21D3"

To repair the language card version, do the following:

1. EXEC LOAD LCASM, and get into the assembler by typing INT (unless you already made the change to the EXEC file noted above).

2. Type the following monitor commands:

```
:$C083 C083
:$2AF<E7FB.E821M
:$E7FD<2AF.2D5M
:$E7FB:84 2F
:$E80D:16
```

3. Type "BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F"

If these patches and my instructions seem too difficult, you can send me $2.50 and your S-C Macro Assembler diskette; I will update it with the new HELLO program, the new LOAD LCASM file, and the patched copies of the assembler.

==============================================================================
DOCUMENT :AAL-8205:Articles:Secret.RWTS.Clr.txt
==============================================================================

Secret RWTS Caller Inside DOS....................Bill Parker

I found a portion of code tucked away in DOS that will perform a RWTS for you, doing away with the necessity of finding a place to put a controlling subroutine, an IOB, etc.

As you know, RWTS (Read/Write Track and Sector) gives the programmer the ability to read a sector from any specified track and put it in a buffer in RAM.  It also allows the programmer to manipulate the buffer and write it back out to any specified track and sector on the disk.

In this 48K DOS routine, which happens to be the same for 3.3 as well as 3.2, all the programmer has to do is to plug in the track and sector desired and whether he wants to read it from disk to the buffer, or write it from the buffer to the disk.  (The buffer is a fixed 256-byte location beginning at $B4BB (46267).)  A simple CALL 45111 or a JSR $B037 will then perform the transfer.  (You must remember to restore the original Read/Write code back to "2" when you are finished, so that the system can write to the directory when it needs to).

Here is a disassembled and commented version of the routine, which (for lack of a better term) I have named "WRTDIR".  This should aid in the development of programs that need to examine or alter the contents of a disk.

This routine, which normally writes a directory sector to the disk from the buffer at $B4BB.B5BA (46267-46522), can be used as a general RWTS utility by plugging in:

| Value | Name | $Loc | Loc |
|---|---|---|---|
| Read/Write (1/2) | RW | $B041 | 45121 |
| Track No. ($0-$22) | TK | $B397 | 45975 |
| Sector No. ($0-$F) | SC | $B398 | 45976 |

Then call 45111 or JSR $B037 and set RW to 2 when done.

=============================================================================
DOCUMENT :AAL-8205:DOS3.3:A.BlkMov.Bnch.txt
=============================================================================

( DTC removed -- lots of garbage characters )

```
==============================================================================
DOCUMENT :AAL-8205:DOS3.3:S.BlkMovBench.txt
==============================================================================

 1000  ********************************
 1010  *                              *
 1020  *  BENCHMARKING BLOCK MOVES    *
 1030  *                              *
 1040  *  BY WILLIAM R. SAVOIE  3/82 *
 1050  *   LIMERICK TRAINING CENTER  *
 1060  *  C/O GENERAL PHYSICS CORP.  *
 1070  *  341 LONGVIEW RD., LINFIELD *
 1080  *   PENNSYLVANIA   ZIP 19468  *
 1090  ********************************
 1100
 1110
 1120  *--------------------------------*
 1130  * APPLE II PAGE ZERO MEMORY USE *
 1140  *--------------------------------*
 1150
 1160  A1L    .EQ $3C      MONITOR
 1170  A1H    .EQ $3D       USE
 1180  A2L    .EQ $3E       FOR
 1190  A2H    .EQ $3F       BLOCK
 1200  A3L    .EQ $40       MOVE
 1210  A3H    .EQ $41
 1220  A4L    .EQ $42
 1230  A4H    .EQ $43
 1240  FACMO  .EQ $A0      FP REGISTER
 1250  FACLO  .EQ $A1      FP REGISTER
 1260
 1270  *--------------------------------*
 1280  * OTHER APPLE II MEMORY MAPPING *
 1290  *--------------------------------*
 1300
 1310  BLTU   .EQ $D39B    BLOCK TRANSFER
 1320  FRMNUM .EQ $DD67    FORMULA=>NUM
 1330  COMA   .EQ $DEBE    CHECK COMA
 1340  AYINT  .EQ $E10C    MAKE INTEGER
 1350  PRNTX  .EQ $F944    PRINT X
 1360  NXTA1  .EQ $FCBA    INCR POINTER
 1370  PRBYTE .EQ $FDDA    PRINT A
 1380  MOVE   .EQ $FE2C    MONITOR MOVE
 1390
 1400
 1410  *--------------------------------*
 1430       .OR $3000
 1440       .TF B.BLOCK MOVE BENCHMARKS
 1450  *--------------------------------*
 1460
 1470  *  THIS CODE ALLOWS SIMPLE ENTRY WITHOUT THE & COMMAND
 1480  BEGIN  JMP MONITOR.MOVE
 1490         JMP APPLESOFT.MOVE
```

```
1500        JMP QUICK.MOVE
1510        JMP DUMP     HEX OUTPUT
1520        JMP FILL     LABLE MEMORY
1530
1540 * TO HELP A HEX CONVERTER
1550 CONVERT
1560        JSR GETVAR   GET VARIABLE
1570        JSR PRNTX    HI BYTE OUT
1580        LDA FACLO    GET LOW BYTE
1590        JMP PRBYTE   HEX OUTPUT
1600        .PG
1610 * THIS CODE FILLS MEMORY WITH IT'S OWN ADDRESS
1620 * WHICH IS VERY USEFULL FOR CHECKING BLOCK MOVES
1630 FILL   JSR GM       GET VARIABLES
1640 .01    LDY #$01     TWO BYTE OFFSET
1650        LDA A1L      GET LOW BYTE
1660        STA (A1L),Y  WRITE ADDRESS LO
1670        DEY          MOVE LEFT
1680        LDA A1H      GET HI ADDRES
1690        STA (A1L),Y  WRITE TO MEMORY
1700        JSR NXTA1    INCREMENT PTR
1710        JSR NXTA1    TWICE
1720        BCC .01      GO TELL DONE
1730        RTS
1740
1750 * A UTILITY DUMP T0 SEE MEMORY FROM BASIC
1760 DUMP   JSR GM       GET VARS
1770        LDA $C010    CLEAR STROBE
1780 .01    LDA $C000    GET KEY IF ONE
1790        BPL .03      GO DUMP HEX
1800        LDA $C010    CLEAR STROBE
1810 .02    LDA $C000    READ KEY AGAIN
1820        BPL .02      WAIT FOR KEY
1830        CMP #$8D     'RETURN' KEY?
1840        BEQ .04      EXIT
1850        LDA $C010    CLEAR STROBE
1860 .03    JSR $FDA3    8 HEX OUT
1870        LDA A2L      END LOW
1880        CMP $A1      DESIRED LOW
1890        LDA A2H      HI TOO
1900        SBC $A0      ENOUGH?
1910        BCC .01      GO TELL DONE
1920 .04    RTS
1930
1940 * GET VARIABLE FROM BASIC
1950 * MUST BE <65357 OR SYNTAX ERR
1960 * PLACE IN REGISTERS X AND A
1970
1980 GETVAR JSR COMA     MUST SEE COMA
1990        JSR FRMNUM   GET NUMBER
2000        JSR AYINT    MAKE INTEGER
2010        LDX FACMO    HI BYTE
2020        LDA FACLO    LOW BYTE
2030        RTS
```

```
* GET BASIC VARIABLES INTO THE
* MONITORS WORK REGISTERS USED BY
* COMMANDS M,V,G,L,S,T,-,+,..ETC

GM     JSR GETVAR   BLOCK START
       STA A1L      LOW BYTE
       STX A1H      HI BYTE
       JSR GETVAR   BLOCK END
       STA A2L      LOW
       STX A2H      HI BYTE
       JSR GETVAR   DEST. START
       STA A4L      LOW
       STX A4H      HI
       RTS
       .PG
*-------------------------------*
* THE OLD MONITOR MOVE          *
* MOVE BLOCK OF MEMORY UP/DOWN  *
*-------------------------------*

MONITOR.MOVE
       JSR GM       SET UP MOVE
       LDA A1L      START LOW
       CMP A4L      END LOW
       LDA A1H      START HI
       SBC A4H      WHICH BIGGER?
       BCS MOVEDN   GO DOWN IN MEM

MOVEUP LDY #$00     CLEAR INDEX
.01    LDA (A2L),Y    GET DATA
       STA (A4L),Y    PUT DATA
       LDA A4L      GET INDEX
       BNE .02      PAGE CROSS?
       DEC A4H      HI BYTE
.02    DEC A4L      LOW BYTE
       LDA A2L
       CMP A1L      END YET?
       LDA A2H
       SBC A1H
       LDA A2L
       BNE .03      PAGE CROSS?
       DEC A2H      HI BYTE
.03    DEC A2L      LOW BYTE
       BCS .01      GO TELL DONE
       RTS

MOVEDN LDY #$00     CLEAR INDEX
       JMP MOVE     MONITOR MOVE


*------------------------------*
* AND ALONG CAME THE PEOPLE AT *
```

```
2580  * MICROSOFT WITH THEIR MOVE    *
2590  *-----------------------------*
2600
2610  APPLESOFT.MOVE
2620         JSR GETVAR   HI ADDRESS OF BLOCK TO MOVE
2630         STA $96      LOW
2640         STX $97      HI BYTE
2650         JSR GETVAR   BLOCK END
2660         PHA          SAVE TELL LAST
2670         TXA          NEED 9B,9C
2680         PHA          TO GETVARS
2690         JSR GETVAR   HI ADDRESS OF DISTINATION
2700         STA $94      LO&HI BYTES
2710         STX $95
2720
2730  * WE USED $9B,9C TO GET THE TWO BYTE FP VALUE
2740         PLA          END OF BLOCK
2750         STA $9C      HI BYTE
2760         PLA
2770         STA $9B      LOW BYTE TOO
2780         SEC          SUBTRACT COMMING
2790         JMP BLTU     A.MOVE
2800         .PG
2810  *-------------------*
2820  * MOVING IN RAM CAN *
2830  * BE EVEN FASTER    *
2840  *-------------------*
2850
2860  QUICK.MOVE
2870         JSR GETVAR   GET START
2880         STA .01+1    LOW BYTE
2890         STA .06+1    COPY HERE TOO
2900         STX .01+2    HI
2910         JSR GETVAR   DESTINATION
2920         STA .02+1    LO
2930         STA .07+1    COPY HERE TOO
2940         STX .02+2    HI BYTE
2950         JSR GETVAR   END ADDRESS
2960         TXA          SET TEST FOR
2970         BEQ .05       MOVE<256?
2980
2990  * X=PAGE NUMBERS TO MOVE
3000         LDY #$00     INDEX=0
3010  .01    LDA $4800,Y  SOURCE
3020  .02    STA $4000,Y  DESTINATION
3030         INY          NEXT BYTE
3040         BNE .01      SMALL MOVE
3050  .03    INC .01+2    HI SOURCE
3060  .04    INC .02+2    HI DESTINATION
3070         DEX          DONE?
3080         BNE .01      Y=0 SO MOVE PAGE
3090
3100  * SET UP REMAINING MOVE
3110  .05    LDY $A1      LOW BYTE LENGTH
```

```
3120          BEQ .08       GO IF NONE
3130          LDA .01+2     COPY HI BYTE
3140          STA .06+2     FOR SOURCE
3150          LDA .02+2      AND
3160          STA .07+2      DESTINATION
3170
3180   * NOW WITH X=0 START MOVING LOW BYTE OF LENGTH
3185   * (Y) = REMAINING BYTES TO MOVE
3190   .06    LDA $4800,X  SOURCE
3200   .07    STA $4000,X  DESTINATION
3210          INX           NEXT
3220          DEY           MOVE ENOUGH?
3230          BNE .06       GO TELL DONE
3240   .08    RTS
3250          .LIST OFF
```

===========================================================================
DOCUMENT :AAL-8205:DOS3.3:S.BRANCH.MACROS.txt
===========================================================================

```
 1000  *  MACRO BRANCH LIBRARY
 1010  *  BY R.F. 0'BRIEN
 1020  *--------------------------------
 1030  *   >BLT P1 (,P2)   BRANCH IF (A) < P1...TO P2
 1040         .MA BLT
 1050         .DO ]#>1
 1060         CMP ]1
 1070         BCC ]2
 1080         .ELSE
 1090         BCC ]1
 1100         .FIN
 1110         .EM
 1120  *--------------------------------
 1130  *   >BLE P1 (,P2)   BRANCH IF (A)<=P1...TO P2
 1140         .MA BLE
 1150         .DO ]#>1
 1160         CMP ]1
 1170         BEQ ]2
 1180         BCC ]2
 1190         .ELSE
 1200         BEQ ]1
 1210         BCC ]1
 1220         .FIN
 1230         .EM
 1240  *--------------------------------
 1250  *   >BGE P1 (,P2)   BRANCH IF (A)>=P1...TO P2
 1260         .MA BGE
 1270         .DO ]#>1
 1280         CMP ]1
 1290         BCS ]2
 1300         .ELSE
 1310         BCS ]1
 1320         .FIN
 1330         .EM
 1340  *--------------------------------
 1350  *   >BGT P1 (,P2)   BRANCH IF (A)>P1...TO P2
 1360         .MA BGT
 1370         .DO ]#>1
 1380         CMP ]1
 1390         BEQ :1
 1400         BCS ]2
 1410  :1
 1420         .ELSE
 1430         BEQ :1
 1440         BCS ]1
 1450  :1
 1460         .FIN
 1470         .EM
 1480  *--------------------------------
```

```
1490  *   >BRA P1         BRANCH ALWAYS...TO P1
1500         .MA BRA
1510         CLV
1520         BVC ]1
1530         .EM
1540  *--------------------------------
1550  *   >BEQ P1 (,P2)   BRANCH IF (A)=P1...TO P2
1560         .MA BEQ
1570         .DO ]#>1
1580         CMP ]1
1590         BEQ ]2
1600         .ELSE
1610         BEQ ]1
1620         .FIN
1630         .EM
1640  *--------------------------------
1650  *   >BNE P1 (,P2)   BRANCH IF (A)<>P1...TO P2
1660         .MA BNE
1670         .DO ]#>1
1680         CMP ]1
1690         BNE ]2
1700         .ELSE
1710         BNE ]1
1720         .FIN
1730         .EM
1740  *--------------------------------
1750  *   >JMP P1,P2      BRANCH ALWAYS TO P1 BY
1760  *                   BRANCHING TO P2 (SEE ARTICLE)
1770         .MA JMP
1780         CLV
1790         BVC :1
1800  ]2     CLV
1810         BVC ]1
1820  :1
1830         .EM
```

```
==================================================================
DOCUMENT :AAL-8205:DOS3.3:S.GAME.BUTTON.txt
==================================================================

 1000  *---------------------------------
 1010  *      GAME BUTTON SUBROUTINES
 1020  *---------------------------------
 1030  GAME.BUTTON .EQ $C061  BASE ADDRESS
 1040  *---------------------------------
 1050         .OR $800
 1060  *---------------------------------
 1070  GAME.BUTTON.INITIALIZE
 1080         LDA #0
 1090         STA GB.STAT
 1100         RTS
 1110  *---------------------------------
 1120  GAME.BUTTON.INSTALLED
 1130         JSR GET.BUTTON.STATUS
 1140         ORA GB.STAT  SET BITS OF ANY BUTTONS
 1150         STA GB.STAT  PLUGGED IN AND NOT PUSHED
 1160         RTS
 1170  *---------------------------------
 1180  GAME.BUTTON.PUSHED
 1190         JSR GET.BUTTON.STATUS
 1200         EOR GB.STAT  MASK OUT BUTTONS WHICH
 1210         AND GB.STAT    ARE NOT PLUGGED IN
 1220         STA GB.PUSH
 1230         RTS
 1240  *---------------------------------
 1250  GET.BUTTON.STATUS
 1260         LDX #2
 1270  .1     LDA GAME.BUTTON,X
 1280         EOR #$80     INVERT SENSE
 1290         ASL          INTO CARRY BIT
 1300  .2     ROR GB.PUSH
 1310         DEX          NEXT BUTTON
 1320         BPL .1
 1330         LDA GB.PUSH
 1340         AND #$E0     CLEAR EXTRANEOUS BITS
 1350         RTS
 1360  *---------------------------------
 1370  GB.STAT .BS 1
 1380  GB.PUSH .BS 1
 1390  *---------------------------------
 1400  MON.CV     .EQ $25
 1410  MON.HOME   .EQ $FC58
 1420  MON.VTAB   .EQ $FC22
 1430  MON.CLREOL .EQ $FC9C
 1440  MON.COUT   .EQ $FDED
 1450  TEST.MASK  .EQ $00
 1460  *---------------------------------
 1470  TEST   JSR MON.HOME
 1480         JSR GAME.BUTTON.INITIALIZE
```

```
1490  .1     LDA #0
1500         STA MON.CV
1510         JSR MON.VTAB
1520         JSR GAME.BUTTON.INSTALLED
1530         JSR GAME.BUTTON.PUSHED
1540         LDA #$84
1550         STA TEST.MASK
1560  .2     LDA TEST.MASK
1570         AND GB.PUSH
1580         BNE .3       PUSHED
1590         LDA TEST.MASK
1600         AND GB.STAT
1610         BNE .4       NOT PUSHED
1620         LDY #QTGONE-QTS   NOT INSTALLED
1630         .HS 2C
1640  .3     LDY #QTPUSHED-QTS
1650         .HS 2C
1660  .4     LDY #QTNOTPSH-QTS
1670         JSR MSGOUT
1680         LSR TEST.MASK
1690         BCC .2
1700         LDA $C000
1710         BPL .1
1720         STA $C010
1730         RTS
1740         BCS .1       ...ALWAYS
1750  *--------------------------------
1760  MSGOUT LDA QTS,Y
1770         PHA
1780         ORA #$80
1790         JSR MON.COUT
1800         INY
1810         PLA
1820         BPL MSGOUT
1830         JSR MON.CLREOL
1840         LDA #$8D
1850         JMP MON.COUT
1860  *--------------------------------
1870  QTS
1880  QTPUSHED   .AT /PUSHED/
1890  QTNOTPSH   .AT /NOT PUSHED/
1900  QTGONE     .AT /NOT INSTALLED/
1910  *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8205:DOS3.3:S.RecurMac.2.txt
==========================================================================

 1000  *--------------------------------
 1010  *      LEE MEADOR'S SECOND RECURSIVE MACRO
 1020  *--------------------------------
 1030         .MA DB
 1040         .DO ]1<2
 1050         .DA ]2
 1060         .ELSE
 1070         >DB ]1/2,]2
 1080         >DB ]1+1/2,]2
 1090         .FIN
 1100         .EM
 1110  *--------------------------------
 1120         >DB 3,#0
```

```
=============================================================================
DOCUMENT :AAL-8205:DOS3.3:S.TRACK.READ.txt
=============================================================================

 1000  *SAVE TRACK READ
 1010  *---------------------------------
 1020  SLOT    .EQ $00     $60 OR $70
 1030  DRIVE   .EQ $01     1 OR 2
 1040  VOLUME  .EQ $02     0 = DON'T CARE
 1050  TRACK   .EQ $03     $00 TO $22
 1060  SECTOR  .EQ $04     $00 TO $0F
 1070  BUFFER  .EQ $05,06
 1080  COMMAND .EQ $07     1 = READ, 2 = WRITE
 1090  PREG    .EQ $48
 1100  *
 1110  RWTS    .EQ $3D9
 1120  *
 1130  IOB         .EQ $B7E8    DOS'S OWN IOB
 1140  IOB.SLOT    .EQ $B7E9
 1150  IOB.DRIVE   .EQ $B7EA
 1160  IOB.VOLUME  .EQ $B7EB
 1170  IOB.TRACK   .EQ $B7EC
 1180  IOB.SECTOR  .EQ $B7ED
 1190  IOB.BUFFER  .EQ $B7F0,F1
 1200  IOB.COMMAND .EQ $B7F4
 1210  IOB.ERROR   .EQ $B7F5
 1220  *
 1230  PRBYTE .EQ $FDDA
 1240  COUT   .EQ $FDED
 1250  *---------------------------------
 1260  SETUP
 1270         LDA #$60
 1280         STA SLOT     SLOT 6
 1290         LDA #$01
 1300         STA DRIVE    DRIVE 1
 1310         LDA #$00
 1320         STA VOLUME   ANY VOLUME
 1330         JSR GET.TRACK
 1340         JSR GET.BUFFER
 1350         JSR GET.COMMAND
 1360  *---------------------------------
 1370  READ.TRACK
 1380         LDA #$0F     START AT SECTOR $0F
 1390         STA SECTOR
 1400  .1     JSR RWTS.CALLER   READ ONE SECTOR
 1410         BCS EXIT     EXIT IF ERROR
 1420         INC BUFFER+1 NEXT BUFFER PAGE
 1430         DEC SECTOR   NEXT SECTOR
 1440         BPL .1       NOT DONE, READ NEXT SECTOR
 1450  *---------------------------------
 1460  DISPLAY
 1470  *---------------------------------
 1480  EXIT   RTS
```

```
1490  *--------------------------------
1500  GET.TRACK
1510         LDA #$11
1520         STA TRACK    TRACK $11 (DIRECTORY)
1530         RTS
1540  *--------------------------------
1550  GET.BUFFER
1560         LDA #0
1570         STA BUFFER   BUFFER AT $4000
1580         LDA #$40
1590         STA BUFFER+1
1600         RTS
1605         .PG
1610  *--------------------------------
1620  GET.COMMAND
1630         LDA #1
1640         STA COMMAND  READ
1650         RTS
1660  *--------------------------------
1670  RWTS.CALLER
1680         LDA SLOT     TRANSFER
1690         STA IOB.SLOT  VALUES
1700         LDA DRIVE      INTO
1710         STA IOB.DRIVE   IOB
1720         LDA VOLUME
1730         STA IOB.VOLUME
1740         LDA TRACK
1750         STA IOB.TRACK
1760         LDA SECTOR
1770         STA IOB.SECTOR
1780         LDA COMMAND
1790         STA IOB.COMMAND
1800         LDA BUFFER
1810         STA IOB.BUFFER
1820         LDA BUFFER+1
1830         STA IOB.BUFFER+1
1840         LDA #$00
1850         STA IOB.ERROR
1860  *--------------------------------
1870         LDY #IOB     LOAD IOB
1880         LDA /IOB     ADDRESS
1890         JSR RWTS     CALL RWTS
1900         LDA #$00
1910         STA PREG     SOOTHE MONITOR
1920         BCS ERROR.HANDLER
1930         RTS
1940  *--------------------------------
1950  ERROR.HANDLER
1960         LDA #$87     BELL
1970         JSR COUT     RING
1980         JSR COUT        ING
1990         JSR COUT          ING
2000         LDA IOB.ERROR
2010         JSR PRBYTE   DISPLAY ERROR CODE
```

```
2020        SEC         EXIT WITH CARRY SET
2030        RTS
```

```
==================================================================
DOCUMENT :AAL-8205:DOS3.3:S.WRTDIR.txt
==================================================================

 1000           .OR $B037
 1010           .TF B.WRTDIR
 1020  *--------------------------------
 1030  BUFSTHI  .EQ $AAC6
 1040  BUFSTLO  .EQ $AAC5
 1050  CALLRWTS .EQ $B052
 1060  IOBBUF   .EQ $B7F0
 1070  RW       .EQ 2
 1080  SC       .EQ $B398
 1090  TK       .EQ $B397
 1100  *--------------------------------
 1110  WRTDIR   JSR SETBUFAD
 1120           LDX TK
 1130           LDY SC
 1140           LDA #RW
 1150           JMP CALLRWTS
 1160  *--------------------------------
 1170  SETBUFAD LDA BUFSTLO     PUT BUFFER'S
 1180           STA IOBBUF      STARTING ADDRESS IN
 1190           LDA BUFSTHI     INPUT OUTPUT BLOCK
 1200           STA IOBBUF+1
 1210           RTS
```

```
==============================================================================
DOCUMENT :AAL-8206:Articles:Auto.Catalog.txt
==============================================================================
```

Automatic CATALOG for S-C Macro Assembler..........Bill Morgan

Being a thoroughly lazy (and fumblefingered) typist, I have been itching for an automatic CATALOG command to go with the automatic LOAD in the S-C Macro Assembler. Well I finally have it; now loading a file is just esc-C, esc-I...IL. I chose esc-C for CATALOG because I never use the esc-ABCD cursor moves. If you do like those, esc-G and -H are available; right now they are like NOP's.

The Macro Assembler takes the character following an escape (@, A, B,..., L, M) and makes it an index into a jump table located from $1467-1482. Esc-C is at $146D in the table, esc-G is $1475, and esc-H is $1477.

The patch is only $28 bytes long, short enough to easily fit in page 3, but I decided to go ahead and create a spare page for patches by moving the symbol table up one page. This technique is mentioned on page 5-3 of the Macro Assembler manual.

To install the patch, first move the symbol table base up by changing location $101D from $32 to $33. Now insert the address of the patch into the jump table by changing locations $146D-6E from $65 FC to $FF 31 (or your location-1). Type "BLOAD PATCH", then "BSAVE ASM MACRO.MOD,A$1000,L$22FF", and there you have it.

===============================================================================
DOCUMENT :AAL-8206:Articles:BRK.Opcodes.txt
===============================================================================

Implementing New Opcodes Using 'BRK'.......Bob Sander-Cederlof

If you have the Autostart ROM, you can control what happens when a BRK instruction is executed. If you do nothing, a BRK will cause entry into the Apple Monitor, and the register contents will be displayed. But (if you have the Autostart Monitor) by a small amount of programming you can make the BRK do marvelous things.

Like simulate neat instructions from the 6809, which are not in the 6502, for example. I am thinking particularly of the LEAX instruction, which loads the effective address into a 16-bit register; of BSR, which enters a subroutine like JSR, but with a relative address; and of BRA, which is a relatively addressed JMP. With these three instructions you can write position-independent programs (programs that execute properly without any modification regardless of where they are loaded in memory).

I am thinking of these because of an article by A. Sato in "Lab Letters" (a publication of ESD Laboratories in Tokyo, JAPAN) Volume 6 No. 1, pages 91-93. It is all written in Japanese (see example below), but I think I deciphered what he is saying.

When a BRK instruction is executed, the program is interrupted as though a Non-Maskable Interrupt (NMI) occurred. The B bit in the status register is set, so the Apple can tell that the interrupt was caused by BRK rather than some external event. After making this determination, the Autostart Monitor performs a "JMP ($3F0)" instruction. This means that you can get control by placing the address of your own program into $3F0 and $3F1. The monitor initialization process puts the address $FA59 there.

By the time the monitor branches to the BRK processor (its own or yours) all the registers have been saved. The address of the BRK instruction plus 2 (PC) has been saved at $3A and $3B; the registers A, X, Y, P (status), and S (stack pointer) have been saved in $45 through $49, respectively.

BRK Interceptor/Interpreter

In the program below, lines 1180-1230 will set up the BRK-vector at $3F0 and $3F1 to point to your own BRK processor. Lines 1250-1320 back up the PC value by one, to point at the byte immediately following the BRK instruction. At this point I can decide what to do about the BRK.

Since I want to simulate the operation of LEAX, BSR, and BRA, I will use the BRK instruction to introduce a pseudo instruction of three bytes. I decided to copy A. Sato on this. LEAX is a BRK instruction followed by LDX from an absolute address. This is $AE in hexadecimal, followed by a 16-bit value representing a relative address. BSR is BRK followed by a JSR instruction ($20) and a relative address; BRA is BRK followed by a JMP instruction ($4C) and a relative address.

Looking back at the program, lines 1310 and 1320 store the address of the secondary opcode byte into PNTR and PNTR+1. These two bytes are inside an instruction at line 1760. I didn't want to use any page-zero space, so I had to resort to this kind of self-modifying code. While we are here, lines 1750-1780 pick up the byte whose address is in PNTR. Lines 1710-1740 increment PNTR. If we call GET.THIS.BYTE, it just picks up the byte currently pointed at. If we call GET.NEXT.BYTE, it increments the pointer and gets the next byte.

Lines 1330-1370 pick up the three bytes which follow the BRK. The opcode byte is saved in the Y-register. Lines 1380-1450 compute the effective address, by adding the actual address of the instruction to the relative address inside the instruction.

Lines 1470-1540 classify the opcode; if it is one of the three we have implemented, it branches to the appropriate code. If not, it jumps back into the monitor and processes the BRK in the normal monitor way.

Opcode Implementation

Lines 1560-1780 implement the three opcodes BSR, BRA, and LEAX. BRA (Branch Always) is the easiest one. We have already computed the effective address and stored it in the address field of the JMP instruction at line 1620. All BRA does is restore the registers (line 1610), and JMP to the effective address.

BSR (Branch to Subroutine) is only slightly harder. We first have to push the return address on the stack, and then do a BRA. Lines 1560-1590 do the pushing.

LEA (Load Effective Address) is the hardest. Lines 1650-1690 do the work. First GET.NEXT.BYTE moves the address in PNTR,PNTR+1 to point at the first byte of the next instruction. That is so we can continue exectution. Then MON.RESTORE gets back the original contents of all the registers. THEN LDY and LDX pick up the effective address in the Y- and X-registers. The high byte of the effective address is in the X-register, and the Z- and N-bits in the status register reflect the value of this byte. If you wish, you could modify this to not change the status by inserting a PHP opcode after line 1660, and PLP after line 1680; then the status register would remain unchanged by the entire LEA process. Or you could reverse lines 1670 and 1680, so that the status reflected the low-order byte of the effective address.

Demonstration Using the New Opcodes

Lines 1800 and beyond are a demonstration of the use of the new opcodes. First I defined some macros for the new opcodes. I didn't have to do this, but it is convenient if you have a macro assembler. If you don't, you can use the BRK instruction on one line, followed by a LDX, JSR, or JMP instruction with a relative address on the next line.

My macros are defined in a nested fashion.  The BRK macro generates two lines:  BRK on the first line, and a second line consisting of the specified opcode and operand.  The LEA, BSR, and BRA macros call BRK to generate LDX, JSR, and JMP instructions after the BRK.  The operand field is a relative address, computed within the BRK macro.

The demonstration program will run anywhere in memory, as long as the BRK interpreter has been loaded and initialized.  You can test this by moving $871-89F to other places and running it.  What it does is print out the message in line 2090.

================================================================================
DOCUMENT :AAL-8206:Articles:BubbleSort.Demo.txt
================================================================================

Bubble Sort Demonstration Program.......Bob Sander-Cederlof

The following program implements one of the most inefficient methods of sorting a list of items ever invented. It is also a very specific implementation, not general at all. But it should be valuable to study if you are not already well-versed in sorting techniques. After execution, the bytes from $00 to $0F will be in ascending order.

==============================================================================
DOCUMENT :AAL-8206:Articles:DFX.Review.txt
==============================================================================

DOS File Exchange:  A Review.....................Bill Morgan

I've just been playing with a new program called DOS File Exchange (DFX), and it is wonderful.  Author Graeme Scott has provided a very useful tool for transferring any type of files through a modem, with full error-checking.  You can even chat at the keyboards while the transfer is going on!

The DFX program must be running on both computers, and one of them must be using an original (primary) disk of the program.  The program can be copied to produce a secondary disk; DFX will even send a copy of itself to a remote Apple, but the copy will be a secondary.

To transfer files, one user selects a "master" mode, so he will control both Apples.  He then chooses whether he will send or receive; the program then transmits the sending Apple's disk catalog to the receiver.  The master user selects the files wanted from the catalog and starts the transfer.  Both users are then free to chat, supervise the transfer in one of three display modes, or even leave the room.

At almost any time, you can switch back and forth between Function and Chat modes.  Function is used to select all control and menu choices; Chat sends all characters entered to the other Apple.

There are three display modes, called M(enu), U(tility), and G(raphic).  Menu shows choices, including the disk catalog when files are being chosen.  Utility displays the transmitted and received data streams, and allows more space for chatting.  Graphic displays the data being transferred on the Hi-res screen, so if you are receiving a picture you can watch it take shape.

The only drawbacks I've found are that DFX will only operate with a Hayes Micromodem II in slot 2 and the disk in slot 6, drive 1.

DFX is available from Arrow Micro Software, 11 Kingsford, Kanata Ont., K2K 1T5 Canada.

```
==================================================================
DOCUMENT :AAL-8206:Articles:Examiner.txt
==================================================================
```

Examiner...........................................Bill Morgan

Here is the program I like to use to examine memory; it displays an entire page on the screen in both hex and ASCII formats.  This makes the screen kind of crowded, but I particularly wanted a full page at a time.  A program like this is useful for inspecting the results of last month's TRACK READ program, studying the internal format of an Applesoft program, or just exploring inside your Apple.

Examiner uses the left and right arrow keys to decrement or increment the page being displayed.  You can also type "P" to allow entry of a page number in hex.  Notice that the number entered is rolled into the page number from the right.  Escape exits the program.

Lines 1180-1260 set things up to start with page zero.

Lines 1280-1390 display the index, then twelve bytes in hex format.

Lines 1410-1460 reset the indices to display the same twelve bytes in ASCII.

Lines 1480-1630 do the ASCII display, changing any inverse or flashing values to normal and substituting periods for control characters.

Lines 1700-1870 process the commands to change the page being displayed.

Lines 1890-2160 accept characters "0" through "F" and convert them into hex values, rolling the values into the page number to be displayed.

Lines 2180-2260 display the header "page=".

This is threatening to turn into a monthly column; what do you readers think of that idea?  Are these routines too trivial?  Too complicated? Do you have any questions about them?  About anything fairly basic? Drop me a line here at AAL and let me know what you think.  I'll look forward to hearing from you.

==============================================================================
DOCUMENT :AAL-8206:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 2 -- Issue 9                              June, 1982

In This Issue...



Advertising in AAL

Due to the increased costs of printing more than 1600 copies per month, and with the desire to limit the percentage of advertising pages to less than 30% each month, I have decided to raise the page rate again.

For the July 1982 issue the price will be $50 for a full page, $30 for a half page.  So-called "classified" ads, of up to forty words, will be $5.

==============================================================================
DOCUMENT :AAL-8206:Articles:Hint.txt
==============================================================================

Macro Hint.........................................Bob and Bill

For an easy semi-automatic SAVE, we use the following line in every program:

```
     1000 *HHHHHHSAVE filename
```

The six H's are control H's (backspaces), entered by holding the CTRL key down and typing OHOHOHOHOHOH.  (Control-O allows a following control character to be entered into a line.)  To save the source file, just type LIST 1000, esc-I, and copy over the line.  Make it a point to always have the SAVE in line 1000; it's much easier to remember.

==============================================================================
DOCUMENT :AAL-8206:Articles:My.Bell.txt
==============================================================================

My Own Little Bell......................Bob Sander-Cederlof

The other day I was working on my Apple at home, and the kids were trying to sleep in the same room. The program I was working on needed to indicate erroneous input by a bell, and I had to test it. Suddenly I realized how loud and sharp the Apple bell is!

With all that motivation, I threw together this little routine which makes a soft and pleasant tone to use for my own little bell. It generates fifty repetitions of a triple-toggle pattern, with time intervals selected for their harmonious character.

Lines 1070, 1170, and 1180 establish a loop equivalent to the Applesoft code:

```
FOR X = 50 TO 1 STEP -1: . . . : NEXT
```

In assembly language it frequently occurs that backwards running loop counts are easier to use than forward ones, and this is just such a case.

Examine lines 1080-1160, and you will see a pattern repeated three times. In each case I load A with a value, call MON.DELAY, and toggle the speaker. The value passed to MON.DELAY is first 14, then 10, and then 6. MON.DELAY is a subroutine in the Apple Monitor ROM which delays an amount of time depending on what value you pass in the A-register, according to the following formula:

```
# cycles delay = 2.5*N*N + 13.5*N +13
```

This includes the six cycles of the JSR used to call the subroutine. Each cycle is...well, the Apple clock is roughly 1.023 MHz...so a cycle is about .9775 microseconds long. The counts of 14, 10, and 6 give intervals between toggles of 630.5, 204, and 195 (including the overhead instructions in SC.BELL).

You can play with the values, and try creating your own variations. You might try adding a fourth toggle per loop, changing the number of loops, changing the delay counts, and so on. Have fun!

==============================================================================
DOCUMENT :AAL-8206:Articles:Search.ZP.txt
==============================================================================

Search for Page-Zero References.........Bob Sander-Cederlof

Many times I have wanted a utility which would list out all references to page-zero locations withing a program. For example, when I am trying to avoid conflicts with DOS or Applesoft, I need to know which ones they use and where.

The following little program hooks into the Apple Monitor through the control-Y user command. You type in the address range you want to search through, control-Y, and a carriage return. The Apple will disassemble only those instructions within the address range which reference page-zero locations.

Lines 1220-1280 set up the control-Y vector. When the monitor detects a control-Y command, it branches to $3F8. The JMP instruction there in turn branches to CTRL.Y at line 1320.

Line 1330 loads the first address of the range into PCL and PCH. If you did not type any range before the control-Y, the previous value will be used.

Lines 1340-1540 decide whether the instruction starting at the address in PCL,PCH references page-zero or not. All instructions which reference page-zero have opcodes of the form x1, x4, x5, or x6. All of the x1, x5, and x6 possiblities are valid; only 24, 84-C4, and E4 in the x4 column are valid.

Lines 1580 and 1590 call on a piece of the monitor L-command to disassemble the one instruction. This also updates PCL,PCH to point to the next opcode byte.

Lines 1600-1700 allow you to stop/start the listing by typing any key, to single-step the listing by pressing any two keys simultaneously, and to abort by typing RETURN.

Lines 1740-1780 are executed if the instruction does not reference page-zero. The call on pieces of the L-command to figure out the number of bytes in the instruction and update PCL,PCH accordingly.

Lines 1820-1870 check to see if the range you specified has been covered yet. If not, keep searching; if so, stop.

This kind of program should be in your tool-kit when you are debugging. Just don't lose it under all those other tools!

==============================================================================
DOCUMENT :AAL-8206:Articles:Shift.Key.Mod.txt
==============================================================================

Using the Shift-Key Mod.................Bob Sander-Cederlof

Have you heard of the "Shift-Key Mod"? By running a wire from the game connector to the right spot on the keyboard circuit, you can use software to tell whether or not the shift key is pressed. You can make your Apple keyboard almost normal!

Some word processors come with a convenient device which has a clip on one end of a wire, and a DIP socket-plug on the other. (I sell such a device for $15 without any software.) Apples with the piggy-back board below the keyboard can use the clip. If you don't have that kind of Apple, you need to solder a small wire to the bottom of either shift key, and clip onto that wire. Of course, you can run the wire all the way to the game connector and avoid the extra expense...I did it that way on my first Apple.

But what about software? All the mod does is bring the shift key into the game connector as PB2. You can read it with LDA $C063. If the value read is $00-7F, the shift key is being pressed; if $80-FF, the shift key is not pressed. You have to write a special keyboard input subroutine to convert letters to lower case ASCII codes if the shift key is not down.

Here is just such a subroutine! It is the one I use in my word processor (a product still being developed). Another routine sets up a cursor on the screen, and then calls READ.KEY.WITH.CASE to get the next keypress.

Lines 1140-1160 read the keyboard, and keep reading until you press a key other than the shift key. Once you press a key, the value at KEYBRD will be a code between $80 and $DF; the value is considered negative by the 6502, so execution continues at line 1170.

Lines 1170-1200 are an optional keyclick routine. In my word processor, a control-P turns the keyclicking on and off. I discovered that a very short "bell" sounds like a clicking keyboard, so that is what I use. The monitor bell subroutine toggles the speaker 192 times at about a 1000 Hertz rate to make a beep; I do it 10 times to make a click.

Lines 1210-1220 pick up the keypress code again and clear the keyboard strobe. This sets up the keyboard electronics so that you can read the next keypress next time around.

Lines 1230-1240 test the shift key. If it is down, the BPL will branch to the upper case section at line 1320. If the shift key is not down, lines 1270-1280 test whether the character is a letter. If so, line 1290 makes it into a lower-case code.

I am using the codes from $E0 through $FF for lower-case. This is standard ASCII, and is also compatible with the various lower-case display adapters available on the Apple. $E1 through $FA are the letters a-z; $E0 is a tick-mark; $FB-FF are special punctuation marks. If you don't have a lower-case display adapter, these codes display as punctuation and numbers.

Lines 1320-1420 handle characters typed with the shift key down. If the code is less than $C0, the keyboard input code is correct already. Above $C0, the code is correct unless you have typed M, N, or P. The Apple translates these shifted letters into @, ], and ^, respectively. My logic translates them back into capital letters.

I use a special control sequence to enter the punctuation characters with codes above $C0, which is not shown here. You type control-O, which stands for "override", and then one of the letters klmnop or KLMNOP. The letter translates into the corresponding punctuation code. For example, control-O, shift-M is a right bracket (]); control-O, shift-P is an at-sign (@).

===============================================================================
DOCUMENT :AAL-8206:Articles:XPlot4ASoft.txt
===============================================================================

A New Hi-Res Function for Applesoft............ Mike Laumer

Most people use the language card as nothing more than a ROM simulator for the other version of BASIC that is not on the motherboard. But it can do much more since the memory is actually RAM. Indeed Bob S-C's Macro Assembler has a version which runs in a Language Card. The FLASH! Integer BASIC compiler which I wrote uses the language card in place of a disk file providing higher speed compilations for those people who have a language card.

One nice aspect of having the language card is the ability to move Apple software from ROM to RAM in the card and make changes to add a new capability. Some people have done this already with the Apple monitor to add an extra feature or two at the expense of another (who needs the tape I/O routines).

The program assciated with this article will allow you to patch a RAM card version of Applesoft to modify the 'HPLOT' command to function as an 'HXPLOT' command. What is 'HXPLOT' you say. Remember the DRAW and XDRAW commands in Applesoft. The 'DRAW' command will place a shape on the screen; 'XDRAW' does the same thing, but 'XDRAW' has the unique ability to redraw the shape and erase it from the screen leaving whatever was on the screen initially still intact. The 'HXPLOT' function in the listing functions the same way for the 'HPLOT' command as 'XDRAW' does for the 'DRAW' command.

I have been developing a Hi-Res graphics editor as my next product. During the development cycle I was working with a line draw game paddle routine. You move a cursor to a position and anchor one end of the line to a point. Then you can move to another position and while you move a line stretches out from the point like a rubber band to the current cursor position. This gives you a preview of what the line looks like before you plot the line. The 'HXPLOT' function does have one sleight problem: it plots independent of the current color.

What the function actually does as it draws a line is to invert each dot of the line path instead of plotting a color. When the same line is drawn with the same coordinates the bits on the line path are inverted again back to their original value, restoring the screen to what it was before you started HXPLOTting.

You may be wondering why not just use the 'HPLOT' as it is to do this. You could just draw the line once with a color of 3 then change the color to 0 and erase the line with another 'HPLOT'. This only works if you have a black screen with no other images on it. If their are other images on the screen then when you erase the line you will draw a black line through those other images causing them to change. Only a function like 'XDRAW' or the 'HXPLOT' will be non-destructive of the background data on the screen.

How It Works

The 'HPLOT' command in Applesoft is actually two commands in one.

```
     HPLOT x, y              plots 1 point
     HPLOT x1,y1 TO x2,y2    plots a line
```

Each of the routines have one common place where they plot a bit onto the hi-res screen. The point plotting routine is at $F457 in the ROM and the line routine is at $F53A in the ROM. By putting Applesoft into the RAM card we can patch into these routines and modify their operation.

The two areas that are patched are at $F457 and $F58D. After you run the patch program you should see the Applesoft prompt character and there will be no program in memory. So type in the small demo program listed here and run it.

<<<<program here>>>>

If you have an Integer BASIC motherboard you should boot up your system master disk and have Applesoft loaded into your RAM card before using the routine.

==============================================================================
DOCUMENT :AAL-8206:Articles:Yes.No.txt
==============================================================================

Yes/No Subroutine.......................Bob Sander-Cederlof

It happens all the time!  I am continually needing to ask Yes/No questions in my programs.  I do it now with the following subroutine, which has been somewhat stripped down for publication.

Assume you have just printed the question itself on the screen, preferably with " (Y/N)?" on the end.  Then call my subroutine with "JSR YES.NO".  The subroutine will clear the keyboard strobe, so that it is sure it is getting the answer to this question, and not just a stray character you accidentally typed.  Then as soon as you hit any key, it will put it on the screen where the question ended and return to you.

At the point you should use BNE to branch where you want to if the user has typed something other than "Y" or "N".  Once that is out of the way, use BCC or BCS to branch on whether it was "Y" or "N".  The subroutine sets carry for "N" and clears carry for "Y".

In my actual programs, I have one more line between 1120 and 1130.  It is JSR MESSAGE.PRINTER, which expects a message number in the Y-register.  You can use it either way.  You might also like to insert two more lines to call the message printer to print " (Y/N)?  " for every question; that way the common string does not have to be repeatedly stored in memory with every question.

```
==============================================================================
DOCUMENT :AAL-8206:DOS3.3:HXPLOT.DEMO.txt
==============================================================================
```

```
ê:π769,1!π28,127:å62454=ÅI–0¡279«10:ÅJ–0¡192«10Mì140,96¡I,J[-ÅZ–
1¡1:ÇZk(ì140,96¡I,Ju2ÇJ:ÇI}d´10
```

```
==============================================================================
DOCUMENT :AAL-8206:DOS3.3:S.AUTO.CATALOG.txt
==============================================================================

 1000  *---------------------------------
 1010         .OR $3200
 1020         .TF PATCH
 1030  *---------------------------------
 1040  CH     .EQ $24
 1050  BASL   .EQ $28
 1060  XSAVE  .EQ $40
 1070  WBUF   .EQ $200
 1080  *---------------------------------
 1090  ESCAPE.C
 1100         CPX #0       BEGINNING OF LINE?
 1110         BNE .2       NO, RETURN
 1120         LDY #1
 1130  .1     LDA MSG-1,Y  GET CHARACTER
 1140         STA (BASL),Y PUT ON SCREEN
 1150         STA WBUF,X   PUT IN BUFFER
 1160         INY
 1170         INX
 1180         CPY #8       DONE?
 1190         BNE .1       NO
 1200         STY CH
 1210         STX XSAVE    TELL THE ASSEMBLER
 1220         TSX          THAT THIS WAS AN
 1230         LDA #$CC     ESCAPE-L, SO IT WILL
 1240         STA $103,X   GO AHEAD AND EXECUTE
 1250         LDX XSAVE    THE COMMAND
 1260  .2     RTS
 1270  *---------------------------------
 1280  MSG    .AS -/CATALOG/
```

==============================================================================
DOCUMENT :AAL-8206:DOS3.3:S.BubbleSrtDemo.txt
==============================================================================

```
 1000  *--------------------------------
 1010  *      BUBBLE-SORT DEMO
 1020  *--------------------------------
 1030  LIST   .EQ $00 THRU $0F
 1040  N      .EQ 16
 1050  FLAG   .EQ $10
 1060  *--------------------------------
 1070  BUBBLE LDY #0       INITIAL INDEX
 1080         STY FLAG     INTERCHANGE FLAG
 1090  .1     LDA LIST+1,Y COMPARE TWO ADJACENT ITEMS
 1100         CMP LIST,Y
 1110         BCS .2       ALREADY IN CORRECT ORDER
 1120         PHA          INTERCHANGE THEM
 1130         LDA LIST,Y
 1140         STA LIST+1,Y
 1150         PLA
 1160         STA LIST,Y
 1170         LDA #$FF     SET INTERCHANGE FLAG
 1180         STA FLAG
 1190  .2     INY          NEXT OVERLAPPING PAIR
 1200         CPY #N-1
 1210         BCC .1       STILL A PAIR LEFT
 1220         LDA FLAG     WAS AN INTERCHANGE PERFORMED?
 1230         BNE BUBBLE   YES, MAKE ANOTHER PASS
 1240         RTS          NO, ALL SORTED
```

```
================================================================================
DOCUMENT :AAL-8206:DOS3.3:S.EXAMINER.txt
================================================================================

 1000  *---------------------------------
 1010         .OR $300
 1020         .TF EXAMINER
 1030  *---------------------------------
 1040  POINT  .EQ $00,01
 1050  PAGE   .EQ $01
 1060  CH     .EQ $24
 1070  *
 1080  KEYBOARD .EQ $C000
 1090  STROBE   .EQ $C010
 1100  *
 1110  PRBL2  .EQ $F94A
 1120  HOME   .EQ $FC58
 1130  RDKEY  .EQ $FD0C
 1140  CROUT  .EQ $FD8E
 1150  PRBYTE .EQ $FDDA
 1160  COUT   .EQ $FDED
 1170  *---------------------------------
 1180  START  LDA #$00
 1190         STA POINT    START WITH
 1200         STA PAGE     PAGE ZERO
 1210  *---------------------------------
 1220  DISPLAY.NEW.PAGE
 1230         JSR HOME
 1240         JSR PRINT.HEADER
 1250         JSR CROUT
 1260         LDY #$00
 1270  *
 1280  NEW.LINE
 1290         LDX #$0C     TWELVE BYTES AT A TIME
 1300         TYA
 1310         JSR PRBYTE   PRINT INDEX
 1320         LDA #$A0
 1330         JSR COUT     SPACE
 1340  .1     LDA (POINT),Y
 1350         JSR PRBYTE   PRINT HEX
 1360         INY
 1370         BEQ FILLIN   PAGE DONE?
 1380         DEX
 1390         BNE .1       TWELVE YET?
 1400  *
 1410  ADJUST TYA
 1420         SBC #$0C     RESET Y
 1430         TAY
 1440         LDA #$A0
 1450         JSR COUT     SPACE
 1460         LDX #$0C     TWELVE AGAIN
 1470  *
 1480  ASCII  LDA (POINT),Y
```

```
1490        CMP #$40     INVERSE?
1500        BCS .1       NO
1510        ORA #$C0     NORMALIZE
1520  .1    CMP #$80     FLASHING?
1530        BCS .2       NO
1540        ORA #$80     NORMALIZE
1550  .2    CMP #$A0     CONTROL?
1560        BCS .3       NO
1570        LDA #$AE     PUT PERIOD
1580  .3    JSR COUT     SEND IT
1590        INY
1600        BEQ GET.COMMAND   PAGE DONE?
1610        DEX
1620        BNE ASCII    LINE DONE?
1630        BEQ NEW.LINE
1640  *
1650  FILLIN LDX #$10     FILL LAST PARTIAL
1660        JSR PRBL2    LINE WITH SPACES
1670        LDY #$08     ADJUST Y
1680        BNE ADJUST
1690  *--------------------------------
1700  GET.COMMAND
1710        JSR CROUT
1720  .1    JSR RDKEY
1730        CMP #$9B     ESCAPE?
1740        BEQ .2
1750        CMP #$95     RIGHT ARROW?
1760        BEQ .3
1770        CMP #$88     LEFT ARROW?
1780        BEQ .4
1790        CMP #$D0     "P"?
1800        BEQ GET.PAGE.NUMBER
1810        BNE .1       NONE OF THE ABOVE
1820  .2    RTS
1830  *
1840  .3    INC PAGE
1850        JMP DISPLAY.NEW.PAGE
1860  .4    DEC PAGE
1870        JMP DISPLAY.NEW.PAGE
1880  *--------------------------------
1890  GET.PAGE.NUMBER
1900        JSR PRINT.HEADER
1910  .1    DEC CH       SO PRBYTE WILL ALWAYS
1920        DEC CH       DISPLAY IN SAME PLACE
1930  .2    LDA KEYBOARD
1940        BPL .2
1950        STA STROBE
1960        CMP #$8D     RETURN?
1970        BEQ .5       YES, EXIT
1980        EOR #$B0
1990        CMP #$A      0-9?
2000        BCC .3       YES
2010        ADC #$88
2020        CMP #$FA     A-F?
```

```
2030         BCC .2       NO
2040  *
2050  .3     LDY #$3      LOOP 4 TIMES
2060         ASL          THROW AWAY HIGH NYBBLE
2070         ASL
2080         ASL
2090         ASL
2100  .4     ASL          SHIFT INTO
2110         ROL PAGE     PAGE NUMBER
2120         DEY
2130         BPL .4
2140         LDA PAGE
2150         JSR PRBYTE   DISPLAY PAGE NUMBER
2160         JMP .1       GET NEXT KEYPRESS
2170  .5     JMP DISPLAY.NEW.PAGE
2180  *--------------------------------
2190  PRINT.HEADER
2200         LDY #$00
2210  .1     LDA QPAGE,Y
2220         JSR COUT
2230         INY
2240         CPY #$05
2250         BNE .1
2260         LDA PAGE
2270         JMP PRBYTE
2280  *
2290  QPAGE  .AS -/PAGE=/
```

```
==============================================================
DOCUMENT :AAL-8206:DOS3.3:S.HXPLOT.txt
==============================================================

 1000  *--------------------------------
 1010  * THIS ROUTINE ADDS AN XPLOT CAPABILITY
 1020  * TO APPLESOFT.  THE FLAG AT $301 (769)
 1030  * CONTROLS WHETHER HPLOT OR XPLOT IS
 1040  * FUNCTIONING.
 1050  *
 1060  *      POKE 769,0   ENABLES HPLOT
 1070  *      POKE 769,1   ENABLES XPLOT
 1080  *--------------------------------
 1090         .OR $300
 1100         .TF B.HXPLOT
 1110  NEW.HLIN LDA #0     TEST 'XPLOT' FLAG
 1120         BNE .2       YES 'XPLOT' MODE
 1130         LDA ($26),Y  PLOT NORMAL LINE
 1140         EOR $1C
 1150         AND $30
 1160  .1     JMP $F593    BACK INTO APPLESOFT LINE ROUTINE
 1170  .2     LDA #$7F     MASK COLOR SHIFT BIT
 1180         AND $30      OFF OF BIT MASK
 1190         AND ($26),Y  TEST SCREEN BIT
 1200         BNE .1       BIT IS SET!... SO CLEAR IT
 1210         LDA #$7F     BIT IS CLEAR!...SO SET IT
 1220         AND $30      BIT MASK WITHOUT COLOR SHIFT BIT
 1230         BPL .1       BRANCH ALWAYS
 1240  *--------------------------------
 1250  NEW.PLOT JSR $F411  CALL HPOSN ROUTINE
 1260         LDA $301     TEST 'XPLOT' FLAG
 1270         BNE .1       YES 'XPLOT' MODE
 1280         JMP $F45A    PLOT NORMAL
 1290  .1     LDA #$7F     XPLOT
 1300         AND $30      MASK COLOR SHIFT BIT OFF
 1310         AND ($26),Y  TEST SCREEN BIT
 1320         BNE .2       SCREEN BIT IS SET
 1330         LDA #$7F     ...CLEAR SO PREPARE TO
 1340         AND $30      SET SCREEN BIT
 1350  .2     JMP $F460    BACK INTO APPLESOFT XPLOT ROUTINE
 1360  *--------------------------------
 1370  *
 1380  * TO USE THE ABOVE FUNCTION YOU MUST HAVE A RAM CARD.
 1390  * APPLESOFT MUST BE IN THE RAM CARD.
 1400  * THEN YOU MUST DO THE FOLLOWING:
 1410  *
 1420  * 0. BLOAD B.XPLOT.FOR.FP     LOAD THE XPLOT ROUTINE
 1430  * 1. CALL-151  TO ENTER THE MONITOR
 1440  * 2. C081 C081   TO WRITE ENABLE THE CARD
 1450  * 3. GO TO STEP 5 IF YOU HAVE AN INTEGER BASIC MOTHER BOARD
 1460  * 4. D000<D000.FFFFM   PUT APPLESOFT INTO RAM CARD
 1470  * 5. F58D:4C 00 03   PATCH FOR LINE ROUTINE
 1480  * 6. F457:4C 1B 03   PATCH FOR POINT PLOT ROUTINE
```

```
1490  * 7. C080            WRITE PROTECT THE RAM CARD
1500  * 8. 3D3G            START APPLESOFT UP
1510  *---------------------------------
1520  * FOR LAZY SOULS HERE IS AN AUTOMATIC PATCH ROUTINE.
1530  *---------------------------------
1540  MON.COUT .EQ $FDED   MONITOR CHARACTER OUT ROUTINE
1550         .OR $4000
1560         .TF B.PATCH.XPLOT
1570  START  LDY #0
1580  .1     LDA MESG,Y
1590         BEQ L.100
1600         JSR MON.COUT PRINT MESSAGE
1610         INY
1620         BNE .1       BRANCH ALWAYS
1630  MESG   .HS 8D84
1640         .AS -/BLOAD B.XPLOT.FOR.FP/
1650         .HS 8D00
1660  L.100  LDA $C081    ROM READ
1670         LDA $C081    RAM CARD WRITE
1680         LDA $E000    CHECK MOTHERBOARD ROM
1690         CMP #$20     IS IT INTEGER BASIC
1700         BEQ L.200    YES SO MUST HAVE FP FROM SYSTEM MASTER
1710         LDA #$D0     NO SO COPY FP FROM ROM TO RAM CARD
1720         STA $1
1730         LDA #0
1740         STA $0
1750  .1     LDY #0
1760  .2     LDA ($0),Y
1770         STA ($0),Y
1780         INY
1790         BNE .2
1800         INC $1
1810         BNE .1
1820  L.200  LDA #$4C     SET PATCHES INTO RAM CARD APPLESOFT
1830         STA $F58D
1840         STA $F457
1850         LDA #NEW.HLIN
1860         STA $F58E
1870         LDA /NEW.HLIN
1880         STA $F58F
1890         LDA #NEW.PLOT
1900         STA $F458
1910         LDA /NEW.PLOT
1920         STA $F459
1930         LDA $C080
1940         JMP $3D3     START UP RAM CARD APPLESOFT
```

```
=======================================================================
DOCUMENT :AAL-8206:DOS3.3:S.Look4ZP.txt
=======================================================================

 1000  *SAVE S.LOOK FOR PAGE ZERO
 1010  *-------------------------------
 1020  *      SEARCH FOR PAGE ZERO REFERENCES
 1030  *-------------------------------
 1040  MON.A1L    .EQ $3C
 1050  MON.A1H    .EQ $3D
 1060  MON.A2L    .EQ $3E
 1070  MON.A2H    .EQ $3F
 1080  MON.PCL    .EQ $3A
 1090  MON.PCH    .EQ $3B
 1100  *-------------------------------
 1110  KEYBOARD   .EQ $C000
 1120  STROBE     .EQ $C010
 1130  *-------------------------------
 1140  MON.LIST2  .EQ $FE63
 1150  MON.INSDS  .EQ $F88C
 1160  MON.A1PC   .EQ $FE75
 1170  MON.PCADJ  .EQ $F953
 1180  MON.NXTA1  .EQ $FCBA
 1190  *-------------------------------
 1200  *      SET UP CONTROL-Y VECTOR
 1210  *-------------------------------
 1220  SETUPY LDA #$4C     'JMP' OPCODE
 1230         STA $3F8
 1240         LDA #CTRL.Y
 1250         STA $3F9
 1260         LDA /CTRL.Y
 1270         STA $3FA
 1280         RTS
 1290  *-------------------------------
 1300  *      CONTROL-Y COMES HERE
 1310  *-------------------------------
 1320  CTRL.Y
 1330         JSR MON.A1PC IF ADDRESS SPECIFIED, PUT IN PC
 1340  .1     LDY #0
 1350         LDA (MON.PCL),Y
 1360         AND #$0F
 1370         CMP #1
 1380         BEQ .3
 1390         CMP #4
 1400         BCC .6
 1410         BNE .2
 1420         LDA (MON.PCL),Y
 1430         AND #$F0
 1440         CMP #$20     BIT Z
 1450         BEQ .3
 1460         CMP #$80
 1470         BCC .6       NO
 1480         CMP #$D0
```

```
1490        BEQ .6       NO
1500        CMP #$F0
1510        BEQ .6       NO
1520        BNE .3       YES
1530  .2    CMP #7
1540        BCS .6
1550  *-------------------------------
1560  *     INSTRUCTION REFERENCES PAGE-ZERO
1570  *-------------------------------
1580  .3    LDA #1       DISASSEMBLE THIS ONE INSTRUCTION
1590        JSR MON.LIST2   DISASSEMBLE
1600        LDA KEYBOARD    SEE IF KEYPRESS
1610        BPL .7       NO
1620        STA STROBE   YES, CLEAR IT
1630        CMP #$8D
1640        BEQ .5
1650  .4    LDA KEYBOARD
1660        BPL .4
1670        STA STROBE
1680        CMP #$8D
1690        BNE .7
1700  .5    RTS
1710  *-------------------------------
1720  *     DOES NOT REFERENCE PAGE-ZERO
1730  *-------------------------------
1740  .6    LDX #0
1750        JSR MON.INSDS    GET LENGTH OF INSTRUCTION
1760        JSR MON.PCADJ
1770        STA MON.PCL
1780        STY MON.PCH
1790  *-------------------------------
1800  *     TEST IF FINISHED
1810  *-------------------------------
1820  .7    LDA MON.PCL
1830        CMP MON.A2L
1840        LDA MON.PCH
1850        SBC MON.A2H
1860        BCC .1
1870        RTS
```

```
==============================================================================
DOCUMENT :AAL-8206:DOS3.3:S.MyOwnLtlBell.txt
==============================================================================

 1000  *---------------------------------
 1010  *      MY OWN LITTLE BELL
 1020  *--------------------------------
 1030  MON.DELAY  .EQ $FCA8
 1040  SPEAKER    .EQ $C030
 1050  *--------------------------------
 1060  SC.BELL
 1070         LDX #50
 1080  .1     LDA #14
 1090         JSR MON.DELAY
 1100         LDA SPEAKER
 1110         LDA #10
 1120         JSR MON.DELAY
 1130         LDA SPEAKER
 1140         LDA #6
 1150         JSR MON.DELAY
 1160         LDA SPEAKER
 1170         DEX
 1180         BNE .1
 1190         RTS
```

```
==========================================================================
DOCUMENT :AAL-8206:DOS3.3:S.NewBrkOpcodes.txt
==========================================================================

 1010  *--------------------------------
 1020  *       IMPLEMENTING BSR, BRA, AND LEA OPCODES
 1030  *       USING THE 'BRK' VECTOR WITH THE
 1040  *       AUTOSTART ROM
 1050  *
 1060  *       ADAPTED FROM AN ARTICLE IN "LAB LETTERS"
 1070  *            BY A. SATO
 1080  *--------------------------------
 1090  MON.PC     .EQ $3A,3B
 1100  MON.XREG   .EQ $46
 1110  MON.YREG   .EQ $47
 1120  *--------------------------------
 1130  BRK.VECTOR .EQ $3F0,3F1
 1140  *--------------------------------
 1150  MON.BRK     .EQ $FA59
 1160  MON.RESTORE .EQ $FF3F
 1170  *--------------------------------
 1180  SETUP
 1190         LDA #BREAK.INTERPRETER
 1200         STA BRK.VECTOR
 1210         LDA /BREAK.INTERPRETER
 1220         STA BRK.VECTOR+1
 1230         RTS
 1240  *--------------------------------
 1250  BREAK.INTERPRETER
 1260         LDY MON.PC+1     PICK UP ADDRESS OF THIRD BYTE
 1270         LDX MON.PC
 1280         BNE .1       BACK UP TO SECOND BYTE
 1290         DEY
 1300  .1     DEX
 1310         STX PNTR     MODIFY ADDRESS IN GET.THIS.BYTE SUBROUTINE
 1320         STY PNTR+1
 1330         JSR GET.THIS.BYTE
 1340         TAY          OPCODE BYTE
 1350         JSR GET.NEXT.BYTE
 1360         PHA          ADDR-LOW BYTE
 1370         JSR GET.NEXT.BYTE
 1380         TAX
 1390         PLA
 1400         SEC          ADDR-HIGH BYTE
 1410         ADC PNTR     COMPUTE EFFECTIVE ADDRESS
 1420         STA EFF.ADDR
 1430         TXA
 1440         ADC PNTR+1
 1450         STA EFF.ADDR+1
 1460  *--------------------------------
 1470         CPY #$20     CLASSIFY OPCODE
 1480         BEQ BSR
 1490         CPY #$4C
```

```
1500        BEQ BRA
1510        CPY #$AE
1520        BEQ LEA
1530        STY MON.YREG
1540        JMP MON.BRK
1550 *---------------------------------
1560 BSR    LDA PNTR+1   PUSH RETURN ADDRESS ON STACK
1570        PHA
1580        LDA PNTR
1590        PHA          AND DO BRA
1600 *---------------------------------
1610 BRA    JSR MON.RESTORE
1620        JMP *-*
1630 EFF.ADDR   .EQ *-2
1640 *---------------------------------
1650 LEA    JSR GET.NEXT.BYTE  POINT AT NEXT INSTRUCTION
1660        JSR MON.RESTORE   RESTORE A-REG AND STATUS
1670        LDY EFF.ADDR      ADDR-LO IN Y
1680        LDX EFF.ADDR+1    ADDR-HI IN X
1690        JMP (PNTR)
1700 *---------------------------------
1710 GET.NEXT.BYTE
1720        INC PNTR
1730        BNE GET.THIS.BYTE
1740        INC PNTR+1
1750 GET.THIS.BYTE
1760        LDA $FFFF    (FILLED IN)
1770 PNTR   .EQ *-2
1780        RTS
1790 *---------------------------------
1800 MSG    .EQ 0,1
1810 JMP.COUT JMP $FDED
1820        .MA LEA
1830        >BRK LDX,]1
1840        .EM
1850        .MA BSR
1860        >BRK JSR,]1
1870        .EM
1880        .MA BRA
1890        >BRK JMP,]1
1900        .EM
1910        .MA BRK
1920        BRK
1930        ]1 ]2-:1
1940 :1
1950        .EM
1960 TEST   >LEA MESSAGE
1970        STX MSG+1
1980        STY MSG
1990        LDY #0
2000 .1     LDA (MSG),Y
2010        PHA
2020        ORA #$80
2030        >BSR JMP.COUT
```

```
2040        INY
2050        PLA
2060        BPL .1
2070        LDA #$8D     CARRIAGE RETURN
2080        >BRA JMP.COUT
2090  MESSAGE .AT /THIS IS MY MESSAGE/
```

```
==========================================================================
DOCUMENT :AAL-8206:DOS3.3:S.ReadKeyCase.txt
==========================================================================

 1000  *--------------------------------
 1010  *      READ KEY WITH CASE CONTROL
 1020  *--------------------------------
 1030  KEYBRD     .EQ $C000
 1040  KYSTRB     .EQ $C010
 1050  SPKR       .EQ $C030
 1060  SHIFT.KEY  .EQ $C063
 1070  *--------------------------------
 1080  MON.BELL2  .EQ $FBE4
 1090  *--------------------------------
 1100  KEY.CLICK.FLAG      .EQ $00
 1110  CASE.INPUT.FLAG     .EQ $01
 1120  CURRENT.CHAR        .EQ $02
 1130  *--------------------------------
 1140  READ.KEY.WITH.CASE
 1150         LDA KEYBRD   GET CHAR FROM KEYBOARD
 1160         BPL READ.KEY.WITH.CASE
 1170         LDA KEY.CLICK.FLAG   CLICKING?
 1180         BEQ .1       NO
 1190         LDY #10      YES, 10 HALF-CYCLES WILL
 1200         JSR MON.BELL2    SOUND LIKE A CLICK
 1210  .1     LDA KEYBRD   CHAR AGAIN
 1220         STA KYSTRB
 1230         BIT SHIFT.KEY    SHIFT KEY DOWN?
 1240         BPL .2       YES
 1250         BIT CASE.INPUT.FLAG
 1260         BMI .2       IN SHIFT LOCK UPPER CASE
 1270         CMP #$C0     NO, LOWER CASE IF LETTER
 1280         BCC .5       NOT A LETTER
 1290         ORA #$20     LETTER, MAKE LOWER CASE
 1300         BNE .5       ...ALWAYS
 1310  *---SHIFT KEY PRESSED------------
 1320  .2     CMP #$C0     SEE IF LETTER
 1330         BCC .5       NOT A LETTER KEY
 1340         BEQ .4       SHIFT-P
 1350         CMP #$DD     SHIFT-M
 1360         BEQ .3       YES
 1370         CMP #$DE     SHIFT-N
 1380         BNE .5       NO
 1390  .3     AND #$EF     MAKE CAPITAL-M OR -N
 1400         BNE .5       ...ALWAYS
 1410  *--------------------------------
 1420  .4     LDA #$D0     MAKE CAPITAL-P
 1430  .5     STA CURRENT.CHAR
 1440         RTS
```

```
================================================================================
DOCUMENT :AAL-8206:DOS3.3:S.YES.NO.txt
================================================================================

 1000  *-------------------------------
 1010  *      YES/NO SUBROUTINE
 1020  *
 1030  *      RETURN .NE. IF NEITHER "Y" NOR "N"
 1040  *             .EQ. AND .CC. IF "Y"
 1050  *             .EQ. AND .CS. IF "N"
 1060  *-------------------------------
 1070  STROBE     .EQ $C010
 1080  MON.RDKEY  .EQ $FD0C
 1090  MON.CH     .EQ $24
 1100  MON.BASE   .EQ $28 AND $29
 1110  *-------------------------------
 1120  YES.NO
 1130         STA STROBE
 1140         JSR MON.RDKEY
 1150         LDY MON.CH
 1160         CMP #'N+$80
 1170         BEQ .1
 1180         CMP #'Y+$80
 1190         BNE .2
 1200         CLC
 1210  .1     STA (MON.BASE),Y
 1220  .2     RTS
```

```
=====================================================================
DOCUMENT :AAL-8207:Articles:Animation.txt
=====================================================================
```

Simple Hires Animation...........................Mike Laumer

One thing that I have been working with in my next product (MIKE'S MAGIC MATRIX) is animation using hires graphics. I have been developing a hires graphics editor using the FLASH! Integer BASIC Compiler. I may not be the first one to bring a commercial product to market using the FLASH! compiler since there are at least six other programmers who are striving to beat me.

There are several methods used to achieve animation in the popular game programs. The one presented in this program is possibly the simplest. This program will animate an image in one place on the screen (in-place animation) from a series of frames of data.

The technique used to display the frame data on the screen is simply moving the data with 'LDA' and 'STA' instructions. A more powerful method of animation is to use the 'EOR' instruction to merge one frame of animation into the next. This is accomplished by using the frame data obtained by 'EOR'ing two successive frames of data. Then using that new data to 'EOR' to the image data. The 'EOR' istruction is very useful since it can add and delete data to and from the screen without disturbing any background that may be on the screen already.

A frame of data for the animation is written to the screen and then a delay loop entered to delay before the next data frame is written to the screen. If the delay is smaller the animation will speed up. If the delay is larger the animation will slow down. The delay could be read from the game paddle.

The method I used in the routine to compute the hires graphics screen addresses is to use two tables (one for lo-byte, one for hi-byte) with 192 entries to convert the Y-coordinate into a hires address. Otherwise, the Y-addresses would have to be computed by using a complicated formula:

```
    A = Y MOD 8
    B = (Y / 8) MOD 8
    C = Y / 64
    YADRS = 8192 + A*1024 + B*128 + C*40
    (add another 8192 if hires page2 adress needed)
```

So you see that even with an efficiently coded machine language routine to compute a screen address it will take a bit of time to perform. It is much more effecient to simply look up the address of the first byte of the Y-row in a table. Since the Y-coordinate never exceeds 191 (which is less than 256) the Y-register can be used easily to index the table. The table in the program only provides the offset from the beginning of a hires page. The program uses an 'ORA' instruction to put $20 or $40 into the hi-byte to specify hires page 1 or 2.

The data for the animations were built with MIKE'S MAGIC MATRIX and the first frame looks like this:

(a printer dump goes here)

The data was written to a text file from within the editor and run through an Applesoft program to create an EXEC file for the S-C Macro Assembler to insert the data tables into the program.

You can make your own frames of animation by a hand process of drawing the animation dots on graph paper and reducing the data into hexadecimal data. To do this you must take each row of dots (on or off) on the graph paper and take them 7 dots at a time. The 7 dots must then be flipped into reverse order before converting into hex. Here is an example of 14 pixels width:

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
---------------------------
. * . * * * . . * * * . * *

. . * * * . * .  . * * . * * * .
0 0 1 1 1 0 1 0  0 1 1 0 1 1 1 0
---------------  ---------------
7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0

      $3A              $6E
```

As you can see the process is a pain in the neck. If the animation has a flaw in it you have to repeat the process for every frame of data that is wrong. That is where a hires graphics editor and animation design tool like MIKE'S MAGIC MATRIX really shines, because you can perfect your animation and test it in the editor without ever leaving. MIKE'S MAGIC MATRIX is not yet ready for sale lacking a manual and a little more work. I expect to have the first version ready in about two more months. Preliminary showings to the Dallas Apple Corps indicated an enormous popularity.

Since hexadecimal strings take up a lot of listing space when they are assembled, I decided to print the tables here using just the LIST command, without the assembled object code listing. The program part is shown in the normal assembled format.

Here is what you will see if you get it all typed correctly:

<9 little men here>

Of course, they will all appear one after the other in the same screen position, not side-by-side.

=============================================================================
DOCUMENT :AAL-8207:Articles:Axlon.Review.txt
=============================================================================

AXLON RAMDISK 320: A Review.........................Bill Morgan

AXLON's RAMDISK 320 is a system designed to add 320K of memory to an Apple, configured to look to the Apple like two very fast disk drives. The speed improvement ranges from half the time for a large assembly to one-twelfth the time for directly dumping 192 pages of memory.

Hardware

The RAMDISK is a cabinet just the size of an Apple disk drive, containing the memory, its own power supply, and a backup battery. There is also a large interface card, which includes 2K of static RAM for the operating system.

The backup battery is said to provide up to three hours of protection against power outage. It did maintain power when we moved the system into another room (about 5 minutes), but you should certainly make floppy disk backups of the RAMDISK data before leaving the system unplugged overnight. As long as it is plugged into the wall, the battery is kept charged and the memory is maintained.

Software

There are several programs supplied with the RAMDISK. These fall into the general categories of system software, utilities, and demonstrations.

RAMDSK1 is the operating system, which is stored in static RAM on the interface card, addressed in the $C800-CFFF space. BRUNning this program hooks it into DOS and copies one or two mechanical drives into the RAMDISK.

RDCOPY copies between the mechanical and RAM disks, to load or back up the RAMDISK. SELECT creates modified versions of RAMDSK1 for different slot/drive configurations.

The EXTRA40K utility allows you to access "tracks" 36-40 on the RAMDISK, but only on a level comparable to using RWTS directly. That is, you must work in terms of addresses and track/sectors rather than variables and filenames. The manual has a complete assembler source listing of this program.

SECTOR CHECKER and BYTE-BY-BYTE are self-test utilities to verify correct operation of the RAMDISK.

The demonstrations are The Directory and the Mini-Base Phone Book. The Directory is a large, disk-based, data-base program, in machine language, which uses the speed of the RAMDISK to its full advantage. The problem with this program is that it is strictly fixed-format, with no provision for modifying the record structure. The fields built into a record are last name, first name, dept #, mail stop, phone, special interest 1, special interest 2, and comments. If you are a large company needing an on-line, internal phone directory, then The Directory is outstanding. Otherwise, it's just an interesting demonstration of the system's capabilities.

The Mini-Base Phone Book is a memory-based data base, somewhat similar to File Cabinet. The Mini-Base is also set up as an internal phone directory, but since it is written in Applesoft, it can be modified to suit your needs. The documentation includes instructions for changing the record structure. The manual also contains instructions for calling special machine-language routines for keyboard input, fast loading of text files (in a specified format), and fast sorting of a string array.

Documentation

The manual is in three sections: 63 pages on the system, 34 pages on The Directory program, and 43 pages on the Mini-Base Phone Book program. It all comes in a large (8 1/2 by 11) 3-ring binder. The system section has chapters on setting up the RAMDISK, using the included software, calling it from DOS 3.3, attaching and using it in Pascal, technical information, and accessing the system from assembly language.

The setup and software chapters are quite good; the DOS chapter just says that everything is standard. I don't have Pascal, so I can't evaluate that section. The technical and assembly language chapters have all the information about memory usage, addressing, and programming techniques needed to use the RAMDISK without all of DOS's overhead.

Using the RAMDISK

To use the RAMDISK with your programs, you need to copy the RAMDSK1 program onto your disk and set up the HELLO program to BRUN RAMDSK1. This will load the operating system into the interface card, then fast-copy your disk into drive one of the RAMDISK. Once your information is loaded into the RAMDISK, you can use all the normal DOS techniques to read and write files; the only difference is speed.

You can avoid the DOS overhead either by calling RWTS in the usual manner, or by directly using the RAMDISK registers and memory window. To do that, you just store track, sector, and drive information into two bytes, then read the data from $C800-C8FF. This approach is fastest, but you must then take on all memory management chores. Appendices to the manual list assembler source code for routines using both techniques.

The Negative Side

We discovered one apparent bug in the RAMDISK's operating system. The program does not properly update the slot and drive found parameters in the I/O Control Block used by RWTS. If a program tries to use those locations to determine which drive it was run from, it will get the wrong data.

Mechanical disk drives are known to be error-prone, so DOS has some built-in protection against errors. Each sector is recorded with a checksum; when a sector is read the checksum should match. This is very poor protection, but it does catch most errors. The RAMDISK has no such protection. The RAMDISK is much less likely to have any errors than the mechanical drives, yet it still would be nice to have at least a sector checksum. Parity on each byte would be even better, but it would be expensive.

Timing Comparisons

| Operation | Disk II time | RAMDISK time |
|---|---|---|
| Assemble 102 sectors of source code. | 89 sec. | 41 sec. |
| BLOAD Hi-res screen. | 11 sec. | 3 sec. |
| LOAD Applesoft program. | 14 sec. | 4 sec. |
| Dump RAM (192 sectors) calling RWTS. | 9 sec. | .8 sec. |
| Dump 192 sectors direct | n/a | .7 sec. |

Summary

The RAMDISK is a well-made and well-documented unit; it performs as advertised. The RAMDISK gives a terrific speed improvement over mechanical disk drives, especially if you do your own reading and writing and avoid DOS.

Two standard Apple drives with controller at normal retail prices would cost $1180; RAMDISK goes for $1395, and you get the equivalent of 10 extra tracks thrown in. (On the other hand, several non-Apple drives are available with 40 to 80 tracks, at competitive prices. And the 5- and 10-megabyte Winchesters are rapidly falling in price.)
I have seen RAMDISK advertised for as low as $1170 in Byte Magazine.

The RAMDISK 320 is available from AXLON, Inc., 170 N. Wolfe Rd., Sunnyvale, CA 94086, (408) 730-0216. RAMDISK 320, The Directory, and Mini-Base Phone Book are trademarks of AXLON INC.

```
=============================================================================
DOCUMENT :AAL-8207:Articles:Flash.Ad.txt
=============================================================================
```

Christmas in July?.........................Bob Sander-Cederlof

Mike Laumer has decided to offer a special price to readers of Apple Assembly Line on his FLASH! Integer BASIC Compiler. For a limited time, AAL readers can buy FLASH! for only $49, a savings of almost 40% from the normal $79 price. The offer expires September 1, 1982, and is limited to one per customer. To qualify you must mention that you read about it in AAL, and call or write directly to Laumer Research. Mike's phone is (214) 245-3927; write to 1832 School Road, Carrollton, TX 75006.

What a bargain! The FLASH! compiler is an incredible software design tool which can translate Integer BASIC programs into extremely fast machine language programs. It is the only full feature compiler on the market that can provide assembly language listings and source files compatible with my S-C Assemblers.

Synergistic Software is now selling the Galfo Integer BASIC Compiler for $149; it is copy protected, has no assembly language output, fewer extensions to the language, an undocumented run-time package, and no option to buy the run-time package source code. I have heard that it is a good compiler, but I think the price is too high.

FLASH!, on the other hand, is NOT copy protected. You can make as many copies for your own use as you need. FLASH! adds features for hi-res graphics and system programming to the Integer BASIC language. The FLASH! run-time package is fully documented, and owners of FLASH! can get the source code of the run-time package on disk for only $39. FLASH! allows easy relocation of the object code for any requirements. Used in combination with the S-C Assembler, you can further optimize the object code for even greater memory and time savings. And at this special price, it truly is a bargain. Christmas in July!

==============================================================================
DOCUMENT :AAL-8207:Articles:Front.Page.txt
==============================================================================

!pr2
$1.50

Volume 2 -- Issue 10 July, 1982

In This Issue...



A New Software Tool: ES-CAPE

ES-CAPE stands for Extended S-C Applesoft Program Editor. You are somewhat familiar with it already as AED II from Linn Software. Bill has added more features, and I am in the process this month of re-doing the reference manual. I am shooting for it to be all packaged by the middle of July. The price will hold at $40 at least until September 1st.

If you are using Applesoft and feel the need for advanced editing tools to use on your programs-under-development, ES-CAPE ought to be in your tool-box. Like any tool, it doesn't do everything and it won't replace all your other tools. (You wouldn't try to tighten a screw with a hammer, or assemble a Heathkit with a SkilSaw....) But neither does it use all your money or memory!

Current Advertising Rates

For the August 1982 issue the price will be $60 for a full page, $35 for a half page. To be included, I must receive your camera-ready copy by July 20th.

==========================================================================
DOCUMENT :AAL-8207:Articles:Giant.Macro.txt
==========================================================================

Giant Macro for Writing Messages..............Robert B. Steiner

Every time I turn around I seem to need a quick and dirty routine to print out a message. I must have written them a dozen different ways, to fill various requirements. Sometimes they are only different because of a silly mistake...a difference usually called a bug. I could keep a handful of them on a subroutine library, but then I might get mixed up as to which one was which.

S-C Macro Assembler to the rescue! By writing one of largest macros I have ever seen, I can get all the message-printer-variants into one neat little package. Then by choosing the correct parameters, the kind of printing routine I want is generated on the spot.

I call the macro CRT, and you call it with up to five parameters. The call line will look like one of these:

```
>CRT L,N,"your message"
>CRT L,I,"your message"
>CRT A,N,address of your message
>CRT A,I,address of your message
```

The first parameter, which may be "L" or "A", indicates whether you will give an actual message in quotation marks, or the address of the message.

The second parameter, which may be "N" or "I", stands for Normal or Inverse video display.

The third parameter is either the message itself in quotes, or the address of the message (a label, of course).

An optional fourth parameter may be "I", "Y", or "R". "I" will generate code to read an immediate one byte reply, which is returned in the A-register. "Y" will generate the one byte reply code, followed by additional code to check for a yes/no response. It will loop until you type "Y" or "N"; then it will echo the letter, print a RETURN, and return with the character in the A-register.

If the fourth parameter is "R",an entire line of reply is expected. If there is no fifth parameter, the line will be at $200 for your program to analyze. If a fifth parameter is used, it is the name of a buffer for the reply message.

If I counted correctly, there are twenty different possible ways the macro can be generated!

Here is the macro definition, and some sample call lines. Try it out; you'll find it fun and educational, whether its useful to you or not. Then you can apply some of the techniques in your own work.

=============================================================================
DOCUMENT :AAL-8207:Articles:Hierographic.txt
=============================================================================

Hierographic Transport (review)...................Mike Laumer

Hierographic Tranport is a Hi-Res printer dump program for the Epson series of printers (MX-70, MX-80 and MX-100). The program is a very easy to use, menu driven system. The user manual is only 12 pages long, but most functions are self-apparent. I used the program for over an hour before I felt the need to refer to the manual. The program allows very complete control over the dot graphics mode of the Epson printers.

From the menus you can load a Hi-Res picture into either page 1 or page 2. Selections are provided for normal/inverse picture, normal/rotated pictures, normal/compressed print mode and a setable left margin to allow centering a picture on the page.

You can control magnifying or scaling the picture from 1 to 99 times normal size in the X or Y directions. This magnification is performed by repeatedly printing each screen dot, in the X and Y directions. The magnification only affects the printed image and not the screen image.

There is also the ability to select a "window" from the Hi-Res Screen that will be printed on the printer. That way you can print the rectangular section of the screen that you are interested in.

The "window" is controlled with two sets of cursor control keys. The "WASZ" keys control the top and left sides of the cursor. While the familiar "IJKM" keys control the right and bottom sides of the cursor. This is adequate for controlling the "window" but I would have prefered one set to control inward movement of the cursor sides and the other set to control outward movement of the cursor sides.

The cursor is presented as a set of blinking lines overlayed on the picture image. This technique uses the HXPLOT function described in the June issue of AAL. This function allows non-destructive lines to be drawn and erased over the top of an image on the Hi-Res screen.

The cursor lines are automatically stepped by an amount from 1-9, selectable by the number keys. The space bar or any other valid command key will stop the cursor from advancing. If "0" is selected for the step distance, the cursor lines will step by 1 whenever a cursor control key is pressed. This allows a fine positioning mechanism.

Once a "window" is selected the user can have it printed on his printer. When this is selected, the program automatically checks up on the parameters you have selected and computes the size of the image as it should be on the printer. If you have scaled the image too big, an error message will result.

The overall design of the program is good. There are however, a few minor problems in operation of the program. When the "window" is very large the automatic steps in advancing the "window" occur slowly. As the size of the "window" gets smaller, the speed of the automatic advance gets very fast making it hard to stop on the exact point you want. The cursor routine needs a delay which varies by the size of the "window" to help even out the speed of the automatic cursor advance.

There is a record of data kept at the bottom of the screen when you are selecting a "window". This data provides you with the cursor locations and a unique display of the computed size of the picture to be printed. As the cursor is moved, the data is updated to the new recomputed picture size. The size display often flickers because blanks are written to the screen and then the data is written. If the data were written then the line cleared to the end of line, the flicker would be less noticeable.

The size display had the only bug in the whole program that I could find. The bug is rather trivial and does not affect the quality of the program. A bug, however, is a bug! [ I am sure they will fix it, once they read this review. ] When a very large scale factor (99 x 99) is used, the routine to print out the size goes bananas and displays some garbage characters on the screen. When compressed printing is selected (where the dot spacing on the Epson goes from 1/60 of an inch to 1/120 of an inch on the horizontal direction), the size display goes one character too far and scrolls the data up the screen. As the cursor window is moved arround the scrolling eventually scrolls the title lines off the main menu.

Unless you plan to print a wall mural for the side of your barn, you should never encounter the problem. A 99 x 99 scaling factor will give a pixel size of 1.5 inches square! A full screen print would be 38 feet by 21 feet in size!!! That's way beyond the carriage width of even the MX-100. The program could handle it though as long as you print it in narrow window strips. (A nice future enhancement would be for the program to automatically print an oversize picture in strips sized for your particular printer.)

The program has a built in configuration routine and can easily be configured for the following interfaces:

Epson APL
CPS Multi-function
Grappler
Micro Buffer II
Prometheus
Apple parallel
Epson APL (modified for 8 bit Transmission)

The Epson printers are very popular, but many more brands of printers are now on the market which have comparable capabilities. For example, the NEC PC-8023, the MPI-88G, and the Okidata series. I hope that the GSR folks come out with equivalent "Transports" for these other printers. All of them on the same disk would be especially nice!

Conclusion: A fine program for graphics printer dumping. I rate this program a "B+". A little attention to its few problems would raise the grade to "A".

This program is sold for $39.00 and is available from GSR Associates, P.O. Box 401462, Garland, Texas 75040. (Don't be afraid of the P. O. Box, they are real people.)

======================================================================
DOCUMENT :AAL-8207:Articles:OtherEpson.Man.txt
======================================================================

Still More About "The Other Epson Reference Manual"

No sooner did I print my cutting comments about Cut The Bull Software last month, than I received a copy of the new edition of "The Other Epson Manual" in the mail. Bill Parker, author and publisher, has done an excellent job. By now all of you who ordered the booklet should have received your copy.

Bill has now quit his previous job to devote full time to the software company. The nature of that previous job prevented him from publishing his telephone number. Now you can reach him at (714) 223-3576. He says that in the future should a back order situation develop he will hold customer checks until ready to ship.

=============================================================================
DOCUMENT :AAL-8207:Articles:Relocatable.JSR.txt
=============================================================================

Run-Anywhere Subroutine Calls...........Bob Sander-Cederlof

Bob Nacon (author of Amper-Magic) called yesterday and told me about his new way to call subroutines in programs that will be loaded anywhere in memory without relocation or reassembly. He does this a lot inside Amper-Magic, and you might want to do it yourself sometime.

Instead of JSR <subroutine name>, put the following three lines whenever you call a subroutine:

```
CLV
JSR $FF58
BVC <subroutine name>
```

The byte at $FF58 in the monitor ROM is always $60, an RTS instruction. Since this is used by most Apple interface boards, Apple has guaranteed that it will always be $60. The JSR to a guaranteed RTS instruction seems silly, doesn't it? Not quite, because it does put two bytes on the stack, and then pop them off again. But we can get them back later, inside the called subroutine, like this:

```
TSX    GET STACK POINTER
DEX
DEX
TXS    REVISED STACK POINTER
```

Now the subroutine we called has a return address to go to, just as though we had used JSR <subroutine name>! The only problem is that if we execute an RTS, we will re-execute the BVC <subroutine name> and be in a loop. Unless....

Unless we set overflow, so the BVC falls through. But there is no SEV opcode in the 6502, so what do we do? $FF58 to the rescue again! Here is how we end the subroutine:

```
BIT $FF58    SET OVERFLOW
RTS
```

The BIT instruction copies bit 7 of $FF58 into the Carry Status bit, and bit 6 into the Overflow Status bit. This, in other words, (since $FF58 has $60 in it) clears carry and sets overflow. If you want carry to be set as a return flag, you can insert SEC between the BIT and RTS lines.

I thank Bob Nacon for this technique, and he thanks Roger Wagner for putting him on the trail to its discovery. Roger writes the monthly column in Softalk Magazine called "Assembly Lines"; the December, 1981, issue covered writing run-anywhere programs. If you haven't got Roger's book yet, called "Assembly Lines: The Book", it is currently the best book for beginners that I know of. The regular price is $19.95+$2 shipping, but I sell them for $18+$2 shipping.

==============================================================================
DOCUMENT :AAL-8207:Articles:Showfile.txt
==============================================================================

A Text File Display Command for DOS........Bob Sander-Cederlof

How many times have you wished that you could see what was in a TEXT file? You end up loading a word processor (if you are lucky enough to have one that can read normal DOS TEXT files), or EXECing it into the S-C Macro Assembler, or writing a special Applesoft program.... Why not a DOS command for this very common need?

The June 1982 issue of Call A.P.P.L.E. has an article by Lee Reynolds describing the addition of a FILEDUMP command to DOS. Lee gives a 20-byte program which fits nicely in an unused space in DOS. He replaced the MAXFILES command with "FILEDUMP". In case you want to try it, here are the patches for Lee's method.

```
]CALL -151
*BCDF:20
*BCE0:8E FD 20 A3 A2 20 8C A6 F0 05 20 F0 FD D0 F6 20
*BCF0:FC A2 60
*A8E7:46 49 4C 45 44 55 4D D0
*9D48:DE BC
*A933:20 30
*3D0G
]
```

$BCDF-BCF2 is the FILEDUMP command processor. $A8E7-$A8EE is the string "FILEDUMP", the command name. The two bytes at $9D48,9D49 are the address (minus 1) of the command processor. The two bytes at $A933,A934 are flags indicating that the FILEDUMP command requires a filename, and can optionally have S and D parameters.

My first reaction to the program, being a programmer, was to try to modify it. The first change I made saved one byte. The last two instructions are a JSR and an RTS. By ending with a JMP to the final subroutine, the RTS at BCF2 is not needed. Then I tried modifying the order of the loop, and saved another three bytes. Here is my revised listing:

<listing here>

After playing with the new command a little, I thought of several more changes. I wanted to be able to stop the file listing, to restart it, and to abort it. The first article I ever wrote about Apples described just such an addition, at that time for Integer BASIC. (See MICRO, June/July, 1978.) With this addition, the program would not fit in the unused space at $BCDF, so I decided to put it in the place of the INIT command instead. I changed the name to "SHOW".

Not all of the code would fit in the spot where the INIT command processor is, at $A54F. Therefore I broke out the routine to check for the pause/abort keys as a separate subroutine, and placed in over the top of some of the INIT code inside the File Manager of DOS. If you install this patch, you could call on the PAUSE.CHECK subroutine from your own programs.

<listing here>

After assembling the program above, the various pieces are in memory in page 8 and 9, instead of inside DOS. I did it this way because DOS is protected during assembly. You can install the patches by hex input commands, or by some memory moves. I did it this way:

```
:$A54F<84F.863M
:$AE8E<88E.8A4M
:$A884:53 48 4F D7
:$A909:20 30
```

Then try typing "SHOW filename", where "filename" is a text file, and see the action.

You may want to put some POKEs in your HELLO file on some disks to install the SHOW command. If so, this is what they might look like:

```
100 DATA 21,42319,32,163,162,169,141,32,240,253,32,142,
         174,240,5,32,140,166,208,243,76,252,162
110 DATA 23,44686,173,0,192,16,17,141,16,192,201,141,
         240,10,173,0,192,16,251,141,16,192,201,141,96
120 DATA 4,43140,83,72,79,215
130 DATA 2,43273,32,48
140 DATA 0
150 READ N : IF N THEN READ A : FOR I = 1 TO N : READ D
    : POKE A+I-1,D : NEXT : GO TO 150
```

I tried several other versions, with features like clearing the screen, filling it up, and waiting; a stand-alone program, rather than a DOS command; and so on. You will probably want to try your own experiments.

```
=============================================================================
DOCUMENT :AAL-8207:Articles:Sorted.ZeroPage.txt
=============================================================================
```

Sorting Out Zero-Page References.............Tracy L. Shafer

The search for page-zero references program in last month's AAL turned out to be (almost) the very thing I've been needing.

I have a clock card capable of generating NMI and IRQ interrupts.  Up to now, I haven't been able to do any deep research on the IRQ due to the DOS and monitor conflict mentioned in the January issue of AAL. (They both use location $48.)  I can't modify the monitor because I don't have access to a PROM burner, and the thought of searching through DOS really put a damper on the IRQ project until now.

Since I didn't need to know every page-zero reference used by DOS, I modified the program to search for a specific page-zero reference. That worked fine, but I didn't want to have to type in a separate search value for every group of references I might need later, so I further changed the program to print out all the references in numerical order of page-zero location.

To make the changes to the program as published last month, just remove the ".3" from line 1580 and add the following lines:

```
1285 PAGE.REF .HS 00           VARIABLE TO HOLD THE
                               CURRENT ZERO-PAGE
                               LOCATION

1571 .3        INY             NEW PLACE FOR ".3" LABEL
1572           LDA (MON.PCL),Y GET PAGE REFERENCE
1573           DEY             RESTORE VALUE OF Y
1574           CMP PAGE.REF    ONE WE ARE SEARCHING FOR?
1575           BNE .6          NO, IGNORE THIS ONE

1861           LDX #1          RESTORE X-VALUE FOR
                               MON.A1PC ABOVE
1862           INC PAGE.REF    NEXT ZERO-PAGE ADDRESS
1863           BNE CTRL.Y      NOT FINISHED
```

The program now searches through the memory range 256 times instead of just once, so it doesn't run nearly as fast, but it's easier to find all the references to specific locations.

==============================================================================
DOCUMENT :AAL-8207:Articles:Who.Are.We.txt
==============================================================================

Who are "we" and what are "we" doing?............Mike Laumer

Some of you may wonder about the people whose articles you see in the AAL on a fairly regular basis and who you may have talked to on the phone at one time or another.

Bob Sander-Cederlof is the president of the S-C Software Corporation and the author of the S-C Assemblers and Double Precision Floating Point package. Bob has been working with computers since 1957, at such places as Control Data Corporation and Texas Instruments. He is developing a new text editor somewhat compatible with Apple Writer. Believe it or not the editor is half the size of Apple Writer. Both the editor and printer sections are in memory at once and it has more capabilities than Apple Writer. He also edits this newsletter every month, with the aid of Bill Morgan.

Bill Morgan is Bob's first full-time employee and helps in all areas: programming, shipping, accounting, phone sales, and writing articles for the AAL. He helps author the reference manuals as well, and tries to make our products fail before we start shipping them (so we can fix 'em before you see 'em!).

Bobby Deen is a part-time employee still in high school. He is currently helping Bob S-C develop a line of compatible Macro Cross Assemblers for 6800, 6809 and Z-80 processors to round out Bob's assembler product line. (The 6800 and 6809 versions are ready now.) He has helped develop an 18-digit decimal math package compatible with Applesoft soon to be a new product. He has also assisted in the CPR project with Mike Laumer.

Mike Laumer (that's me!) is owner of Laumer Research and author of FLASH! the Integer BASIC compiler, and of the upcoming MIKE'S MAGIC MATRIX hires graphics editor and animation design tool. As a sub-contractor to S-C Software for the last year, I have been working on an incredible application using Apples and video disks. You can read all about it in the June 1982 issue of BYTE magazine, pages 108-138. The American Heart Association sponsors the project, which will teach Cardiopulmonary Resuscitation (CPR). The Apple is supported by a video disk player, light pen, two CPR manikins, a random-access audio unit, and two monitors.

If you have called, you may have talked with Bob's daughter Patricia (oldest of five children). She is a Junior in High School, and works part-time at shipping, phone sales, mailing list maintenance, word processing, Visicalc-ing, program entry, paste-up and folding, and whatever comes up. She is assisted by Lisa MacCorkle, another high school friend.

We enjoy talking with all of you, so if you have a problem, need a book, or whatever, give us a call!

```
==============================================================================
DOCUMENT :AAL-8207:DOS3.3:Inst.Show.Cmd.txt
==============================================================================
```

\dÉ21,42319,32,163,162,169,141,32,240,253,32,142,174,240,5,32,140,166,
208,243,76,252,162ªnÉ23,44686,173,0,192,16,17,141,16,192,201,141,240,1
0,173,0,192,16,251,141,16,192,201,141,96′xÉ4,43140,83,72,79,215ËÇÉ2,43
273,32,48ÔåÉ0   ñáN:≠NƒáA:ÅI–1¡N:áD:πA»I…1,D:Ç:´150

```
=============================================================================
DOCUMENT :AAL-8207:DOS3.3:S.FILEDUMP.txt
=============================================================================

 1000  *--------------------------------
 1010  *      "FILEDUMP" COMMAND
 1020  *--------------------------------
 1030  DOS.OPEN.TEXT.FILE   .EQ $A2A3
 1040  DOS.CLOSE.FILE       .EQ $A2FC
 1050  DOS.READ.ONE.BYTE    .EQ $A68C
 1060  MON.COUT1            .EQ $FDF0
 1070  *--------------------------------
 1080         .OR $BCDF
 1090         .TA $8DF
 1100  FILEDUMP
 1110         JSR DOS.OPEN.TEXT.FILE
 1120         LDA #$8D
 1130  .1     JSR MON.COUT1
 1140         JSR DOS.READ.ONE.BYTE
 1150         BNE .1       PRINT IT
 1160         JMP DOS.CLOSE.FILE
 1170  *--------------------------------
 1180         .OR $A8E7
 1190         .TA $8E7
 1200         .AT /FILEDUMP/   NAME OF FILEDUMP COMMAND
 1210  *--------------------------------
 1220         .OR $9D48
 1230         .TA $848
 1240         .DA FILEDUMP-1   BRANCH FOR FILEDUMP COMMAND
 1250  *--------------------------------
 1260         .OR $A933
 1270         .TA $833
 1280         .HS 2030     FILENAME REQUIRED, SLOT & DRIVE
 1290  *                   ARE OPTIONAL
```

```
==========================================================================
DOCUMENT :AAL-8207:DOS3.3:S.GIANT.MACRO.txt
==========================================================================

 1000  *--------------------------------
 1010  * MACRO: >CRT SRC,DSPMODE,MSG
 1020  *         >CRT SRC,DSPMODE,MSG,REPMODE
 1030  * MACRO: >CRT SRC,DSPMODE,MSG,REPMODE,REPADDR
 1040  *--------------------------------
 1050         .MA CRT
 1060         LDY #0       INITIALIZE INDEX
 1070         .DO ']1='L   *** LITERAL MESSAGE ***
 1080  :1     LDA :9,Y     GET MESSAGE CHARACTER
 1090         .ELSE        *** ADDRESSED MESSAGE ***
 1100  :1     LDA ]3,Y     GET MESSAGE CHARACTER
 1110         .FIN
 1120         PHA          SAVE CHARACTER ON STACK
 1130  *--------------------------------
 1140         .DO ']2='N   *** NORMAL DISPLAY ***
 1150         ORA #$80     SET TOP BIT OF CHARACTER
 1160         .ELSE        *** INVERSE DISPLAY ***
 1170         AND #$3F
 1180         .FIN
 1190  *--------------------------------
 1200         JSR $FDF0    DISPLAY CHARACTER
 1210         INY          POINT TO NEXT CHARACTER
 1220         PLA          GET ORIGINAL CHARACTER
 1230         BMI :1       MORE IF TOP BIT = 1
 1240  *--------------------------------
 1250         .DO ']1='L   *** LITERAL ***
 1260         BPL :2       ...ALWAYS
 1270  :9     .AT -']3'    MESSAGE ITSELF
 1280  :2
 1290         .FIN
 1300  *--------------------------------
 1310         .DO ]#=3     *** DISPLAY ONLY ***
 1320         LDA #$8D     CARRIAGE RETURN
 1330         JSR $FDF0
 1340         .ELSE
 1350         .DO ']4='R   *** STRING REPLY EXPECTED ***
 1360         LDA #$8D     CARRIAGE RETURN
 1370         JSR $FDF0
 1380         JSR $FD6F    READ REPLY
 1390         .DO ]#=5     *** SPECIFY REPLY LOCATION ***
 1400         LDY #0
 1410  :3     LDA $200,Y   MOVE REPLY TO CALLER'S BUFFER
 1420         STA ]5,Y
 1430         INY
 1440         CMP #$8D     WAS IT END OF LINE?
 1450         BNE :3       MORE TO MOVE
 1460         .FIN
 1470         .ELSE
 1480         LDA #$A0     ADD ONE BLANK TO MESSAGE
```

```
1490         JSR $FDF0
1500  :5     JSR $FD0C    GET REPLY CHARACTER
1510         .DO ']4='Y   *** Y/N REPLY ***
1520         CMP #'Y+$80
1530         BEQ :6
1540         CMP #'N+$80
1550         BNE :5       NEITHER Y NOR N
1560  :6
1570         .FIN
1580         PHA          SAVE REPLY
1590         JSR $FDF0    DISPLAY THE CHARACTER
1600         LDA #$8D
1610         JSR $FDF0    CARRIAGE RETURN
1620         PLA          GET REPLY CHARACTER
1630         .DO ']4='Y   *** Y/N REPLY ***
1640         CMP #'Y+$80  .EQ. IF "Y", .NE. IF "N"
1650         .FIN
1660         .FIN
1670         .FIN
1680         .EM
1690  *--------------------------------
1700         >CRT L,N,"ABCDEFG"
1710         >CRT L,I,"ABCDEFG"
1720         >CRT A,N,MSG
1730         >CRT A,I,MSG
1740  *--------------------------------
1750         >CRT L,N,"ABCDEFG",Y
1760         >CRT L,I,"ABCDEFG",I
1770         >CRT A,N,MSG,R
1780         >CRT A,I,MSG,R,BUFFER
1790         RTS
1800  MSG    .AT -/MESSAGE/
1810  BUFFER .BS 256
```

```
==============================================================================
DOCUMENT :AAL-8207:DOS3.3:S.SHOW.txt
==============================================================================

 1000  *--------------------------------
 1010  *      "SHOW" COMMAND
 1020  *--------------------------------
 1030  DOS.OPEN.TEXT.FILE  .EQ $A2A3
 1040  DOS.CLOSE.FILE      .EQ $A2FC
 1050  DOS.READ.ONE.BYTE   .EQ $A68C
 1060  KEYBOARD            .EQ $C000
 1070  STROBE              .EQ $C010
 1080  MON.COUT1           .EQ $FDF0
 1090  *--------------------------------
 1100         .OR $A54F
 1110         .TA $84F
 1120  SHOW
 1130         JSR DOS.OPEN.TEXT.FILE
 1140         LDA #$8D
 1150  .1     JSR MON.COUT1
 1160         JSR PAUSE.CHECK
 1170         BEQ .2
 1180         JSR DOS.READ.ONE.BYTE
 1190         BNE .1       PRINT IT
 1200  .2     JMP DOS.CLOSE.FILE
 1210  *--------------------------------
 1220  *      RETURN .EQ. IF ABORT
 1230  *             .NE. IF CONTINUE
 1240  *--------------------------------
 1250         .OR $AE8E         OVER "INIT" CODE
 1260         .TA $88E
 1270  PAUSE.CHECK
 1280         LDA KEYBOARD      ANY KEY PRESSED?
 1290         BPL .2            NO, CONTINUE
 1300         STA STROBE        YES, CLEAR STROBE
 1310         CMP #$8D          ABORT?
 1320         BEQ .2            YES, RETURN .EQ. STATUS
 1330  .1     LDA KEYBOARD      NO, PAUSE TILL KEYPRESS
 1340         BPL .1            NONE PRESSED YET
 1350         STA STROBE        CLEAR STROBE
 1360         CMP #$8D          ABORT?
 1370  .2     RTS               .EQ. IF ABORT
 1380  *--------------------------------
 1390         .OR $A884
 1400         .TA $884
 1410         .AT /SHOW/   SHOW COMMAND NAME
 1420  *--------------------------------
 1430         .OR $A909
 1440         .TA $809
 1450         .HS 2030     FLAGS FOR SHOW COMMAND
 1460  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8207:DOS3.3:S.Smpl.Anim.txt
==============================================================================

 1000  *SAVE S.SIMPLE ANIMATION
 1010  *--------------------------------
 1020  * SIMPLE ANIMATION
 1030  *--------------------------------
 1040  MON.WAIT    .EQ $FCA8    MONITOR DELAY ROUTINE
 1050  *--------------------------------
 1060  T1          .EQ $0,1
 1070  T2          .EQ $2,3
 1080  T3          .EQ $4,5
 1090  Y.INDEX     .EQ $6,7
 1100  *--------------------------------
 1110  * ANIMATION PLAYBACK LOCATIONS
 1120  *--------------------------------
 1130  HIRES.PAGE  .EQ $20      $20 = PAGE 1, $40 = PAGE 2
 1140  Y.COORD     .EQ 100      WHERE TO PUT ANIMATION
 1150  X.COORD     .EQ 20       WHERE TO PUT ANIMATION
 1160  *--------------------------------
 1170         .OR $803
 1180         .TF B.ANIMATE
 1190  *--------------------------------
 1200  START  JSR HIRES.INIT    INITIALIZE HIRES SCREEN
 1210  .1     JSR PLAY.FRAMES   PLAY 1 SET OF FRAMES
 1220         JMP .1            GO DO IT AGAIN
 1230  *--------------------------------
 1240  PLAY.FRAMES LDA #0       INIT FRAME INDEX
 1250         STA FRAME.INDEX
 1260  .1     LDA FRAME.INDEX   GET FRAME INDEX POINTER
 1270         CMP #NUM.FRAMES   HAVE ALL FRMES BEEN DONE
 1280         BEQ .3            YES, SO RETURN
 1290         LDY $C000    HAS A KEY BEEN PRESSED
 1300         BPL .2       NO, SO KEY PLAYING THE FRAMES
 1310         LDA $C051    RESTORE TEXT SCREEN
 1320         LDA $C054    PRIMARY PAGE
 1330         JMP $3D0     EXIT ON ANY KEY
 1340  .2     ASL          DOUBLE INDEX
 1350         TAY
 1360         LDA FRAME.TABLE,Y  GET TABLE ADDRESS
 1370         STA T1       SAVE ADRS IN T1
 1380         INY          NEXT BYTE OF ADRS
 1390         LDA FRAME.TABLE,Y
 1400         STA T1+1
 1410         JSR ANIMATE  MOVE FRAME DATA TO SCREEN
 1420         INC FRAME.INDEX NEXT FRAME
 1430         BNE .1       ...ALWAYS
 1440  .3     RTS
 1450  *--------------------------------
 1460  HIRES.INIT LDA #HIRES.PAGE
 1470         STA T1+1
 1480         LDY #0
```

```
1490          STY T1
1500  .0      TYA           ZERO A REG
1510  .1      STA (T1),Y    CLEAR SCREEN PAGE
1520          INY
1530          BNE .1
1540          INC T1+1      NEXT PAGE
1550          LDA T1+1      CHECK FOR
1560          AND #$1F      END OF HIRES PAGE
1570          BNE .0        NO, CLEAR MORE
1580          LDA $C050     ENABLE GRAPHICS
1590          LDA $C057     ENABLE HIRES
1600          LDA $C054     ENABLE PAGE 1 (C055 IS PAGE 2)
1610          LDA $C052     NOMIX
1620          RTS
1630  *--------------------------------
1640  INTER.FRAME.DELAY     .DA #20
1650  XSIZE   .DA #4         X FRAME SIZE IN BYTES
1660  YSIZE   .DA #24        Y FRAME SIZE IN BYTES
1670  FRAME.TABLE
1680          .DA FRAME1
1690          .DA FRAME2
1700          .DA FRAME3
1710          .DA FRAME4
1720          .DA FRAME5
1730          .DA FRAME6
1740          .DA FRAME7
1750          .DA FRAME8
1760          .DA FRAME9
1770  NUM.FRAMES .EQ 9
1780  FRAME.INDEX .DA #0
1790  *--------------------------------
1800  ANIMATE LDA #Y.COORD     THIS IS THE STARTING ROW
1810          STA Y.INDEX       FOR THE ANIMATION
1820          LDY YSIZE     NUMBER OF ROWS TO PUT ON SCREEN
1830          STY T2
1840  .1      LDY Y.INDEX
1850          LDA YTBL.LO,Y     COMPUTE THE ROW ADRS
1860          CLC
1870          ADC #X.COORD      ADD THE X OFFSET
1880          STA T3
1890          LDA YTBL.HI,Y
1900          ADC #HIRES.PAGE   ADD THE HIRES PAGE BITS
1910          STA T3+1          T3 POINTS TO ROW POSITION
1920          LDY XSIZE         NUMBER OF BYTES TO PUT INTO ROW
1930          DEY               INDEX BEGINS AT ZERO TO XSIZE-1
1940  .3      LDA (T1),Y        GET FRAME DATA
1950          STA (T3),Y        PUT ONTO SCREEN
1960          DEY               FOR ALL BYTES IN THE ROW
1970          BPL .3
1980  .4      INC Y.INDEX       NEXT ROW INDEX
1990          LDA T1
2000          CLC
2010          ADC XSIZE         STEP FRAME ADRS AHEAD
2020          STA T1            TO NEXT ROW OF DATA
```

```
2030        LDA T1+1
2040        ADC #0
2050        STA T1+1
2060        DEC T2            COUNT DOW THE ROWS
2070        BNE .1            GO MOVE REST OF FRAME ROWS
2080        LDY INTER.FRAME.DELAY
2090        BEQ .6            NO DELAY BETWEEN FRAMES
2100        STY T2            SAVE DELAY
2110 .5     LDA #30           REPEAT THIS SMALL DELAY
2120        JSR MON.WAIT
2130        DEC T2            FOR COUNT IN 'T2'
2140        BNE .5            MORE DELAY
2150 .6     RTS               FRAME IS ALL DONE
2160 *--------------------------------
2170 * HIRES Y OFFSET TABLES
2180 * OFFSET FROM $2000 OR $4000
2190 * HIRES PAGE DISPLAYS
2200 * USING THESE TABLES SPEEDS UP
2210 * HIRES SCREEN ADRS COMPUTATION
2220 * A GREAT DEAL!
2230 *
2240 * FOR EVERY Y VALUE FROM 0-191
2250 * THERE IS AN ENTRY IN THIS TABLE
2260 * TO COMPUTE THE ADRS OF FIRST
2270 * BYTE IN THE ROW.
2280 *--------------------------------
2290 YTBL.LO   .EQ *
2300  .HS 00000000000000008080808080808080
2310  .HS 00000000000000008080808080808080
2320  .HS 00000000000000008080808080808080
2330  .HS 00000000000000008080808080808080
2340  .HS 2828282828282828A8A8A8A8A8A8A8A8
2350  .HS 2828282828282828A8A8A8A8A8A8A8A8
2360  .HS 2828282828282828A8A8A8A8A8A8A8A8
2370  .HS 2828282828282828A8A8A8A8A8A8A8A8
2380  .HS 5050505050505050D0D0D0D0D0D0D0D0
2390  .HS 5050505050505050D0D0D0D0D0D0D0D0
2400  .HS 5050505050505050D0D0D0D0D0D0D0D0
2410  .HS 5050505050505050D0D0D0D0D0D0D0D0
2420 YTBL.HI   .EQ *
2430  .HS 0004080C1014181C0004080C1014181C
2440  .HS 0105090D1115191D0105090D1115191D
2450  .HS 02060A0E12161A1E02060A0E12161A1E
2460  .HS 03070B0F13171B1F03070B0F13171B1F
2470  .HS 0004080C1014181C0004080C1014181C
2480  .HS 0105090D1115191D0105090D1115191D
2490  .HS 02060A0E12161A1E02060A0E12161A1E
2500  .HS 03070B0F13171B1F03070B0F13171B1F
2510  .HS 0004080C1014181C0004080C1014181C
2520  .HS 0105090D1115191D0105090D1115191D
2530  .HS 02060A0E12161A1E02060A0E12161A1E
2540  .HS 03070B0F13171B1F03070B0F13171B1F
2550 *--------------------------------
2560 * ANIMATION DATA
```

```
2570  *--------------------------------
2580  FRAME1
2590    .HS 000000000000000000600300
2600    .HS 0070070000580D0000780F00
2610    .HS 00380E000070070000600300
2620    .HS 004001000040010000780F00
2630    .HS 007C1F000066330000436100
2640    .HS 006363000073670040714701
2650    .HS 40394E0100180C0000180C00
2660    .HS 00180C0000180C0000180C00
2670  FRAME2
2680    .HS 000000000060030000700700
2690    .HS 00580D0000780F0000380E00
2700    .HS 007007000060030000400100
2710    .HS 0040010000780F00007C1F00
2720    .HS 00663300404141016C60031B
2730    .HS 3C70071E0070070000380E00
2740    .HS 00180C00000C1800000C1800
2750    .HS 000C1800000C180000000000
2760  FRAME3
2770    .HS 006003000070070000580D00
2780    .HS 00780F0000380E0000700700
2790    .HS 066003300640013000C400118
2800    .HS 78780F0E607F7F0740677301
2810    .HS 004001000060030000700700
2820    .HS 00700700001C1C00000C1800
2830    .HS 000C1800000C180000063000
2840    .HS 000630000000000000000000
2850  FRAME4
2860    .HS 000000004061430140714701
2870    .HS 60580D0320780F0220380E02
2880    .HS 607007034061430100414100
2890    .HS 00463100007E3F0000780F00
2900    .HS 006003000040010000600300
2910    .HS 00700700007007000001C1C00
2920    .HS 000630000003600000036000
2930    .HS 400140014001400100000000
2940  FRAME5
2950    .HS 000000000000000040610303
2960    .HS 4071070320580D0220780F02
2970    .HS 60380E036070070340614301
2980    .HS 00436100004631000007E3F00
2990    .HS 00780F000060030000400100
3000    .HS 006003000070070000700700
3010    .HS 001C1C000006300000036000
3020    .HS 000360004001400140014001
3030  FRAME6
3040    .HS 000000004061430140714701
3050    .HS 60580D0320780F0260380E03
3060    .HS 607007034061430100436100
3070    .HS 00463100007E3F0000780F00
3080    .HS 006003000040010000600300
3090    .HS 00700700007007000001C1C00
3100    .HS 000630000003600000036000
```

```
3110    .HS 4001400140014001000000000
3120  FRAME7
3130    .HS 4061430140714701605 80D03
3140    .HS 20780F0220380E0260700703
3150    .HS 406143010043610000463100
3160    .HS 007E3F0000780F0000600300
3170    .HS 004001000060030000700700
3180    .HS 00700700001C1C0000063000
3190    .HS 000360000003600040014001
3200    .HS 400140010000000000000000
3210  FRAME8
3220    .HS 006003000070070000580D00
3230    .HS 00780F0000380E0000700700
3240    .HS 0660033006400130 0C400118
3250    .HS 78780F0E607F7F0740677301
3260    .HS 004001000060030000700700
3270    .HS 00700700001C1C00000C1800
3280    .HS 000C1800000C180000063000
3290    .HS 000630000000000000000000
3300  FRAME9
3310    .HS 000000000060030000700700
3320    .HS 00580D0000780F0000380E00
3330    .HS 007007000060030000400100
3340    .HS 0040010000780F00007C1F00
3350    .HS 006633004041410 16C60031B
3360    .HS 3C70071E0070070000380E00
3370    .HS 00180C00000C1800000C1800
3380    .HS 000C1800000C180000000000
```

```
==============================================================================
DOCUMENT :AAL-8207:DOS3.3:S.ZP.InOrder.txt
==============================================================================

 1000  *SAVE S.PAGE-ZERO IN ORDER
 1010  *--------------------------------
 1020  *       SEARCH FOR PAGE ZERO REFERENCES
 1030  *       (MODIFIED BY TRACY SHAFER)
 1040  *--------------------------------
 1050  MON.A1L    .EQ $3C
 1060  MON.A1H    .EQ $3D
 1070  MON.A2L    .EQ $3E
 1080  MON.A2H    .EQ $3F
 1090  MON.PCL    .EQ $3A
 1100  MON.PCH    .EQ $3B
 1110  *--------------------------------
 1120  KEYBOARD   .EQ $C000
 1130  STROBE     .EQ $C010
 1140  *--------------------------------
 1150  MON.LIST2  .EQ $FE63
 1160  MON.INSDS  .EQ $F88C
 1170  MON.A1PC   .EQ $FE75
 1180  MON.PCADJ  .EQ $F953
 1190  MON.NXTA1  .EQ $FCBA
 1200  *--------------------------------
 1210  *       SET UP CONTROL-Y VECTOR
 1220  *--------------------------------
 1230  SETUPY LDA #$4C     'JMP' OPCODE
 1240         STA $3F8
 1250         LDA #CTRL.Y
 1260         STA $3F9
 1270         LDA /CTRL.Y
 1280         STA $3FA
 1290         RTS
 1300  *--------------------------------
 1310  PAGE.REF   .HS 00   VARIABLE TO HOLD THE CURRENT
 1320  *                   PAGE-ZERO LOCATION
 1330  *--------------------------------
 1340  *       CONTROL-Y COMES HERE
 1350  *--------------------------------
 1360  CTRL.Y
 1370         JSR MON.A1PC IF ADDRESS SPECIFIED, PUT IN PC
 1380  .1     LDY #0
 1390         LDA (MON.PCL),Y
 1400         AND #$0F
 1410         CMP #1
 1420         BEQ .3
 1430         CMP #4
 1440         BCC .6
 1450         BNE .2
 1460         LDA (MON.PCL),Y
 1470         AND #$F0
 1480         CMP #$20     BIT Z
```

```
1490        BEQ .3
1500        CMP #$80
1510        BCC .6       NO
1520        CMP #$D0
1530        BEQ .6       NO
1540        CMP #$F0
1550        BEQ .6       NO
1560        BNE .3       YES
1570  .2    CMP #7
1580        BCS .6
1590  *--------------------------------
1600  *     INSTRUCTION REFERENCES PAGE-ZERO
1610  *--------------------------------
1620  .3    INY
1630        LDA (MON.PCL),Y   GET PAGE REFERENCE
1640        DEY          RESTORE VALUE OF Y
1650        CMP PAGE.REF   ONE WE ARE SEARCHING FOR?
1660        BNE .6         NO, IGNORE THIS TIME
1670        LDA #1       DISASSEMBLE THIS ONE INSTRUCTION
1680        JSR MON.LIST2   DISASSEMBLE
1690        LDA KEYBOARD    SEE IF KEYPRESS
1700        BPL .7       NO
1710        STA STROBE   YES, CLEAR IT
1720        CMP #$8D
1730        BEQ .5
1740  .4    LDA KEYBOARD
1750        BPL .4
1760        STA STROBE
1770        CMP #$8D
1780        BNE .7
1790  .5    RTS
1800  *--------------------------------
1810  *     DOES NOT REFERENCE PAGE-ZERO
1820  *--------------------------------
1830  .6    LDX #0
1840        JSR MON.INSDS    GET LENGTH OF INSTRUCTION
1850        JSR MON.PCADJ
1860        STA MON.PCL
1870        STY MON.PCH
1880  *--------------------------------
1890  *     TEST IF FINISHED
1900  *--------------------------------
1910  .7    LDA MON.PCL
1920        CMP MON.A2L
1930        LDA MON.PCH
1940        SBC MON.A2H
1950        BCC .1
1960        LDX #1       RESTORE X-VALUE FOR MON.A1PC ABOVE
1970        INC PAGE.REF NEXT PAGE
1980        BNE CTRL.Y   NOT FINISHED
1990        RTS
```

==============================================================================
DOCUMENT :AAL-8208:Articles:AGAG.Review.txt
==============================================================================

Review of "Apple Graphics & Arcade Game Design"

If you are at all interested in Apple graphics, or writing animated hi-res games, this book is for you. Jeffrey Stanton, the author, may already be known to you. He is the editor of "The Book of Apple Software, and also has several Apple arcade games on the market. "Apple Graphics & Arcade Game Design" (AGAG) is 288 pages long, and retails for $19.95. (I am selling it for $18 plus shipping.) A coupon in the back enables you to purchase all of the source code shown in the book on diskette for only $15.

There are two parts to the book: first, a thorough explanation of Apple graphics, with numerous examples in both Applesoft and assembly language; second, design and programming of all the parts of a working arcade game.

AGAG is written for the advanced Applesoft or beginning assembly language programmer. You learn about both lo-res and hi-res graphics at the assembly language level. You learn the fundamentals, and then proceed to program scene scrolling, page flipping, laser fire, bomb drops, explosions, scoring, and paddle control routines. Sorry, nothing much about sound generation.

AGAG's pages are divided into 8 chapters as follows:

```
1. ( 25 pages) Applesoft Hi-Res
2. ( 34 pages) Lo-Res Graphics
3. ( 17 pages) Machine Language Access to Applesoft
               Hi-Res Routines
4. ( 23 pages) Hi-Res Screen Architecture
5. ( 36 pages) Bit-Mapped Graphics
6. ( 90 pages) Arcade Graphics
7. ( 44 pages) Games that Scroll
8. (  5 pages) What Makes a Good Game
```

I noticed a few errors in the book: on page 149, flow chart lines are incorrectly drawn; on page 284, there is a large block of repeated text, and therefore possibly a missing block which should have been in the space. The word "initialize" is always incorrectly spelled "initilize". The index is very brief, only about 70 lines long; I believe it should be about 3 or 4 times longer to really help in locating items of interest.

Jeff does not seem to know about the existence of the S-C Macro Assembler. He repeatedly mentions the TED, Big-Mac, Merlin Assemblers, and occasionally refers to Lisa and DOS ToolKit. All the listings are in the Big-Mac format. You should have no trouble adapting them to the S-C format.

AGAG is an excellent tutorial, and includes many useful programs and ideas for anyone interested in Apple graphics. I heartily recommend the book, ranking it just under "Beneath Apple DOS" in importance and utility.

==============================================================================
DOCUMENT :AAL-8208:Articles:Auto.Man.Toggle.txt
==============================================================================

AUTO-MANUAL Toggle for S-C Macro Assembler........R. F. O'Brien

Here is a small program to accompany Bill Morgan's Automatic Catalog in the June '82 issue of AAL. This routine adds an AUTO/MANUAL command toggle to the S-C Macro Assembler. Using CTRL-A when the cursor is at the beginning of a line enters the AUTO line numbering mode and waits for input of a line number and/or RETURN. Entering another CTRL-A while in AUTO mode and at the start of a line executes a MANUAL command.

In addition, I have added some code to provide slow and fast listings at a single keypress. CTRL-S does a SLOW LIST command, which is cancelled by a 'RETURN' during listing. CTRL-L will provide a listing at normal speed (assuming the slow list has been cancelled.)

The patch is implemented as follows:

```
1. Enter the S-C Macro Assembler
2. :$101D:33 N 1000G
3. :BLOAD AUTO/MANUAL PATCH
4. :$138D: 4C 28 32        (JMP PATCH instead of JSR BELL)
5. :BSAVE AUTO/MAN S-C MACRO ASM,A$1000,L$2300
```

Note: You may omit step 2 if you have already installed Bill's automatic CATALOG.

================================================================================
DOCUMENT :AAL-8208:Articles:Cursor.Routine.txt
================================================================================

Blinking Underline Cursor Routine..................Bill Linn

Early users of the ES-CAPE Applesoft Editing system (formerly known as AED II) have really come to appreciate the blinking underline cursor -- it simply doesn't tire the eyes as much as the standard flashing blank does. With the following subroutine, you can add this special touch to your own assembly language or BASIC programs!

The subroutine hooks into the monitor keyboard input vector at $38 and $39. Each time the monitor RDKEY subroutine is called, my KEYIN subroutine gets control. If the character on the screen at the cursor position is not an underline, I alternate the display of an underline and the original character every 1/4 second. If the original character was an underline, I alternate it with a blank. (If I alternate an underline with an underline, it is difficult to see anything happen!)

Lines 1210-1250 store the KEYIN subroutine's address in the keyboard input vector. When a request for a key press is made by an Applesoft INPUT command, for example, we get control at line 1270. The A-register has the current screen character. I save the A- and X-registers, because KEYIN must exit with the original values unchanged.

Lines 1290-1320 test the current screen character to see whether it is already an underline or not. If it is, I use a blank for the alternating character. Otherwise, I use the original screen contents for an alternating character. I push the alternating character onto the stack.

Lines 1330-1500 do the alternating. I look at the character on the screen: if it is an underline, I substitute the alternating character; if not, I store an underline. The lines 1430-1500 delay for about 1/4 second before the next alternation. If a keypress occurs, the loop ends by branching to ".5" at line 1540. You may wish to vary the blink rate by changing the value loaded into the Y-register at line 1430.

When a key is pressed we end up at line 1540, where I pop the alternating character off the stack. The I call the monitor bell subroutine for a short (10 half-cycles) bell. This makes an audible "click" for user feedback. (If you don't appreciate clicking keyboards, just delete lines 1550 and 1560.) Then I restore the Y-, X-, and A-registers to their orignal values, and jump into the monitor's KEYIN subroutine at $FD26. The monitor restores the original character to the screen, and returns with the keypress value in the accumulator.

I have set the subroutine origin to $300, but you can assemble it anywhere you like. In fact, it will run anywhere you put without reassembly, just so you load the correct address into $38 and $39 in the HOOK routine.

After assembly, assuming it is origined at $300, you can BSAVE it with "BSAVE B.UNDERLINE,A$300,L$3C. Then to activate this routine from Applesoft, just BRUN the file B.UNDERLINE. All keyboard input through the standard monitor RDKEY subroutine ($FDOC) or Applesoft GET and INPUT statements will be prompted by the underline cursor. An "IN#0" will restore the familiar flashing blank. Have fun!

```
==========================================================================
DOCUMENT :AAL-8208:Articles:Free.Space.txt
==========================================================================
```

Free Space Patch For the S-C Assembler...........Mike Sanders

Volume 5, Number 6 of Call A.P.P.L.E. has an article giving a DOS patch to replace the volume number printed during catalog with number of free sectors remaining on the disk.

The routine as published works for both Applesoft and Integer BASIC, but does not work with the language card version of the S-C Assembler. Only a few changes were needed to make it work with all three.

A call to Bob gave me the location of the decimal print routine in the S-C Macro Assembler, Language Card Version.

The original code as published in CAll A.P.P.L.E. checked location $E006 to see what language is in use. My code looks at $E001, which has a different value in each of the three:

| Language | $E001 |
|---:|---|
| Applesoft: | $28 |
| Integer BASIC: | $00 |
| S-C Macro Assembler: | $94 |

The code in lines 1320-1370 checks which language is in use and jumps to the right routine. I also changed the zero page locations used to count the number of free sectors because the S-C Assembler print routine expects the two-byte value to be in $D3 and $D4.

The rest of the code works as explained in the Call A.P.P.L.E. article. I refer you to it for more details and as an excellent lesson on reducing the size of code.

Install the two patches to DOS by BLOADing the two binary files FREE.SECTORS.1 and FREE.SECTORS.2. The type CATALOG to see the how many free sectors you have.

=====================================================================
DOCUMENT :AAL-8208:Articles:Front.Page.txt
=====================================================================

$1.50

Volume 2 -- Issue 11                                        August, 1982

In This Issue...

=============================================================================
DOCUMENT :AAL-8208:Articles:Large.Src.Files.txt
=============================================================================

Large Source Files and the S-C Macro Assembler......Bill Morgan

One of the more common questions we get is: "How do I best use the .IN and .TF directives to handle very large programs?"

The main technique we use is the Assembly Control File (ACF), a short source file which is mostly made up of .IN statements to call the other modules. Here is an example, called SAMPLE.ACF:

```
1000        .IN SAMPLE.EQUATES
1010        .PG
1020        .IN SAMPLE.CODE.1
1030        .PG
1040        .IN SAMPLE.CODE.2
1050        .PG
1060        .IN SAMPLE.DATA
1070        .PG
```

SAMPLE.EQUATES is all the definitions for the program, SAMPLE.CODE.1 and SAMPLE.CODE.2 are the main body of the program, and SAMPLE.DATA contains all the variables and ASCII text. When you want to assemble the program, just LOAD SAMPLE.ACF and type MON C then ASM. The Macro Assembler will load each file and assemble it, in the order they are listed in the ACF. The "MON C" shows you the "LOAD file name" for each file, helping you to tell what's where.

Using this technique, a program can conveniently be broken into as many modules as you want, and can be as large as you want. The Macro Assembler itself is 26 source files on two disks! To spread the files across more than one disk, just add drive (and/or slot) specifiers to all the file names.

You can also use the ACF to do global search-and-replace operations on the entire program. Here are the commands to search SAMPLE for all occurences of the label MON.COUT:

```
:LOAD SAMPLE.ACF
:REP /       .IN/LOAD/A
:REP /       .PG/FIND "MON.COUT"/A
:TEXT COUT.SEARCH
:MON I
:EXEC COUT.SEARCH
```

This converts SAMPLE.ACF into an EXEC file that will list each occurence of "MON.COUT" in every module of the program. Here's what the file looks like now:

```
1000 LOAD SAMPLE.EQUATES
1010 FIND "MON.COUT"
```

```
1020 LOAD SAMPLE.CODE.1
1030 FIND "MON.COUT"
1040 LOAD SAMPLE.CODE.2
1050 FIND "MON.COUT"
1060 LOAD SAMPLE.DATA
1070 FIND "MON.COUT"
```

The ACF is also a good place for the .OR and .TF statements, comments about the assembly process, and any condition flags.  Here is a more complicated version of SAMPLE.ACF:

```
1000 *--------------------------------
1010 *      SAMPLE FILE TO DEMONSTRATE ACF
1020 *--------------------------------
1030 LC.FLAG .EQ 0  =0 IF UPPER CASE ONLY
1040 *              =1 IF LOWER CASE VERSION
1050 *--------------------------------
1060        .OR $803
1070        .DO LC.FLAG
1080        .TF B.SAMPLE.LC
1090        .ELSE
1100        .TF B.SAMPLE.UC
1110        .FIN
1120 *--------------------------------
1130        .IN SAMPLE.EQUATES
1140        .PG
1150        .IN SAMPLE.CODE
1160        .PG
1170        .DO LC.FLAG
1180        .IN SAMPLE.LOWER.CASE.ROUTINES
1190        .PG
1200        .ELSE
1210        .IN SAMPLE.NORMAL.ROUTINES
1220        .PG
1230        .FIN
1240        .IN SAMPLE.DATA
1250        .PG
```

To use this ACF, just LOAD it, EDIT line 1030 to set LC.FLAG to 0 or 1, set MON C, and ASM.  The Macro Assembler will load the appropriate source files for the version you want and direct the object code to the correct target file.  To turn this ACF into an EXEC file for searching, you must delete lines 1000-1120, 1170, 1200, and 1230 before doing the REP commands.

For more information on the .IN and .TF directives, see pages 4-6 and 5-3/4 in the Macro Assembler manual.  Conditional assembly is discussed on pages 5-9/10 and in chapter 7.

```
==============================================================================
DOCUMENT :AAL-8208:Articles:Macro.LC.Patch.txt
==============================================================================
```

Patch for S-C Macro Assembler...........Bob Sander-Cederlof

When I added the lower-case options to the S-C Macro Assembler, I overlooked the fact that within .AS and .AT strings, and in ASCII literal constants, you would want lower case codes to be assembled. The assembler as it now is converts all lower case codes to upper case during assembly. For example, ".AS /Example/" would assemble all upper case ASCII, just as though you had written ".AS /EXAMPLE/"

The following patches will correct this problem, allowing you to specify lower case strings and constants when you wish.

```
$2961:EA EA EA EA EA EA

$31B8<1235.124BM

$1074:B8 31
$118C:B8 31
$11B2:B8 31
$187F:B8 31
$23FA:B8 31

$31CF:C8 84 7B C9 60 90 04 29 5F 85 61 60

$1240:20 CF 31

BSAVE ASM.WITH.LC.IN.AS,A$1000,L$21DB
(or whatever file name you wish)
```

The patches above are for the version which runs in mother-board RAM. The Language card version has different addresses, and you must first write-enable the language card. Assuming you are currently running the language card version, perform the patch as follows:

```
$C083 C083
$EAAD:EA EA EA EA EA EA

$F304<D235.D24BM

$D074:04 F3
$D18C:04 F3
$D1B2:04 F3
$D87F:04 F3
$E546:04 F3

$F31B:C8 84 7B C9 60 90 04 29 5F 85 61 60

$D240:20 1B F3
```

BSAVE LC.ASM.WITH.LC.IN.AS,A$D000,L$2327
(or whatever file name you wish)

Be aware that the above patches may conflict with other patches you may already have applied to your copy of the assembler. If you have already used the area from $31B8 through $31DB, or $F304 through $F326, you will need to use a different area and change the references accordingly.

```
==============================================================================
DOCUMENT :AAL-8208:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler...............................................$80.00
S-C Macro Cross Assembler Modules
    6800/6801/6802 Version........................................$32.50
    6809 Version..................................................$32.50
    Z-80 Version..................................................$32.50
    Requires ownership of S-C Macro Assembler.
    Each disk includes regular and language card versions.

S-C ASSEMBLER II Version 4.0......................................$55.00
Upgrade from Version 4.0 to MACRO.................................$27.50
Source code of Version 4.0 on disk................................$95.00
    Fully commented, easy to understand and modify to your own tastes.

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)......(regular $79) $49.00
    Special price to AAL readers only, until 9/1/82!
Source Code for FLASH! Runtime Package.............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00

ES-CAPE:  Extended S-C Applesoft Program Editor...................$40.00

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00

Ashby Shift-Key Mod...............................................$15.00
Paymar Lower-Case Adapter.........................................$37.50
    For Apples before Revision 7 only
Lower-Case Display Encoder ROM....................................$25.00
    Works only Revision level 7 Apples.  Replaces the encoder ROM.

Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
    Corrugated folder specially designed for mailing mini-floppy
    diskettes.  Fits in standard 6x9-inch envelope.  (Envelopes
    5-cents each, if you need them.)

Books, Books, Books.........................compare our discount prices!
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", William Leubert.........($14.95)  $14.00
    "6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
```

```
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00
```

================================================================
DOCUMENT :AAL-8208:Articles:Quick.DOS.Write.txt
================================================================

Quick Way to Write DOS on a Disk..............Bob Perkins
Tussy, OK 73088

I just received the July AAL and liked the little article on the "FILEDUMP" command. I had already done just about the same thing.

In fact, I make a lot of changes to DOS. Too many to POKE in every time I boot up. So I started looking around for a simple way to replace the DOS image on a disk without disturbing the programs already on it, and without using MASTER.CREATE. The July Call A.P.P.L.E. had a program to do it, only it seems much more complicated than my solution.

I used the S-C Macro Assembler to create a text file like this:

```
:1000  LOAD HELLO
:1010  POKE -21921,0:POKE -18448,0:POKE
-18447,157:POKE-18453,0:CALL-18614
:TEXT WRITE.DOS
```

Note the leading blank before the LOAD and the first POKE. It is there to leave room for Applesoft's "]" prompt.

Whenever I want to write the DOS image on a disk, I use the SHOW command to list out WRITE.DOS, and then trace over the two command lines from Applesoft. Presto-Changeo, a new copy of DOS goes out to the disk. I suppose you could even EXEC it, though I prefer to trace over it and haven't tried EXECing.

The LOAD HELLO is there to get the boot file name into DOS's filename buffer. You can use whatever filename you want, of course. POKE-21921 tell DOS that the last command was an INIT for its startup procedure (i.e. AA5F:00). POKE-18448 and -18447 start the write at 9D00 (B7F0:00 9D). POKR -18453 sets the expected volume number to zero, so a match to any volume will occur (B7EB:00). The CALL is to the "write DOS image" code inside DOS.

==============================================================================
DOCUMENT :AAL-8208:Articles:QuickTrace.txt
==============================================================================

Review of QUICKTRACE............................Mike Sanders

I had already started writing my own debugger when I discovered QUICKTRACE; it was just what I needed and saved me all that work.

It has a good display that does not interfere with the normal Apple text screen. You can even trace code that sets the KSWL and CSWL switches and outputs to the screen. The tracing display takes the bottom four lines, but pressing the "P" key causes the normal bottom four lines to be displayed.

Tracing can be in one of three modes: single-step, trace, and background. Single-step and trace are what you would expect, analogous to the commands in the old Apple monitor ROM. Background turns off the display of executed instructions until a breakpoint occurs or the "ESC" key is pressed. This makes background the fastest mode.

Breakpoints can be set to stop when:

1. Any register or a memory location takes on a specified value.
2. An address or a range of addresses is referenced.
3. A specified opcode occurs.

QUICKTRACE can be BRUN at any point in memory and then called from your code by a JSR, or you can preset the QUICKTRACE program counter and start tracing at any location.

Subroutines can be executed at full 6502 speed (not traced). If you already know what the subroutine does there is no need to trace through it. Normally DOS calls are automatically done this way to prevent timing problems.

Overall I feel that QUICKTRACE is one of the five or so best programs I have ever purchased and no machine code programmer should be without it.

One feature not to be overlooked: QUICKTRACE is not copy protected.

QUICKTRACE was programmed by John Rogers and it is distributed by Anthro-Digital Software (formerly called Aurora Systems). It only costs $50.

=============================================================================
DOCUMENT :AAL-8208:Articles:Search.Perform.txt
=============================================================================

Search and Perform Subroutine..............Bob Sander-Cederlof

When writing an editor or other single-keystroke command system, a very common need is a subroutine which branches according to the value of a character. In Pascal and some other languages there is even a special statement for this programming need: CASE. You might do it like this in Applesoft:

```
1000 GET A$
1010 IF A$ = "A" THEN 2000
1020 IF A$ = "C" THEN 3000
1030 et cetera
```

You will often find the equivalent code in assembly language programs:

```
1000        LDA CHARACTER
1010        CMP #'A
1020        BEQ CHAR.WAS.A
1030        CMP #'C
1040        BEQ CHAR.WAS.C
1050   et cetera
```

Of course, it frequently happens that the number of different values is small, and the code sequence above with several CMP-BEQ pairs is the most efficient. It loses a little of its appeal, though, when you have to do it for more than about ten different values. And what if the branch points are too far away for BEQ relative branches? Then you have to write:

```
1000        LDA CHARACTER
1010        CMP #'A
1020        BNE .1
1030        JMP CHAR.WAS.A
1040 .1     CMP #'C
1050        BNE .2
1060        JMP CHAR.WAS.C
1070 .2   et cetera
```

That takes seven bytes of program for each value of the character.

Personally, I like to put the possible values and the corresponding branch addresses in a table, and search that table whenever necessary. Each table entry takes only three bytes. If the subroutine is used with several tables, and if there are a lot of possible values, then the tabular method saves a lot of memory.

I used the tabular method in my still-in-development word-processor. To speed and simplify the coding of the table entries, I wrote a macro definition JTBL as follows:

```
1020        .MA JTBL
1030        .DA #$]1,]2-1
1040        .EM
```

This defines a macro JTBL with two parameters. The first one will be the hexadecimal value to compare the test-character with, and the second one will be the branch address for that value. For example, if I write the macro call:

```
1400        >JTBL 86,FLIP.CHARS
```

the S-C Macro Assembler will generate:

```
            .DA #$86,FLIP.CHARS-1
```

The "-1" is appended to each branch address in the table, because I use the PHA-PHA-RTS method to perform the branch. Before I go any farther, here is the search and branch subroutine:

```
1220 SEARCH.AND.PERFORM.NEXT
1230        INY                 POINT TO NEXT ENTRY
1240        INY
1250        INY
1260 SEARCH.AND.PERFORM
1270        LDA T.BASE,Y        GET VALUE FROM TABLE
1280        BEQ .1              NOT IN THE TABLE
1290        CMP CURRENT.CHAR
1300        BNE SEARCH.AND.PERFORM.NEXT
1310 .1     LDA T.BASE+2,Y      LOW-BYTE OF BRANCH
1320        PHA
1330        LDA T.BASE+1,Y      HIGH-BYTE OF BRANCH
1340        PHA
1350        LDY #0     (SINCE MOST BRANCHES WANT Y=0)
1360        RTS                 DO THE BRANCH!
```

There are so far four different value-branch tables in my word processor. Here is an abbreviated listing:

```
1380 T.BASE
1390 T.ESC0 >JTBL 81,AUXILIARY.MENU
1400        >JTBL 82,SCAN.BEGIN
1410        >JTBL 83,TOGGLE.CASE.LOCK
 . . . . . .
1540        >JTBL 9B,ESC0.ESC
1550        >JTBL 00,SC.BELL
1560 *--------------------------------
1570 T.ESC2 >JTBL 81,AUXILIARY.MENU
 . . . . . .
1690        >JTBL EB,SCAN.RIGHT
1700        >JTBL ED,SCAN.DOWN
1710        >JTBL 00,ESC2.END
1720 *--------------------------------
1730 T.MAIN >JTBL C4,MAIN.DOS
```

```
1740        >JTBL C5,MAIN.EDIT
 . . . . . .
1800        >JTBL D3,MAIN.SAVE
1810        >JTBL 00,MON.BELL
1820 *--------------------------------
1830 T.AUX  >JTBL C3,COPY.BLOCK
1840        >JTBL C4,DELETE.BLOCK
 . . . . . .
1890        >JTBL D3,SAVE.SEGMENT
1900        >JTBL 00,SC.BELL
```

Notice that each of the four tables ends with a 00 value. The branch address after the 00 value tells where to branch if the current character does not match any values in the table.

When I want to compare the current character with entries in the T.MAIN table, here is how I do it:

```
2000        LDY #T.MAIN-T.BASE
2010        JSR SEARCH.AND.PERFORM
```

The LDY instruction sets Y to the offset of the table from T.BASE, and the search subroutine references the table relative to T.BASE. I use JSR to call the search subroutine. The search subroutine uses PHA-PHA-RTS to effectively JMP to the chosen branch address. And then the value processor ends with RTS to return to the next line after the JSR SEARCH.AND.PERFORM.

Counting all four tables, I have 45 branches, occupying 3*45 = 135 bytes. If I had used the CMP-BEQ method, which occupy four bytes per value, it would have taken 4*45 = 180 bytes. The subroutine is only 23 bytes long, so I saved 22 bytes. But if I needed the longer CMP-BNE-JMP sequences throughout, I would have had 7*45 = 315 bytes! Wow! Long live tables!

Tables have even more advantages. For one, they are a lot easier to modify when you want to add or delete a value. For another, the program is easier to read when there is no rat's nest of branches to try to unravel. For me, it almost makes the assembly listing as easy to read as the reference manual!

Notice that it would be possible to overlap tables using my subroutine. I might need at some times to search for 13 different values, and at others to search for only 7 of those same values, with the same branches. If so, the seven entries in common would be grouped at the end of the 13-entry table. The table has two labels, like this:

```
3000 T.13   >JTBL C1,DO.A
3010        >JTBL C4,DO.D
 . . . . .
3050        >JTBL CF,DO.O
3060 T.7    >JTBL C2,DO.B
3070        >JTBL C5,DO.E
```

```
 . . . . .
3120        >JTBL D7,DO.W
3130        >JTBL 00,DO.NOTHING
```

What about speed? Well, it is pretty fast too. The CMP-BNE-JMP takes five cycles for each value that does not compare equal, and finally seven cycles for the one which compares equal. If the tenth comparison bingos, that is 9*5+7 = 52 cycles. The subroutine takes 171 cycles for the same search. Over three times longer, but still less that 120 microseconds longer. You would have to perform the search over 8000 times in one day to add a whole second of computer time!

=============================================================================
DOCUMENT :AAL-8208:Articles:Shorts.txt
=============================================================================

Correction

Last month I described the BIT instruction incorrectly. The next to the last paragraph on page 2 (in "Run-Anywhere Subroutine Calls") should read:

The BIT instruction copies bit 7 of $FF58 into the N-status bit, and bit 6 into the Overflow status bit. This, in other words (since $FF58 has $60 in it) clears N and sets Overflow.

BIT does not affect Carry Status in any way. BIT also sets or clears the Z-status bit, according to the value of the logical product of the A-register and the addressed byte. If you want Z and/or N to be flags to the calling program, you will have to modify them after the BIT instruction.

Another Customizing Patch for the S-C Macro Assembler

Version 4.0 of the S-C Assembler stopped after any assembly error. Many users requested that I modify it to continue to the end of assembly, and display the error count at the end. So I did.

Now some users are requesting that I change it back. They walk away during assembly, and the error messages scroll off the screen. (But you can put .LIST OFF at the beginning, and then only the error lines will list.)

There is a very simple patch for this. The byte at $1D6F ($DD6F in the language card version) is now $18. Change it $38 and assembly will stop after the first error message.

Subscription Renewals

If your address label shows a number 8209 or smaller in the upper right corner, it is time to renew. That is $15 bulk mail in the USA; $18 First Class in USA, Canada, and Mexico; $28 to other countries.

New Macro Cross Assemblers Available

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various microprocessors. Combining your very versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system.

There are now three cross-assembler modules ready for the S-C Macro Assembler, and more to come. Each cross-assembler disk costs $32.50 to registered owners of the S-C Macro Assembler. You get both regular and language card versions, with documentation of the special features and differences.

The 6809 cross-assembler is designed to work with the Stellation Mill. The MGO command starts the 6809 processor executing your assembled object code.  Likewise, the Z-80 version is designed to work with the Microsoft Softcard.

We have begun working on a Motorola 68000 version....

==============================================================================
DOCUMENT :AAL-8208:Articles:Videx.Patches.txt
==============================================================================

The Macro-Videx Connection........................Don Taylor

It seems that whenever I purchase a new hardware product for my Apple, I spend countless hours honing my most precious software tools to make them compatible with it. I purchased my Videx Videoterm card for use with Pascal, and had no intention of using it with the S-C Assembler. Then one fateful day I made a temporary patch to Version 4.0 -- just to see what it would look like -- and I was immediately hooked....

You won't believe what it's like to assemble with 80 columns of display! You can actually write source files that are legible on the screen, with no wraparound on comments -- even during assembly. What you see on the display is what you would see on a printer, only cleaner.

When I upgraded to the S-C Macro Assembler, I was compelled to produce a configuration file that would modify the new assembler to work with the Videoterm board. The resulting source file is included with this article.

The assembled SCM80 file will reconfigure a copy of the S-C Macro Assembler Version 1.0 that is currently resident in memory (for more about this concept, see "Controlling Software Configuration", AAL April '82).

Once the mods are installed you will be able to use your Videx for everything except: (1) Using the Escape-L sequence to LOAD a disk file whose name appears on the display, and (2) Using the copy key (right arrow). You will still be able to use Escape-L to generate the normal dashed comment line, and you can use the other escape functions to move the cursor and clear portions of the screen.

SCM80 will display control characters (and other selected strings intended to be so) in inverse on your screen, provided you have the standard (inverse) alternate character generator ROM installed in your Videoterm. If you have some other ROM installed, these characters and strings may be printed in Chinese. In this case you may want to modify the new character output routine!

SCM80 will also permit painless switching of case while using the assembler. A control-A keypress will always be recognized as a "shift lock" signal, while a control-Z will be treated as a "shift unlock". This feature makes it easy to write easy-to-read source files.

The assembled SCM80 code is moved into memory immediately following the assembler, and is located at one of two places, depending on which flavor (vanilla or language card) of the assembler you're using. The flavor of the configuration file is made to match that of the assembler through the use of a conditional flag (LCVERSION) and several conditional assembly statements. Another equate variable, SLOTNUM, allows you to specify the slot in which your Videx board resides.

How It All Works

There are two primary steps involved in installing the modified code in the assembler: (1) Moving the new code into the area of memory immediately following the assembler, and (2) Patching the existing assembler code to point to the new routines and then returning or cold-starting the system.

The SCM80 code contains both the new Videoterm support routines and the routines used to install those support routines. It loads in at $4000, stuffs the Videoterm routines just beyond the assembler code, and then performs the return or cold start. Depending on the flavor, a few other small tasks are performed in the process; let's take a closer look.

Lines 1280-1310 contain the two constants used to tailor SCM80 to assembler flavor and Videoterm slot number. The last two lines are the starting addresses where the new code will be relocated, depending on the flavor. The LCVERSION flag is used to determine the base address of the assembler in lines 1340-1380; this base address is used throughout the rest of the listing to determine absolute patch addresses within the assembler.

The Videoterm support routines are contained in lines 3240-3770. Lines 3400-3700 contain replacement routines for two of the routines in the line editor portion of the assembler. The NEW.WARM.ENTRY routine in lines 3240-3260 is intended to keep the Videoterm in the saddle during a RESET or system warm start.

The code in lines 3820-4740 are replacements for some of the standard monitor routines. Several of these routines have no other purpose than to support the escape cursor movements. In the case of the language card flavored RDKEY, an extra subroutine is provided to unprotect the RAM during case-shift sequences (more about that in a minute).

Lines 1770-2040 use the monitor's MOVE routine to slip the support routines into their designated origin at $3200 or $F400. The vanilla version patches the assembler's symbol table address to make room for the move; the language card version unprotects RAM prior to the move.

The patching of the assembler is done in lines 2050-2920. unused code is NOP-ed out here, and jumps are strategically poked in to point to the new routines. A replacement escape jump table created in lines 2950-3090 gets installed in the assembler, so the new escape routines can be accessed in the standard manner. The assembler's cold start routines are patched to point to the resilient NEW.WARM.ENTRY routine (more about that in a monent, too).

Lines 2870-2920 complete the installation and patching process. For the vanilla version, a simple RTS returns control to the calling program. The language card version first write protects RAM and then performs a DOS cold start. Once the assembled code has been installed and the patches made, the installation portion of SCM80 is of no use, so a cold start should be performed to reset the assembler's file pointers, leaving only the SCM80 code that is now supporting your Videoterm.

Assembly and Installation

You'll note the absence of any .TF directive in the listing, meaning you'll have to manually save this file when you're done. This is because although the resulting object code will be located in continuous memory, it has origins (.OR directives) at two locations. The actual length of the file is calculated by a variable called LENGTH. The instructions for assembly are contained in the source file's title block. I call my vanilla patch file SCM80, and the language card version SCM80.LC.

With the assembler code resident in memory, there are several ways of installing the patches. Perhaps the most straightforward is to BRUN the assembled patch file, or BLOAD it and type 4000G as a monitor command. If you're using the vanilla assembler, you'll need to force a cold start of the assembler by typing "NEW" or 1000G as a monitor command; this action will ensure all the internal patches have been installed into DOS as well. The language card version cold starts itself, and requires no intervention.

A cleaner way is to use an EXEC file. The following file will bring up the vanilla version of the assembler:

```
REM LOAD ASM
CALL -151                Enter the monitor
BLOAD S-C.ASM.MACRO      Load the Assembler
BLOAD SCM80              Load the patches
4000G                    Install them, and
1000G                    Start the assembler!
```

To load the language card patches with an EXEC file, refer to Bob's EXEC file on the top of Page 4 of the May '82 AAL, and replace "3D3G" with the following two lines:

```
BLOAD SCM80.LC           Load the LC patches
4000G                    Install them and cold start!
```

The character I/O is being vectored through routines at the end of the assembler; for the language card version, these routines are somewhere in $F4XX. If you decide to issue an "FP" command from that version, you'll find yourself in "Never-Never Land". It's good practice to issue a "PR#n" first (where "n" is the Videoterm's slot number). When you type "INT" to restart the assembler, the special I/O routines will automatically be hooked in.

**A Funny Thing Happened on the Way...**

Bob thought it would be enlightening to touch on some some of the crazy things that went on during the development of these routines. I always marvel at people like Bob, Mike, Bill, and Lee, who have a gift for writing machine language, and can sit down and bang out a line editor in a few hours.

The rest of us aren't quite so fortunate. SCM80 took my three days to write, even though I had done some quick patches on Version 4.0. A couple of good ones popped up during that time, and I'll pass them along.

I was determined to interface the Videoterm using only its terminal functions, avoiding any internal Videoterm ROM routines that would make the interface version-dependent (my card matches neither the descriptions nor the ROM source listings contained in my manual!).

The Videoterm will not move its flashing cursor to a GOTOXY Location unless the cursor is first placed there and then a character is output; under BASIC, you can't just HTAB and VTAB to a position and GET a character -- you have to print a character first (even a null character will do it), in order to move the cursor!

After spending several hours fighting with the Videoterm over who was controlling the input and output cursor locations, I finally decided to designate my own locations for CH and CV (normally at $24 and $25) for use by the editing routines.

The other frustration I incurred was doing the case-switching in the replacement RDKEY routine. I was using the language card version, and had carefully checked my code, but the assembler just wouldn't switch case for me. True confession: it took almost fifteen minutes before it dawned on me that the assembler's case flag (at $D016) was write protected! Hence, the special unprotect subroutine called by the new RDKEY.

One final note concerns the contortions in the replacement COUT and WARM.ENTRY routines (at least I saw these coming!). We need to keep our new RDKEY routine in the DOS input hook to keep things working predictably. The Videoterm, when installed by placing it in the output hook and calling it to output a character, takes over the input hook as well. In addition, we have a replacement COUT routine that is designed to detect and modify control characters for display prior to their output.

In order to avoid arm-wrestling with the Videoterm over who controls the input hook, I used a strange but effective technique. During the installation and patch portion, I install the Videoterm in the designated slot, hook it in, and send a bogus character to make sure it has installed its warm entry I/O locations in DOS ( $AA52-$AA56 for 48K machines). The code immediately following uses an internal assembler routine to calculate the address of the DOS output hook, regardless of memory size. The contents of the DOS output hook are then moved into the new COUT routine, immediately following a JMP, and the same COUT routine is forced into the DOS hook, along with the new RDKEY routine.  Whenever a character is output, it will first be given to COUT; when COUT has done its work, the character is then passed to the Videoterm's warm entry.

During the installation and patch, the warm start vector within the assembler was modified to point to the NEW.WARM.START routine, which re-installs COUT and RDKEY, keeping everything in sync.  A RESET will always restore this condition, no matter what the Videoterm may have in mind!

The S-C Macro Assembler is a wonderful piece of software, and the upgrade is a steal at $27.50.  The only thing that can top it is being able to use it with 80 columns of display!

If you find any errors in my patches, or come up with some new features, contact me at (206) 779-9508.

```
==============================================================================
DOCUMENT :AAL-8208:DOS3.3:Do.Torens.Videx.txt
==============================================================================

0100 ALL COMMENTS REFER TO FOLLOWING
0110 COMMANDS
0111 This file contains the new code
BLOAD AUTOSAVE&VIDEX
0112 This is a new addr table for the input translation
BLOAD JMP.OBJ
0130 Change 'ESC' to CTL P for cursor movement
$136A:90
0132 Lengthen the comment line from an 'esc'L to printer width from
ASM
$1494:4C
0134 Kill the 'auto load' from the ESC mode, it won't work
$1486:10
0140 Addr of AUTO SAVE in command table
$1678:FF 31
0150 move symbol table up to make room for new code
$101D:34
0160 Replace VTAB with GOTOXY in the EDIT command
$1B3B:31 33
$1CB5:31 33
0170 Check for 79 char/line rather than 39
$1B52:4F
0175 New line display length of 80 rather than 40
$1CA8:50
$1CAC:50
0180 New CLREOP function
$1B64:63 33
0205 Since it is not knowN if inverse exists on the target VIDEOTERM
0206 I will display a '?' for control characters in EDIT
$1B4D:A9 BF
0207 Displayed spaces from EDIT will be $A0 rather than $20.
0208 this will be set to $20 in READLINE, and $A0 in EDIT.
$1AF6:CD 33
$134B:D7 33
0210 Patch in AUTO SHIFT on second tab
$14D9:A9 33
$1521:A9 33
0220 New tabs (2nd will trigger SHIFT in col 32
$1010:0E 20 00 00 00
0222 Patch ^O overrride because of $2C trick
$1393:4C E1 33
0224 Right arrow (->) will read from buffer rather than screen
$1397:BD 00 02 EA
0230 Add warm start to setup and clear VIDEX.
$1004:8C 33
0240
0250 *** You may now BSAVE filename,A$1000,L$2400
0260 *
$1003G
```

```
==================================================================
DOCUMENT :AAL-8208:DOS3.3:S.AutoMan.Tgle.txt
==================================================================

 1000  *---------------------------------
 1010  *  AUTO/MANUAL TOGGLE
 1020  *
 1030  *  BY ROBERT F. O'BRIEN
 1040  *     14, CLONSHAUGH LAWN, DUBLIN 5.
 1050  *---------------------------------
 1060        .OR $3228
 1070        .TF AUTO/MANUAL PATCH
 1080  *---------------------------------
 1090  CH     .EQ $24
 1100  SC.SLOW    .EQ $11D2
 1110  SC.REENTER .EQ $135E
 1120  SC.RETURN  .EQ $13C3
 1130  SC.INSTALL .EQ $152A
 1140  SC.LIST    .EQ $183F
 1150  MON.BELL   .EQ $FF3A
 1160  *---------------------------------
 1170  AUTO.MANUAL.COMMAND
 1180         CMP #$81     CTRL-A?
 1190         BEQ AUTO.TOGGLE
 1200         CMP #$8C     CTRL-L?
 1210         BEQ LIST
 1220         CMP #$93     CTRL-S?
 1230         BEQ SLOW.LIST
 1240  *
 1250  BACK   JSR MON.BELL
 1260         JMP SC.REENTER    BACK TO ASSEMBLER
 1270  *---------------------------------
 1280  AUTO.TOGGLE
 1290         LDA CH
 1300         CMP #1       BEGINNING OF LINE?
 1310         BEQ AUTO.CMD
 1320         CMP #6       AFTER LINE NUMBER?
 1330         BEQ MANUAL.CMD
 1340         BNE BACK
 1350  *---------------------------------
 1360  AUTO.CMD
 1370         LDX #0
 1380  .1     LDA AUTO.TEXT,X   GET CHARACTER
 1390         JSR SC.INSTALL  PROCESS CHAR
 1400         CPX #5
 1410         BCC .1
 1420         JMP SC.REENTER
 1430  AUTO.TEXT  .AS -/AUTO /
 1440  *---------------------------------
 1450  MANUAL.CMD
 1460         LDX #0
 1470         STX CH       GO TO START OF LINE
 1480  .1     LDA MANUAL.TEXT,X
```

```
1490        JSR SC.INSTALL
1500        CPX #6
1510        BCC .1
1520        JMP SC.RETURN
1530 MANUAL.TEXT .AS -/MANUAL/
1540 *--------------------------------
1550 LIST   LDA CH
1560        CMP #1       BEGINNING OF LINE?
1570        BNE BACK
1580        JSR SC.LIST
1590        JMP SC.RETURN
1600 *--------------------------------
1610 SLOW.LIST
1620        LDA CH
1630        CMP #1
1640        BNE BACK
1650        JSR SC.SLOW  SET SLOW MODE
1660        JSR SC.LIST
1670        JMP SC.RETURN
```

```
==============================================================
DOCUMENT :AAL-8208:DOS3.3:S.Free.Sectors.txt
==============================================================

 1000  *SAVE S.FREE SECTORS
 1010  *---------------------------------
 1020  *      FREE SECTORS PATCH FOR DOS 3.3
 1030  *---------------------------------
 1040  LOBYTE .EQ $D3
 1050  HIBYTE .EQ $D4
 1060  *---------------------------------
 1070  SECTOR.MAP .EQ $B3F2
 1080  LANG.ID    .EQ $E001    LANGUAGE ID
 1090  PRT.INT    .EQ $E51B    INTEGER BASIC PRINT ROUTINE
 1100  PRT.FP     .EQ $ED24    APPLESOFT PRINT ROUTINE
 1110  PRT.SC     .EQ $DE00    S-C ASSEMBLER PRINT ROUTINE
 1120  *---------------------------------
 1130         .OR $BA69
 1140         .TF FREE.SECTORS.1
 1150  *---------------------------------
 1160  FREE.SECTOR.PATCH
 1170         LDY #$C8
 1180  .1     LDA SECTOR.MAP,Y
 1190         BEQ .4            NO FREE SECTORS IN THIS BYTE
 1200  .2     ASL               SHIFT INTO CARRY
 1210         BCC .2            SECTOR IN USE
 1220         PHA               SECTOR FREE
 1230         INC LOBYTE        COUNT IT
 1240         BNE .3
 1250         INC HIBYTE
 1260  .3     PLA               SECTOR MAP BYTE AGAIN
 1270         BNE .2            IF ANY LEFT
 1280  .4     DEY               NEXT BYTE OF SECTOR MAP
 1290         BNE .1
 1300         LDX LOBYTE        VALUE IN X AND A
 1310         LDA HIBYTE
 1320         LDY LANG.ID       CHECK WHICH LANGUAGE
 1330         BMI SCASM         $94: S-C ASSEMBLER
 1340         BEQ INTEGR        $00: INTEGER BASIC
 1350         JMP PRT.FP        $28: APPLESOFT
 1360  INTEGR JMP PRT.INT
 1370  SCASM  JMP PRT.SC
 1380  *---------------------------------
 1390         .OR $ADB9
 1400         .TF FREE.SECTORS.2
 1410  *---------------------------------
 1420         NOP               FILLER
 1430         LDA #0            ZERO THE COUNT
 1440         STA LOBYTE
 1450         STA HIBYTE
 1460         JSR FREE.SECTOR.PATCH
 1470  *---------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8208:DOS3.3:S.SearchPerform.txt
=============================================================================

 1000  *SAVE S.SEARCH AND PERFORM
 1010  *--------------------------------
 1020         .MA JTBL
 1030         .DA #$]1,]2-1
 1040         .EM
 1050  *--------------------------------
 1060  SEARCH.AND.PERFORM.NEXT
 1070         INY               POINT TO NEXT ENTRY
 1080         INY
 1090         INY
 1100  SEARCH.AND.PERFORM
 1110         LDA T.BASE,Y      GET VALUE FROM TABLE
 1120         BEQ .1            NOT IN THE TABLE
 1130         CMP CURRENT.CHAR
 1140         BNE SEARCH.AND.PERFORM.NEXT
 1150  .1     LDA T.BASE+2,Y    LOW-BYTE OF BRANCH
 1160         PHA
 1170         LDA T.BASE+1,Y    HIGH-BYTE OF BRANCH
 1180         PHA
 1190         LDY #0       (SINCE MOST BRANCHES WANT Y=0)
 1200         RTS               DO THE BRANCH!
 1210  *--------------------------------
 1220  T.BASE
 1230  T.ESC0
 1240         >JTBL 83,TOGGLE.CASE.LOCK
 1250         >JTBL 89,TAB.INSERT
 1260         >JTBL 8D,INSERT.CHAR.INTO.TEXT
 1270         >JTBL 8F,OVERRIDE
 1280         >JTBL 94,TAB.REPLACE
 1290         >JTBL 9B,ESC0.ESC
 1300         >JTBL 00,SC.BELL
 1310  *--------------------------------
 1320  T.ESC2
 1330         >JTBL 83,SET.CASE.TOGGLE
 1340         >JTBL 89,TAB.SKIP
 1350         >JTBL 94,TAB.SKIP
 1360         >JTBL 9B,ESC2.ESC
 1370         >JTBL C9,SCAN.UP.12
 1380         >JTBL CA,SCAN.LEFT.6
 1390         >JTBL CB,SCAN.RIGHT.6
 1400         >JTBL CD,SCAN.DOWN.12
 1410         >JTBL E9,SCAN.UP
 1420         >JTBL EA,SCAN.LEFT
 1430         >JTBL EB,SCAN.RIGHT
 1440         >JTBL ED,SCAN.DOWN
 1450         >JTBL 00,ESC2.END
 1460  *--------------------------------
 1470  T.BOTH
 1480         >JTBL 81,AUXILIARY.MENU
```

```
1490        >JTBL 82,SCAN.BEGIN
1500        >JTBL 84,DELETE.FORWARD.TO.X
1510        >JTBL 85,SCAN.END
1520        >JTBL 86,FLIP.CHARS
1530        >JTBL 88,PUSH.CHAR.ON.KEYSTACK
1540        >JTBL 90,TOGGLE.CLICKER
1550        >JTBL 91,MAIN.MENU
1560        >JTBL 93,SEARCH.AND.REPLACE
1570        >JTBL 95,PULL.CHAR.OFF.KEYSTACK
1580        >JTBL 97,DELETE.WORD
1590        >JTBL 98,DELETE.LINE
1600        >JTBL 9D,TOGGLE.CR.SEE
1610        >JTBL 00,PROCESS.CHAR.1
1620 *--------------------------------
1630 T.MAIN
1640        >JTBL C4,MAIN.DOS
1650        >JTBL C5,MAIN.EDIT
1660        >JTBL CC,MAIN.LOAD
1670        >JTBL CE,MAIN.NEW
1680        >JTBL D0,MAIN.PRINT
1690        >JTBL D1,MAIN.QUIT
1700        >JTBL D3,MAIN.SAVE
1710        >JTBL 00,MON.BELL
1720 *--------------------------------
1730 T.AUX
1740        >JTBL C3,COPY.BLOCK
1750        >JTBL C4,DELETE.BLOCK
1760        >JTBL C6,DISPLAY.FREE
1770        >JTBL C9,INSERT.FILE
1780        >JTBL CD,MOVE.BLOCK
1790        >JTBL D3,SAVE.SEGMENT
1800        >JTBL D4,TAB.SET
1810        >JTBL 00,SC.BELL
1820 *--------------------------------
```

```
=======================================================================
DOCUMENT :AAL-8208:DOS3.3:S.UL.Cursor.txt
=======================================================================

 1000  *SAVE S.UNDERLINE CURSOR
 1010  *--------------------------------
 1020  *      BLINKING UNDERLINE CURSOR
 1030  *      WRITTEN BY BILL LINN
 1040  *--------------------------------
 1050         .OR $300
 1060  *--------------------------------
 1070  MON.CH      .EQ $24
 1080  MON.BASL    .EQ $28
 1090  MON.KSWL    .EQ $38
 1100  MON.RNDL    .EQ $4E
 1110  *--------------------------------
 1120  DOS.REHOOK .EQ $3EA
 1130  *--------------------------------
 1140  MON.BELL2  .EQ $FBE4
 1150  MON.WAIT   .EQ $FCA8
 1160  MON.KEYIN3 .EQ $FD26
 1170  *--------------------------------
 1180  BLANK      .EQ $A0
 1190  UNDERLINE  .EQ $DF
 1200  *--------------------------------
 1210  KEYBOARD   .EQ $C000
 1220  *--------------------------------
 1230  HOOK   LDA #KEYIN   SET INPUT HOOK
 1240         STA MON.KSWL
 1250         LDA /KEYIN
 1260         STA MON.KSWL+1
 1270         JMP DOS.REHOOK
 1280  *--------------------------------
 1290  KEYIN  PHA             SAVE SCREEN CHAR
 1300         STX MON.RNDL     SAVE X-REG
 1310         CMP #UNDERLINE  IF CHAR ON SCREEN IS
 1320         BNE .1            AN UNDERLINE
 1330         LDA #BLANK        THEN ALTERNATE WITH BLANK
 1340  .1     PHA          SAVE CHAR TO ALTERNATE
 1350  *--------------------------------
 1360  *      ALTERNATE UNTIL KEY IS PRESSED
 1370  *--------------------------------
 1380  .2     LDA #UNDERLINE
 1390         LDY MON.CH
 1400         CMP (MON.BASL),Y
 1410         BNE .3
 1420         PLA          GET ALTERNATE CHAR
 1430         PHA          MAINTAIN ON STACK ALSO
 1440  .3     STA (MON.BASL),Y
 1450         LDY #80      80*256 BETWEEN BLINKS
 1460  .4     LDA KEYBOARD      KEY PRESSED?
 1470         BMI .5       YES, CLICK AND RETURN
 1480         DEX
```

```
1490         BNE .4
1500         DEY
1510         BNE .4
1520         BEQ .2       ...ALWAYS
1530  *--------------------------------
1540  *      A KEY HAS BEEN PRESSED
1550  *--------------------------------
1560  .5     PLA          POP STACK ONCE
1570         LDY #10      MAKE A "CLICK"
1580         JSR MON.BELL2
1590         LDY MON.CH
1600         LDX MON.RNDL     RESTORE X-REG
1610         PLA          RESTORE ORIGINAL SCREEN CHAR
1620         JMP MON.KEYIN3
```

```
==============================================================================
DOCUMENT :AAL-8208:DOS3.3:S.Videx.RtArrow.txt
==============================================================================

 1000  *--------------------------------
 1010  * MODIFIED BY MIKE LAUMER
 1020  * TO INCLUDE RIGHT ARROW
 1030  *--------------------------------
 1040  *  Patches for S-C Macro Assembler V1.0
 1050  *        for Videx Videoterm Card
 1060  *
 1070  * Date: 7/10/82
 1080  *
 1090  * Don Taylor
 1100  * infoTool corporation
 1110  * Drawer 809, Poulsbo, WA  98370
 1120  *
 1130  * To assemble this file:
 1140  *
 1150  *      1.  Set SLOTNUM to slot number of videx card
 1160  *
 1170  *      2.  Set LCVERSION flag for
 1180  *             .EQ 1 for Language card version ($D000)
 1190  *             .EQ 0 for Standard version ($1000)
 1200  *
 1210  *      3.  Assemble as usual
 1220  *
 1230  *      4.  Use VAL LENGTH to get length in hex
 1240  *
 1250  *      5.  BSAVE SCM80, A$4000, L$LENGTH
 1260  *
 1270  *--------------------------------
 1280  *
 1290  SLOTNUM             .EQ 3    VIDEX slot
 1300  LCVERSION           .EQ 1    SCM80 version
 1310  PATCH.AREA          .EQ $3200
 1320  LC.PATCH.AREA       .EQ $F400
 1330  *
 1340  *--------------------------------
 1350         .DO LCVERSION
 1360  SCM.BASE   .EQ $D000
 1370         .ELSE
 1380  SCM.BASE   .EQ $1000
 1390         .FIN
 1400  *--------------------------------
 1410  *  Program Constants
 1420  *--------------------------------
 1430  MON.CSW             .EQ $36
 1440  MON.KSW             .EQ $38
 1450  MON.A1L             .EQ $3C
 1460  MON.A2L             .EQ $3E
 1470  MON.A4L             .EQ $42
 1480  SCM.POINTER         .EQ $58
```

```
1490  SCM.CURR.CHAR       .EQ $61
1500  SCM.ED.BEGLIN       .EQ $80
1510  NEW.CH              .EQ $98
1520  NEW.CV              .EQ $99
1530  SCM.WBUF            .EQ $200
1540  DOS.COLD.ENTRY      .EQ $3D3
1550  DOS.IOHOOK          .EQ $3EA
1560  FLAGS               .EQ $7F8  VIDEX Flag Byte
1570  KEYBOARD            .EQ $C000
1580  KEYSTROBE           .EQ $C010
1590  SCM.WARM.ENTRY      .EQ SCM.BASE+$003
1600  SCM.SHIFT.FLAG      .EQ SCM.BASE+$016
1610  SCM.SYM.TABLE       .EQ SCM.BASE+$01D
1620  SCM.TEST.DOS        .EQ SCM.BASE+$31E
1630  SCM.RDL.EOL         .EQ SCM.BASE+$35E
1640  SCM.RDL3            .EQ SCM.BASE+$3C3
1650  SCM.ESC.TABLE       .EQ SCM.BASE+$467
1660  SCM.ESC.L           .EQ SCM.BASE+$483
1670  SCM.RDKEY.NO.CASE   .EQ SCM.BASE+$520
1680  SCM.RDKEY.WITH.CASE .EQ SCM.BASE+$4CA
1690  SCM.SPC             .EQ SCM.BASE+$D92
1700  MON.MOVE            .EQ $FE2C
1710  MON.OUTPORT         .EQ $FE95
1720  MON.COUT            .EQ $FDED
1730  MON.RTS             .EQ $FF58
1740  *--------------------------------
1750         .OR $4000
1760  START1     .EQ *
1770  *--------------------------------
1780  MOVE.CODE
1790         LDA #HERE
1800         STA MON.A1L
1810         LDA /HERE
1820         STA MON.A1L+1
1830         LDA #THERE
1840         STA MON.A2L
1850         LDA /THERE
1860         STA MON.A2L+1
1870  *--------------------------------
1880         .DO LCVERSION
1890         BIT $C083    Unprotect language card RAM
1900         BIT $C083
1910         LDA #LC.PATCH.AREA
1920         STA MON.A4L
1930         LDA /LC.PATCH.AREA
1940         STA MON.A4L+1
1950         .ELSE
1960         LDA #$33     Modify symbol table address
1970         STA SCM.SYM.TABLE
1980         LDA #PATCH.AREA
1990         STA MON.A4L
2000         LDA /PATCH.AREA
2010         STA MON.A4L+1
2020         .FIN
```

```
2030  *-------------------------------
2040         LDY #0
2050         JSR MON.MOVE
2060  INSTALL.PATCHES
2070         LDA #$EA     "NOP-OUT" unused code:
2080         STA SCM.BASE+$343
2090         STA SCM.BASE+$344
2100         STA SCM.BASE+$028
2110         STA SCM.BASE+$029
2120         STA SCM.BASE+$02A
2130         LDX #9
2140  .1     STA SCM.BASE+$298,X
2150         DEX
2160         BPL .1
2170         LDX #14
2180  .2     STA SCM.BASE+$4DE,X
2190         DEX
2200         BPL .2
2210         LDX #48
2220  .3     STA SCM.BASE+$B35,X
2230         DEX
2240         BPL .3
2250         LDA #$20              Install Videx during a
2260         STA SCM.BASE+$295     cold start
2270         LDA #INSTALL.VECTORS
2280         STA SCM.BASE+$296
2290         LDA /INSTALL.VECTORS
2300         STA SCM.BASE+$297
2310         LDA #HOME    Patch clear screen routine
2320         STA SCM.BASE+$2A6
2330         LDA /HOME
2340         STA SCM.BASE+$2A7
2350         LDA #NEW.WARM.ENTRY   Set up warm start so
2360         STA SCM.BASE+$309     VIDEX card stays in..
2370         LDA /NEW.WARM.ENTRY
2380         STA SCM.BASE+$30A
2390         LDA #$10     Patch Escape Routine
2400         STA SCM.BASE+$486
2410         LDY #27
2420  .4     LDA NEW.ESC.TABLE,Y
2430         STA SCM.ESC.TABLE,Y
2440         DEY
2450         BPL .4
2460         LDA #$18     Modify MON.RDKEY jump addr
2470         STA SCM.BASE+$4D9
2480         LDA #$4C     Patch jump to new DISP LINE
2490         STA SCM.BASE+$B32
2500         LDA #NEW.E.DISP.LINE
2510         STA SCM.BASE+$B33
2520         LDA /NEW.E.DISP.LINE
2530         STA SCM.BASE+$B34
2540         LDA #80      Patch E.INPUT Routine
2550         STA SCM.BASE+$CA8
2560         STA SCM.BASE+$CAC
```

```
2570        LDA #NEW.CH
2580        STA SCM.BASE+$CB1
2590        LDA #NEW.CV
2600        STA SCM.BASE+$CB3
2610        LDA #VTAB
2620        STA SCM.BASE+$CB5
2630        LDA /VTAB
2640        STA SCM.BASE+$CB6
2650        LDA #SLOTNUM     Install VIDEX in hook
2660        JSR MON.OUTPORT
2670        JSR DOS.IOHOOK
2680        LDA #$8D         Send CR to get VIDEX warm
2690        JSR MON.COUT     entry point in DOS hook,
2700        LDY #8           then find warm entry address
2710        JSR SCM.TEST.DOS
2720        LDY #1
2730        LDA (SCM.POINTER),Y
2740        STA FAKE.COUT+1   Save warm entry as normal
2750        INY               VIDEX COUT entry
2760        LDA (SCM.POINTER),Y
2770        STA FAKE.COUT+2
2780        LDA #COUT         Hook in new I/O routines
2790        STA MON.CSW
2800        LDA /COUT
2810        STA MON.CSW+1
2820        LDA #RDKEY
2830        STA MON.KSW
2840        LDA /RDKEY
2850        STA MON.KSW+1
2860        JSR DOS.IOHOOK
2870 *---------------------------------
2880        .DO LCVERSION
2890        BIT $C080    Write protect RAM
2900        JMP DOS.COLD.ENTRY
2910        .ELSE
2920        RTS
2930        .FIN
2940 *---------------------------------
2950 *
2960 NEW.ESC.TABLE
2970        .DA HOME-1
2980        .DA ADVNCE-1
2990        .DA BS-1
3000        .DA LF-1
3010        .DA UP-1
3020        .DA CLREOL-1
3030        .DA CLREOP-1
3040        .DA MON.RTS-1
3050        .DA MON.RTS-1
3060        .DA UP-1
3070        .DA BS-1
3080        .DA ADVNCE-1
3090        .DA SCM.ESC.L-1
3100        .DA LF-1
```

```
3110  *-------------------------------
3120  *  New routines to bind into the
3130  *  S-C Macro assembler
3140  *-------------------------------
3150  LENGTH1             .EQ *-START1
3160  HERE                .EQ *
3170         .DO LCVERSION
3180         .OR $F400
3190         .ELSE
3200         .OR $3200
3210         .FIN
3220         .TA HERE
3230  START2              .EQ *
3240  *-------------------------------
3250  NEW.WARM.ENTRY
3260         JSR INSTALL.VECTORS
3270         JMP SCM.WARM.ENTRY
3280  *
3290  INSTALL.VECTORS
3300         LDA #COUT
3310         STA MON.CSW
3320         LDA /COUT
3330         STA MON.CSW+1
3340         LDA #RDKEY
3350         STA MON.KSW
3360         LDA /RDKEY
3370         STA MON.KSW+1
3380         JSR DOS.IOHOOK
3390         RTS
3400  *
3410  NEW.E.DISP.LINE
3420         LDA SCM.ED.BEGLIN
3430         STA NEW.CV
3440         LDA #0
3450         STA NEW.CH
3460         JSR VTAB
3470         JSR SCM.SPC
3480         INC NEW.CH
3490         INC NEW.CH
3500         LDX #0
3510  .1     LDA SCM.WBUF,X
3520         BEQ .5
3530         ORA #$80
3540         CMP #$A0     Control char?
3550         BCS .2       No..
3560         AND #$7F     Flag it as inverse
3570  .2     LDY NEW.CH
3580         CPY #80      End of screen line?
3590         BCC .4       No..
3600         LDY #0       Set CH to beg of line
3610         STY NEW.CH
3620         LDY NEW.CV
3630         CPY #23
3640         BCS .3
```

```
3650        INC NEW.CV   No..
3660        BNE .4       ..Always
3670 .3     DEC SCM.ED.BEGLIN
3680 .4     JSR MON.COUT
3690        INC NEW.CH
3700        INX
3710        BNE .1       ..Always
3720 .5     JMP CLREOP
3730 *
3740 NEW.E.ZAP
3750        LDA #0       EOL mark
3760        STA SCM.WBUF,X
3770        JSR CLREOL
3780        RTS
3790 *--------------------------------
3800 *  Monitor Replacement Routines
3810 *--------------------------------
3820 *
3830 HOME   LDA #$8C     Send Form Feed Char
3840        JMP MON.COUT
3850 *
3860 CLREOL LDA #$9D     Send CLEAREOL char
3870        JMP MON.COUT
3880 *
3890 CLREOP LDA #$8B     Send CLEAREOS char
3900        JMP MON.COUT
3910 *
3920 ADVNCE LDA #$9C     Non-destructive space
3930        JMP MON.COUT
3940 *
3950 BS     LDA #$88     Backspace
3960        JMP MON.COUT
3970 *
3980 LF     LDA #$8A     Linefeed
3990        JMP MON.COUT
4000 *
4010 UP     LDA #$9F     Reverse Linefeed
4020        JMP MON.COUT
4030 *--------------------------------
4040 V.BASEL .EQ $478+SLOTNUM
4050 V.BASEH .EQ $4F8+SLOTNUM
4060 V.CHORZ .EQ $578+SLOTNUM
4070 V.XSAV1 .EQ $402
4080 V.OLDCHAR .EQ $678
4090 *
4100 V.DEV0  .EQ SLOTNUM*16+$C080
4110 V.DISP0 .EQ $CC00
4120 V.DISP1 .EQ $CD00
4130 *--------------------------------
4140 *
4150 RDKEY  LDA KEYBOARD
4160        BPL RDKEY
4170        STA KEYSTROBE
4180        ORA #$80
```

```
4190        CMP #$81     Shift lock?
4200        BNE .1
4210        .DO LCVERSION
4220        JSR UNPROTECT.LC.RAM
4230        .FIN
4240        LSR SCM.SHIFT.FLAG
4250        BPL .2       Return with errant key
4260 .1     CMP #$9A     Shift unlock?
4270        BNE CTRLU    No, return with key
4280        .DO LCVERSION
4290        JSR UNPROTECT.LC.RAM
4300        .FIN
4310        SEC
4320        ROR SCM.SHIFT.FLAG
4330 .2     LDA #$96     Return with errant key
4340        .DO LCVERSION
4350        BIT $C080    Reprotect LC RAM
4360        RTS
4370 *
4380 UNPROTECT.LC.RAM
4390        BIT $C083    Enable Bank 2
4400        BIT $C083
4410        .FIN
4420        RTS
4430 *
4440 CTRLU  CMP #$95     CTRL-U COPY KEY
4450        BNE .3
4460        STX $400
4470        STY $401
4480        LDA V.CHORZ
4490        JSR PSNCALC
4500        BCS .1
4510        LDA V.DISP0,X
4520        BCC .2
4530 .1     LDA V.DISP1,X
4540 .2     ORA #$80
4550        STA V.OLDCHAR
4560        LDX $400
4570        LDY $401
4580 .3     RTS
4590 *
4600 PSNCALC CLC
4610        ADC V.BASEL
4620        STA V.XSAV1
4630        LDA #0
4640        ADC V.BASEH
4650        LSR
4660        PHP
4670        AND #3
4680        ASL
4690        ASL
4700        TAY
4710        LDA V.DEV0,Y
4720        PLP
```

```
4730        LDX V.XSAV1
4740        RTS
4750 *--------------------------------
4760 *
4770 VTAB   LDA #$9E     Send GOTOXY char
4780        JSR MON.COUT
4790        CLC          Create ASCII x-posn
4800        LDA NEW.CH
4810        ADC #160
4820        JSR MON.COUT
4830        CLC          Create ASCII y-posn
4840        LDA NEW.CV
4850        ADC #160
4860        JMP MON.COUT
4870 *
4880 COUT
4890        PHA            Test for inverse
4900        PLA
4910        BMI FAKE.COUT  Not inverse: Take as is
4920        ORA #$80       Restore to "Normal" Apple ASCII
4930        CMP #$A0       Control char?
4940        BCS .1         No..
4950        ORA #$40       Yes: Make it printable
4960 .1     TAY            Save char
4970        LDA FLAGS+SLOTNUM
4980        PHA            Save flag byte
4990        ORA #1         Switch in alt char set
5000        STA FLAGS+SLOTNUM
5010        TYA            Get char back
5020        JSR FAKE.COUT
5030        PLA            Restore flag byte
5040        STA FLAGS+SLOTNUM
5050        RTS
5060 FAKE.COUT
5070        JMP $FFFF      Address will be fixed later..
5080 *--------------------------------
5090 LENGTH2             .EQ *-START2
5100 THERE               .EQ HERE+LENGTH2-1
5110 LENGTH              .EQ LENGTH1+LENGTH2
5120 *--------------------------------
5130        .EN
```

```
=====================================================================
DOCUMENT :AAL-8208:DOS3.3:S.Videx.Taylor.txt
=====================================================================

 1000         .LIST OFF
 1010  *--------------------------------
 1020  *                 SCM80
 1030  *  Patches for S-C Macro Assembler V1.0
 1040  *        for Videx Videoterm Card
 1050  *
 1060  * Date: 7/10/82
 1070  *
 1080  * Don Taylor
 1090  * infoTool corporation
 1100  * Drawer 809, Poulsbo, WA  98370
 1110  *
 1120  * To assemble this file:
 1130  *
 1140  *      1.  Set SLOTNUM to slot number of videx card
 1150  *
 1160  *      2.  Set LCVERSION flag for
 1170  *            .EQ 1 for Language card version ($D000)
 1180  *            .EQ 0 for Standard version ($1000)
 1190  *
 1200  *      3.  Assemble as usual
 1210  *
 1220  *      4.  Use VAL LENGTH to get length in hex
 1230  *
 1240  *      5.  BSAVE SCM80, A$4000, L$LENGTH
 1250  *
 1260  *--------------------------------
 1270  *
 1280  SLOTNUM             .EQ 3    VIDEX slot
 1290  LCVERSION           .EQ 1    SCM80 version
 1300  PATCH.AREA          .EQ $3200
 1310  LC.PATCH.AREA       .EQ $F400
 1320  *
 1330  *--------------------------------
 1340         .DO LCVERSION
 1350  SCM.BASE   .EQ $D000
 1360         .ELSE
 1370  SCM.BASE   .EQ $1000
 1380         .FIN
 1390  *--------------------------------
 1400  *  Program Constants
 1410  *--------------------------------
 1420  MON.CSW             .EQ $36
 1430  MON.KSW             .EQ $38
 1440  MON.A1L             .EQ $3C
 1450  MON.A2L             .EQ $3E
 1460  MON.A4L             .EQ $42
 1470  SCM.POINTER         .EQ $58
 1480  SCM.CURR.CHAR       .EQ $61
```

```
1490  SCM.ED.BEGLIN       .EQ $80
1500  NEW.CH              .EQ $98
1510  NEW.CV              .EQ $99
1520  SCM.WBUF            .EQ $200
1530  DOS.COLD.ENTRY      .EQ $3D3
1540  DOS.IOHOOK          .EQ $3EA
1550  FLAGS               .EQ $7F8  VIDEX Flag Byte
1560  KEYBOARD            .EQ $C000
1570  KEYSTROBE           .EQ $C010
1580  SCM.WARM.ENTRY      .EQ SCM.BASE+$003
1590  SCM.SHIFT.FLAG      .EQ SCM.BASE+$016
1600  SCM.SYM.TABLE       .EQ SCM.BASE+$01D
1610  SCM.TEST.DOS        .EQ SCM.BASE+$31E
1620  SCM.RDL.EOL         .EQ SCM.BASE+$35E
1630  SCM.RDL3            .EQ SCM.BASE+$3C3
1640  SCM.ESC.TABLE       .EQ SCM.BASE+$467
1650  SCM.ESC.L           .EQ SCM.BASE+$483
1660  SCM.RDKEY.NO.CASE   .EQ SCM.BASE+$520
1670  SCM.RDKEY.WITH.CASE .EQ SCM.BASE+$4CA
1680  SCM.SPC             .EQ SCM.BASE+$D92
1690  MON.MOVE            .EQ $FE2C
1700  MON.OUTPORT         .EQ $FE95
1710  MON.COUT            .EQ $FDED
1720  MON.RTS             .EQ $FF58
1730  *--------------------------------
1740         .OR $4000
1750  START1     .EQ *
1760  *--------------------------------
1770  MOVE.CODE
1780         LDA #HERE
1790         STA MON.A1L
1800         LDA /HERE
1810         STA MON.A1L+1
1820         LDA #THERE
1830         STA MON.A2L
1840         LDA /THERE
1850         STA MON.A2L+1
1860  *--------------------------------
1870         .DO LCVERSION
1880         BIT $C083    Unprotect language card RAM
1890         BIT $C083
1900         LDA #LC.PATCH.AREA
1910         STA MON.A4L
1920         LDA /LC.PATCH.AREA
1930         STA MON.A4L+1
1940         .ELSE
1950         LDA #$33     Modify symbol table address
1960         STA SCM.SYM.TABLE
1970         LDA #PATCH.AREA
1980         STA MON.A4L
1990         LDA /PATCH.AREA
2000         STA MON.A4L+1
2010         .FIN
2020  *--------------------------------
```

```
2030          LDY #0
2040          JSR MON.MOVE
2050   INSTALL.PATCHES
2060          LDA #$EA      "NOP-OUT" unused code:
2070          STA SCM.BASE+$343
2080          STA SCM.BASE+$344
2090          STA SCM.BASE+$028
2100          STA SCM.BASE+$029
2110          STA SCM.BASE+$02A
2120          LDX #9
2130   .1     STA SCM.BASE+$298,X
2140          DEX
2150          BPL .1
2160          LDX #14
2170   .2     STA SCM.BASE+$4DE,X
2180          DEX
2190          BPL .2
2200          LDX #48
2210   .3     STA SCM.BASE+$B35,X
2220          DEX
2230          BPL .3
2240          LDA #$20                Install Videx during a
2250          STA SCM.BASE+$295       cold start
2260          LDA #INSTALL.VECTORS
2270          STA SCM.BASE+$296
2280          LDA /INSTALL.VECTORS
2290          STA SCM.BASE+$297
2300          LDA #HOME     Patch clear screen routine
2310          STA SCM.BASE+$2A6
2320          LDA /HOME
2330          STA SCM.BASE+$2A7
2340          LDA #NEW.WARM.ENTRY     Set up warm start so
2350          STA SCM.BASE+$309       VIDEX card stays in..
2360          LDA /NEW.WARM.ENTRY
2370          STA SCM.BASE+$30A
2380          LDA #$10      Patch Escape Routine
2390          STA SCM.BASE+$486
2400          LDY #27
2410   .4     LDA NEW.ESC.TABLE,Y
2420          STA SCM.ESC.TABLE,Y
2430          DEY
2440          BPL .4
2450          LDA #$18      Modify MON.RDKEY jump addr
2460          STA SCM.BASE+$4D9
2470          LDA #$4C      Patch jump to new DISP LINE
2480          STA SCM.BASE+$B32
2490          LDA #NEW.E.DISP.LINE
2500          STA SCM.BASE+$B33
2510          LDA /NEW.E.DISP.LINE
2520          STA SCM.BASE+$B34
2530          LDA #80       Patch E.INPUT Routine
2540          STA SCM.BASE+$CA8
2550          STA SCM.BASE+$CAC
2560          LDA #NEW.CH
```

```
2570        STA SCM.BASE+$CB1
2580        LDA #NEW.CV
2590        STA SCM.BASE+$CB3
2600        LDA #VTAB
2610        STA SCM.BASE+$CB5
2620        LDA /VTAB
2630        STA SCM.BASE+$CB6
2640        LDA #SLOTNUM     Install VIDEX in hook
2650        JSR MON.OUTPORT
2660        JSR DOS.IOHOOK
2670        LDA #$8D         Send CR to get VIDEX warm
2680        JSR MON.COUT     entry point in DOS hook,
2690        LDY #8           then find warm entry address
2700        JSR SCM.TEST.DOS
2710        LDY #1
2720        LDA (SCM.POINTER),Y
2730        STA FAKE.COUT+1   Save warm entry as normal
2740        INY               VIDEX COUT entry
2750        LDA (SCM.POINTER),Y
2760        STA FAKE.COUT+2
2770        LDA #COUT         Hook in new I/O routines
2780        STA MON.CSW
2790        LDA /COUT
2800        STA MON.CSW+1
2810        LDA #RDKEY
2820        STA MON.KSW
2830        LDA /RDKEY
2840        STA MON.KSW+1
2850        JSR DOS.IOHOOK
2860 *--------------------------------
2870        .DO LCVERSION
2880        BIT $C080    Write protect RAM
2890        JMP DOS.COLD.ENTRY
2900        .ELSE
2910        RTS
2920        .FIN
2930 *--------------------------------
2940 *
2950 NEW.ESC.TABLE
2960        .DA HOME-1
2970        .DA ADVNCE-1
2980        .DA BS-1
2990        .DA LF-1
3000        .DA UP-1
3010        .DA CLREOL-1
3020        .DA CLREOP-1
3030        .DA MON.RTS-1
3040        .DA MON.RTS-1
3050        .DA UP-1
3060        .DA BS-1
3070        .DA ADVNCE-1
3080        .DA SCM.ESC.L-1
3090        .DA LF-1
3100 *--------------------------------
```

```
3110 *   New routines to bind into the
3120 *   S-C Macro assembler
3130 *--------------------------------
3140 LENGTH1             .EQ *-START1
3150 HERE                .EQ *
3160        .DO LCVERSION
3170        .OR $F400
3180        .ELSE
3190        .OR $3200
3200        .FIN
3210        .TA HERE
3220 START2              .EQ *
3230 *--------------------------------
3240 NEW.WARM.ENTRY
3250        JSR INSTALL.VECTORS
3260        JMP SCM.WARM.ENTRY
3270 *
3280 INSTALL.VECTORS
3290        LDA #COUT
3300        STA MON.CSW
3310        LDA /COUT
3320        STA MON.CSW+1
3330        LDA #RDKEY
3340        STA MON.KSW
3350        LDA /RDKEY
3360        STA MON.KSW+1
3370        JSR DOS.IOHOOK
3380        RTS
3390 *
3400 NEW.E.DISP.LINE
3410        LDA SCM.ED.BEGLIN
3420        STA NEW.CV
3430        LDA #0
3440        STA NEW.CH
3450        JSR VTAB
3460        JSR SCM.SPC
3470        INC NEW.CH
3480        INC NEW.CH
3490        LDX #0
3500 .1     LDA SCM.WBUF,X
3510        BEQ .5
3520        ORA #$80
3530        CMP #$A0     Control char?
3540        BCS .2       No..
3550        AND #$7F     Flag it as inverse
3560 .2     LDY NEW.CH
3570        CPY #80      End of screen line?
3580        BCC .4       No..
3590        LDY #0       Set CH to beg of line
3600        STY NEW.CH
3610        LDY NEW.CV
3620        CPY #23
3630        BCS .3
3640        INC NEW.CV   No..
```

```
3650          BNE .4        ..Always
3660   .3     DEC SCM.ED.BEGLIN
3670   .4     JSR MON.COUT
3680          INC NEW.CH
3690          INX
3700          BNE .1        ..Always
3710   .5     JMP CLREOP
3720   *
3730   NEW.E.ZAP
3740          LDA #0        EOL mark
3750          STA SCM.WBUF,X
3760          JSR CLREOL
3770          RTS
3780   *--------------------------------
3790   *  Monitor Replacement Routines
3800   *--------------------------------
3810   *
3820   HOME   LDA #$8C      Send Form Feed Char
3830          JMP MON.COUT
3840   *
3850   CLREOL LDA #$9D      Send CLEAREOL char
3860          JMP MON.COUT
3870   *
3880   CLREOP LDA #$8B      Send CLEAREOS char
3890          JMP MON.COUT
3900   *
3910   ADVNCE LDA #$9C      Non-destructive space
3920          JMP MON.COUT
3930   *
3940   BS     LDA #$88      Backspace
3950          JMP MON.COUT
3960   *
3970   LF     LDA #$8A      Linefeed
3980          JMP MON.COUT
3990   *
4000   UP     LDA #$9F      Reverse Linefeed
4010          JMP MON.COUT
4020   *--------------------------------
4030          .DO LCVERSION
4040   RDKEY  LDA KEYBOARD
4050          BPL RDKEY
4060          STA KEYSTROBE
4070          ORA #$80
4080          CMP #$81      Shift lock?
4090          BNE .1
4100          JSR UNPROTECT.LC.RAM
4110          LSR SCM.SHIFT.FLAG
4120          BPL .2        Return with errant key
4130   .1     CMP #$9A      Shift unlock?
4140          BNE .3        No, return with key
4150          JSR UNPROTECT.LC.RAM
4160          SEC
4170          ROR SCM.SHIFT.FLAG
4180   .2     BIT $C080     Reprotect LC RAM
```

```
4190        LDA #$96     Return with errant key
4200 .3     RTS
4210 *
4220 UNPROTECT.LC.RAM
4230        BIT $C083    Enable Bank 2
4240        BIT $C083
4250        RTS
4260        .ELSE
4270 RDKEY  LDA KEYBOARD
4280        BPL RDKEY
4290        STA KEYSTROBE
4300        ORA #$80
4310        CMP #$81     Shift lock?
4320        BNE .1
4330        LSR SCM.SHIFT.FLAG
4340        BPL .2       Return with errant key
4350 .1     CMP #$9A     Shift unlock?
4360        BNE .3       No, return with key
4370        SEC
4380        ROR SCM.SHIFT.FLAG
4390 .2     LDA #$96     Return with errant key
4400 .3     RTS
4410        .FIN
4420 *--------------------------------
4430 *
4440 VTAB   LDA #$9E     Send GOTOXY char
4450        JSR MON.COUT
4460        CLC          Create ASCII x-posn
4470        LDA NEW.CH
4480        ADC #160
4490        JSR MON.COUT
4500        CLC          Create ASCII y-posn
4510        LDA NEW.CV
4520        ADC #160
4530        JMP MON.COUT
4540 *
4550 COUT
4560        PHA            Test for inverse
4570        PLA
4580        BMI FAKE.COUT  Not inverse: Take as is
4590        ORA #$80       Restore to "Normal" Apple ASCII
4600        CMP #$A0       Control char?
4610        BCS .1         No..
4620        ORA #$40       Yes: Make it printable
4630 .1     TAY            Save char
4640        LDA FLAGS+SLOTNUM
4650        PHA            Save flag byte
4660        ORA #1         Switch in alt char set
4670        STA FLAGS+SLOTNUM
4680        TYA            Get char back
4690        JSR FAKE.COUT
4700        PLA            Restore flag byte
4710        STA FLAGS+SLOTNUM
4720        RTS
```

```
4730  FAKE.COUT
4740         JMP $FFFF      Address will be fixed later..
4750  *--------------------------------
4760  LENGTH2             .EQ *-START2
4770  THERE               .EQ HERE+LENGTH2-1
4780  LENGTH              .EQ LENGTH1+LENGTH2
4790  *--------------------------------
4800         .EN
```

```
==============================================================
DOCUMENT :AAL-8208:DOS3.3:S.Videx.Toren.txt
==============================================================

 1000        .TI 60,VIDEX mods TO S-C Macro Assembler by Rip Toren
 1010  *=========================================================
 1020  * Modifications to the S-C Macro Assembler to interface   *
 1030  * with the VIDEX 80 col. display board.                  *
 1040  *                                                        *
 1050  *      BY                                                *
 1060  * Richard P. Toren                                       *
 1070  *  July 1982                                             *
 1080  *                                                        *
 1090  *=========================================================
 1095
 1096
 1100  *  .FN "ASAVE&VIDEX
 1101
 1102
 1110  *=========================================================
 1120  * AUTO-SAVE
 1125  *
 1130  * from AAL Apr 1982
 1140  * The mod to the version number was not implemented.
 1150  *
 1160  * I have included this since this is one of the best mods
 1165  * for the assembler
 1166  *
 1170  * My file name is in a comment:
 1180  *
 1190  * .FN "filename
 1200  *
 1210  * LENGTH OF $141
 1220  * A$3200,-$3341
 1225  *=========================================================
 1230  MON.COUT   .EQ $FDED
 1240  MON.CROUT  .EQ $FD8E
 1250  MON.BELL1  .EQ $FBDD
 1260  IN.BUF     .EQ $200
 1270  SCR.END    .EQ $4C,4D
 1280  SCR.START  .EQ $CA,CB
 1290  NEXT       .EQ $1D
 1300  SEARCH     .EQ $1E,1F
 1310
 1320        .OR $3200
 1330        .TF AUTOSAVE&VIDEX
 1340
 1350  AUTO.SAVE
 1360        LDA SCR.START
 1370        STA SEARCH
 1380        LDA SCR.START+1
 1390        STA SEARCH+1
 1400        CLD
```

```
1410  ADDRESS.END.CMP
1420         LDA SEARCH
1430         CMP SCR.END
1440         BNE .1
1450         LDA SEARCH+1
1460         CMP SCR.END+1
1470         BEQ ERROR1
1480
1490  .1     LDY #0
1500         LDA (SEARCH),Y
1510         STA NEXT
1520         LDY #3
1530         LDA (SEARCH),Y
1540         CMP #'*
1550         BNE NEW.LINE  MINE IN COM
1560  .5     CMP #$C0
1570         BNE .2
1580  .4     INY
1590         INY
1600         CLV
1610         BVC .3
1620  .2     CMP #$80
1630         BCS OPCHK
1640  .3     INY
1650         LDA (SEARCH),Y
1660         BEQ NEW.LINE
1670         BNE .5
1700
1710  NEW.LINE
1720         CLC
1730         LDA SEARCH
1740         ADC NEXT
1750         STA SEARCH
1760         BCC ADDRESS.END.CMP
1770         INC SEARCH+1
1780         BNE ADDRESS.END.CMP
1810
1820  ERROR1
1830         LDY #0
1840  PRTERR LDA NO.TTL,Y
1850         BMI ERREND
1860         ORA #$80
1870         JSR MON.COUT
1880         INY
1890         BNE PRTERR
1900  ERREND JSR MON.COUT
1910         JSR MON.BELL1
1920         JSR MON.BELL1
1930         JSR MON.CROUT
1940         RTS
1950  ERROR2
1960         LDY #18
1970         BNE PRTERR
1980
```

```
2000
2010  OPCHK  LDX #0
2020  .1     INY
2030         LDA (SEARCH),Y
2040         BEQ NEW.LINE
2050         CMP OPS,X
2060         BNE NEW.LINE
2070         INX
2080         CPX #3
2090         BNE .1
2120
2130  TITLE  INY
2140         LDA (SEARCH),Y
2150         BEQ ERROR1
2160         CMP#'"        "FILE NAME"
2170         BNE TITLE
2180  .1     INY
2190         LDA (SEARCH),Y
2200         BEQ ERROR1
2210         CMP #$C0
2220         BEQ COMP.CODE1
2230         CMP #$80
2240         BCS .1
2250         CMP #'A
2260         BCC ERROR2
2270         CMP $58
2280         BCS ERROR2
2320
2330         PHA
2340         LDX #0
2350  .2     LDA SAVE,X
2360         JSR MON.COUT
2370         INX
2380         CPX #5
2390         BNE .2
2400         PLA
2410  NEXT.CHAR1
2420         ORA #$80
2430         JSR MON.COUT
2440         INX
2450  NEXT.CHAR2
2460         INY
2470         LDA (SEARCH),Y
2480         BEQ DOS.OP
2490         CMP #',
2500         BNE .1
2510         LDA #'/
2520         BNE NEXT.CHAR1
2530  .1     CMP #$C0
2540         BEQ COMP.CODE2
2550         CMP #$80
2560         BCC NEXT.CHAR1
2570         INY
2580         LDA (SEARCH),Y
```

```
2590        BEQ DOS.OP
2600        DEY
2610        LDA #$20
2620        BNE NEXT.CHAR1
2650
2660 COMP.CODE1
2670        INY
2680        LDA (SEARCH),Y
2690        STA NEXT
2700        INY
2710        LDA (SEARCH),Y
2720        CMP #'A
2730        BCC ERROR2
2740        CMP #$5B
2750        BCS ERROR2
2760        PHA
2770        LDX #0
2780 .1     LDA SAVE,X
2790        JSR MON.COUT
2800        INX
2810        CPX #5
2820        BNE .1
2830        PLA
2840        BNE STORE
2850 COMP.CODE2
2860        INY
2870        LDA (SEARCH),Y
2880        STA NEXT
2890        INY
2900        LDA (SEARCH),Y
2910        CMP #',
2920        BNE STORE
2930        LDA #'/
2940 STORE
2950        ORA #$80
2960        JSR MON.COUT
2970        INX
2980        DEC NEXT
2990        BNE STORE
3000        BEQ NEXT.CHAR2
3005
3010 *
3020 * Skip the version update for the moment
3030 *
3040 DOS.OP
3050        JSR MON.CROUT
3060 END    RTS
3090
3100 OPS    .AS /.FN/
3110 SAVE   .HS 84       ctl D
3120        .AS -/SAVE/
3130 NO.TTL .AT /!! NO TITLE ERROR /
3140        .AT /!! ILLEGAL TITLE /
3150
```

```


*=============================================================
* The following are the commands that are issued to the
* VIDEX to perform the given function.
*=============================================================

COUT   .EQ $FDED

* This routine will position cursor at CH,CV
GOTOXY JSR SAVEREG
       LDA $25      save CV
       PHA
       LDA $24      save CH
       PHA
       LDA #30
       JSR COUT     goto leadin
       PLA          get CH
       CLC
       ADC #$20
       JSR COUT
       PLA          get CV
       ADC #$20
       JSR COUT
       JMP RESTREG
ADVANCE
       LDA #28
       JMP SAFEOUT
BS
       LDA #8
       JMP SAFEOUT
UP
       LDA #31
       JMP SAFEOUT
DOWN
       LDA #10
       JMP SAFEOUT
CLREOP
       LDA #11
       JMP SAFEOUT
CLEAR
       LDA #12
       JMP SAFEOUT
CLREOL
       LDA #29
       JMP SAFEOUT
*
SAFEOUT
       JSR SAVEREG
       JSR COUT
       JSR RESTREG
       RTS
*
SAVEREG
```

```
 3730        STX SX          don't distrurb the regs
 3740        STY SY
 3750        RTS
 3760  RESTREG
 3770        LDX SX          get them back
 3780        LDY SY
 3790        RTS
 3800  SX     .DA #0
 3810  SY     .DA #0
 3820
 3830  *************************************************************
 3840  * The following is the warm start patch that will initialize
the
 3850  * the VIDEX board, and set the cursor to an underline. This
routine
 3860  * is also used by "NEW".
 3870  *************************************************************
 3880
 3890
 3900  *   !!!!!!!!!!!!
 3910  *  SET THE SLOT VALUE FOR YOUR VIDEX BOARD
 3920
 3930  SLOT   .EQ $3             slot#
 3940  SLOT16 .EQ SLOT*16
 3950  V.DEV0 .EQ $C080          addresses the CRTC internal regs
 3960  V.DEV1 .EQ $C081          content of specified reg
 3970  *===========================================================
 3980  VID.ON LDA #SLOT
 3990         JSR $FE95          simulate PR#3
 4000  V.CRSON
 4010         LDA #$0A           CRTC reg 10
 4020         STA V.DEV0+SLOT16
 4030         LDA #$88           non-flashing underline
 4040         STA V.DEV1+SLOT16
 4050  * self modification for automatic shift to lower case.
 4060         LDA $1011          second tab
 4070         SBC #1             offset for initial blank
 4080         STA VT1+1          target of self-modification
 4090         JSR CLEAR          clear screen
 4100         JMP $101C            SCM warm-start
 4110  *
 4120  *===========================================================
 4130  * This routine will shift you from upper to lower case at
 4140  * the second TAB stop. I use this for the comments.
 4145  *===========================================================
 4150  RDKEY  .EQ $FD0C
 4160  AUTOSHIFT
 4170  VT1    CPX #0             will be modified with 2nd tab value
 4180         BNE .1             no action
 4190         LDA $205
 4200         CMP #$AA           "*" this is a comment
 4210         BEQ .1             no conversion on comments
 4220         LDA #$40           force lower case
 4230         STA $7F8+SLOT
```

```
4240  .1     JSR RDKEY         now get the input key
4250         CMP #$8D          on carriage return, flip back
4260         BEQ VT2
4270         CMP #$98          ^X  same here
4280         BEQ VT2
4290         RTS               stay in current case
4300  VT2    PHA
4310         LDA #$00          force upper case
4320         STA $7F8+SLOT
4330         PLA
4340         RTS
4350
4360  *=========================================================
4370  * The following are needed since my EDIT wants $A0 for space,
4380  * while everyone else is looking for a $20.
4390  *=========================================================
4400  ZAP.A0 PHA
4410         LDA #$A0
4420         STA $127A
4430         PLA
4440         JMP $185D    GET.KEY.STRING
4450  ZAP.20 PHA
4460         LDA #$20
4470         STA $127A
4480         PLA
4490         JMP $1D94    CHO
4500
4510  *=========================================================
4520  * RDL.OVERRIDE is needed because of the $2C trick and I
4530  * must read '->' characters from buffer.
4535  *=========================================================
4540  RDL.OVERRIDE
4550         JSR $14CA         read.key.with.case
4560         ORA #$80          assure sign bit on
4570         JMP $139B         rdl.add.char
4580
4590  ZZZEND .EQ *
4600  ZZZLEN .EQ ZZZEND-AUTO.SAVE
4610  *=========================================================
4620  * Replace the jump addres in the ESC handler routine.
4630  *=========================================================
4635
4640         .OR $1467
4650         .TF JMP.OBJ
4660         .DA CLEAR-1       videx routinee
4670         .DA ADVANCE-1     videx routine
4680         .DA BS-1          videx routine
4690         .DA DOWN-1        videx routine
4700         .DA UP-1          videx routine
4710         .DA CLREOL-1      videx routine
4720         .DA CLREOP-1      videx routine
4730         .DA $FC2A         no change
4740         .DA $FC2A         no change
4750         .DA UP-1          videx routine
```

```
4760        .DA BS-1          videx routine
4770        .DA ADVANCE-1     videx routine
4780        .DA $1482         no change
4790        .DA DOWN-1        videx routine
```

==============================================================================
DOCUMENT :AAL-8208:DOS3.3:Toren.Dox.txt
==============================================================================

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8209:Articles:Amper.Vector.txt
==============================================================================

Relocatable Ampersand-Vector........................Steve Mann

In recent issues of AAL there have been a variety of routines to produce relocatable code. The BSR, BRA and LEAX opcodes in the June issue and the run-anywhere subroutine calls in the July issue are two examples.

However, in making some of my code relocatable, I encountered a new problem with routines that interface with Applesoft programs through the & command. The problem is that the routine doesn't know what address to place in the & jump vector because that address may change with each run.

A rather inelegant solution is to derive the address from Applesoft's pointers, then POKE it into the & vector before calling it. What I wanted was a method to determine the correct address from within the code itself, in much the same way that a non-relocatable program sets up the vector:

```
1000        LDA #$4C
1010        STA AMPER.VECTOR
1020        LDA #START
1030        STA AMPER.VECTOR+1
1040        LDA /START
1050        STA AMPER.VECTOR+2
1060 *
1070 START   ...
```

I have written a short routine which will handle the initialization at the beginning of relocatable programs, as long as the program's entry point immediately follows, as in the sample program listed below.

The routine works by first jumping to the subroutine at $FF58, which is simply an RTS instruction. As Bob explained in the July AAL, this places the return address on the stack and then pops it back off again. The return address can then be found by reading the first two open bytes below the stack. The TSX instruction in line 1100 loads the offset to those two bytes into the X-register. Lines 1110-1130 load the bytes into the A- and Y-registers.

Now we have the address of the third byte of the JSR RETURN instruction - the MSB in Y and the LSB in A. What we need is the address of the program's entry point, which corresponds to the label START. To get that address, we must add in the length of the rest of the SETUP routine, that is, the difference between the address at START and the address in the Y- and A-registers.

This is handled in lines 1140-1170. Line 1150 adds the offset ($1B for this particular routine) to the low byte of the base address. The extra 1 in the ADC intruction is necessary because the address in Y and A is one less than the actual return address (corresponding to .1). Lines 1160-1170 check for a carry and adjust the high byte if necessary. The entry point address is then saved in the ampersand vector at $3F5-$3F7.

The same principle can be used to set up the monitor's control-Y vector at $3F8-$3FA. As a matter of fact, I usually use a macro with conditional assembly to set up whichever vector I need. Here's the macro:

```
1000        .MA VECTOR
1010        JSR $FF58
1020 :1     TSX
1030        LDY $100,X
1040        DEX
1050        LDA $100,X
1060        CLC
1070        ADC #:3-:1+1
1080        BCC :2
1090        INY
1100 :2     .DO ']1='Y   CTRL-Y?
1110        STA $3F9
1120        STY $3FA
1130        LDA #$4C
1140        STA $3F8
1150        .ELSE        OR &?
1160        STA $3F6
1170        STY $3F7
1180        LDA #$4C
1190        STA $3F5
1200        .FIN
1210        RTS
1220 :3
1230        .EM
```

Just include this definition at the beginning of your program. Then macro can then be called like this:

```
2000        >VECTOR,Y
2010 START  ...
```

to set the control-Y vector, or like this:

```
2000        >VECTOR,&
2010 START  ...
```

to set the ampersand vector. (Actually any character other than Y will result in setting the & vector.)

(Note: When I showed this macro to Bob I asked him if the .DO in line 1100 would really work. He looked at it for a minute and said, "yes, it sure will. The assembler's macros are even more powerful than I thought!"...Bill)

```
==============================================================================
DOCUMENT :AAL-8209:Articles:Directives.txt
==============================================================================
```

Assembler Directives......................Bob Sander-Cederlof

Of all the Apple assemblers on the market, it seems that no two have exactly the same list of assembler directives. Directives, also called "pseudo-ops", are used to control the assembly process and to define data in your programs. When you see a listing of an assembly language program in a magazine, or in this newsletter, or in a book on 6502 programming, you may have to translate the directives to fit the assembler you own.

All directives in the S-C Macro Assembler begin with a period. This helps to distinguish them visually from 6502 and SWEET-16 opcodes. This same convention is used by Carl Moser's (Eastern House Software) MAE assembler, by the MOS Technology and Rockwell assemblers, and some others. Most other assemblers use 3- or 4-character mnemonics beginning with a letter. Which combination of letters cause the assembler to perform a particular function is not standardized at all, but there are enough similarities to make programs readable once you learn the general techniques.

What follows is an alphabetical listing of all the directives I have encountered in various manuals and magazine-published programs. The assemblers represented are coded like this:

```
B = Big Mac              SC= S-C Macro Assembler
K = DOS Tool Kit         T = TED II
L = Lisa                 W = Weller's Assembler
M = Merlin
```

In each case I have given a brief description of the directive, and tried to show how to do the same thing in the S-C Macro Assembler. I suggest looking up the S-C directives in the reference manual if you are not sure exactly how to use them.

```
ADR    ADdRess                 L
```
Stores the expression as an address, low-order byte first.
SC: Use .DA directive

```
ASC    ASCii string definition L K T B M
```
SC: Use .AS or .AT directives.

```
AST    ASTerisks               T B M
```
Prints the number of asterisks specified on the listing. Used to save space in the source file.
SC: Not needed, because SC compresses repeated characters automatically.

```
BLK    BLinKing characters     L
```
Generates a string of characters in Apple's FLASH code.

SC:  Not available, but a combination of .AS and .HS directives will do the job.

```
BYT    BYTe data              L
```
Define data value, storing low-order byte only.
SC:  Use .DA with "#" before value.

```
CHK    CHecKsum               B M
```
SC:  Not available

```
CHN    CHaiN to next source   K
```
SC:  Use .IN directive.

```
CHR    Set CHaR for REP directive  K
```
Used to create fancy comments with repeated strings; saves space in source file.
SC:  Not necessary, because SC compresses repeated characters automatically.

```
.DA    DAta definition        L SC
```
Apparently Randy borrowed this one from me.  (See the reviews he wrote in Call APPLE some time ago.)

```
DA     Define Address         T B M
```
Defines a 16-bit value with low-byte first.
SC:  Use .DA directive.

```
DATA   DATA definition        W
```
Defines numeric and ASCII data bytes
SC:  Use .DA directive, preceding each value with "#".

```
DB     Data Byte              T
```
Defines a data value, only using the low byte of the expression.
SC:  Use .DA directive with "#" before the expression.

```
DBL    DouBLe precision data  W
```
Defines 16-bit data values.
SC:  Use .DA directive.

```
DBY    Double BYte data       L
```
Generates a 16-bit value and stores it high-byte first.
SC:  Not directly available, but use .DA as follows:
```
       .DA /expression,#expression
```

```
DCI    Define Characters Immed L K T B M
```
Stores string with sign bit of last byte opposite that of the rest of the bytes.
SC:  Use .AT directive.

```
DCM    DOS CoMmand            L
```
Issue a DOS command during assembly.  Usually used to BSAVE a section of the generated object code.
SC:  Use .TF directive

```
DDB    Define Double Byte      K B M
Defines a 16-bit value which is stored with the high-byte first.
SC:  Not directly available, but use .DA as follows:
       .DA /expression,#expression

DEND   Dummy END               K
Marks end of a dummy section (see DSECT).
SC:  Not available

DFB    DeFine Byte             K B M
Defines one or more bytes.
SC:  Use .DA, preceding each expression with "#".

DFC    DeFine Character        L (old version)
Data definition, byte expression list
SC:  Use .DA directive, preceding each expression with "#".

DFS    DeFine Storage          L
Reserve a block of bytes.  An optional second operand will cause the
reserved bytes to be set to the specified value.
SC:  Use .BS directive.  No option to set the reserved bytes to a
specified value.

DO     DO                      K B M
Start a conditional assembly block.
SC:  Use .DO directive.

DPH    DePHase                 L
Terminates a PHS directive.
SC:  Not available.

DS     Data Storage            T K B M
Reserve a block of bytes.
SC:  Use .BS directive.

DSC    Data SeCtion            L (old version)
Not sure what this was for.

DSECT  Dummy SECTion           K
Starts a block in which the object code bytes are not written on the
output file.
SC:  Not available.

DW     Define Word             K T
Defines a 16-bit value, with the low-byte stored first.
SC:  Use .DA directive.

.EL    ELse                    L SC
For conditional assembly.

ELSE   ELSE                    K B M
For conditional assembly, toggles the truth value from the DO
directive.
SC:  Use .ELSE directive.
```

```
END    END of program          L T B M W
Most assemblers REQUIRE an "END" directive at the end of the source
code.  S-C allows it but does not require it.
SC:  Use .EN directive.

ENTRY  ENTRY                   K
Indicates a symbol is to be made reference-able from other separately-
assembled modules.  To be used by a linking loader program, which
Apple does not provide.
SC:  Not available.

EOM    End Of Macro            B M
Marks end of a macro definition.
SC:  Use .EM directive.

EPZ    Equate Page Zero        L
label  EPZ expression
Defines the label to have the value of the expression, which must be
from $00 to $FF.  When EPZ-defined labels are used in address fields,
zero-page addressing mode will be used whenever possible.
SC:  Use .EQ directive.  SC automatically uses page-zero mode whenever
possible.

EQU    EQUate                  L T K B M W
label  EQU expression
Defines the label to have the value of the expression during the
assembly process.
SC:  Use .EQ directive.

ESP    End ScratchPad          W
Works with SPD to bracket a data section.
SC:  Not needed.

EXP    EXPansion of macros     B M
Controls whether macro expansion code is printed or not on the output
listing.
SC:  Use .LIST directive.

EXTRN  EXTeRNal                K
Indicates that a label is externally defined.  To be used with a
linking loader program, which Apple does not provide.
SC:  Not available.

.FI    end of conditional      L SC

FIN    end of conditional      K B M
SC:  Use .FIN directive.

FLS    FLaSH                   B M
Define a string in flashing mode.
SC:  Not available, but a combination of .AS and .HS directives will
do the job.
```

```
GEN    GENerate code listing   L
Turns on listing of all object code bytes.
SC:  Not available, object code listing is always on.

HBY    High BYte               L
Define one-byte data value, storing only the high-byte of an
expressions value.
SC:  Use .DA directive, writing "/" before the value.

HEX    HEXadecimal data        L T B M
label  HEX hexstring
SC:  Use .HS directive.

ICL    InCLude                 L
Really is a CHAIN to next source file.
SC:  Use .IN directive.

.IF    conditional assembly    L
SC:  Use .DO directive.

INV    INVerted characters     L B M
Generates a string of characters in Apple's INVERSE screen code.
SC:  Not available, but you can convert to hexadecimal and use .HS
directive.

LET    label reassignment      L
Same as EQU, except label can be redefined during assembly.
SC:  Not available.

LST    LiST option             L T K B M
Turn assembly listing on or off.
SC:  Use .LIST directive.

MAC    MACro definition        B M
Start a macro definition.
SC:  Use .MA directive.

MSB    Most Signficant Bit     K
Controls whether the ASC directive generates bytes with the first bit
set or clear.
SC:  Use .AS or .AT directives with or without the "-" before the
first delimiter to indicate the MSB value.

NLS    No List option          L
Turn assembly listing off.
SC:  Use .LIST OFF directive.

NOG    NO Generate             L
Turns off listing of all but first three bytes of any particular
source line.
SC:  Not available.

OBJ    OBJect address          L T B M
Set actual memory address for assembled object code to be stored in.
```

```
SC:  Use .TA directive.

ORG    ORiGin                  L T K B M
Set memory address program will execute at.
SC:  Use .OR directive.

PAG    PAGe eject on listing   L T B M
Sends control-L to listing device.
SC:  Use .PG directive.

PAGE   PAGE eject on listing   K
Sends control-L to listing device.
SC:  Use .PG directive.

PAU    PAUse and force error   L B M
SC:  Not available.

PHS    PHaSe                   L
Allows setting ORG without changing OBJ.  Terminated with DPH.
SC:  Not available.

PMC    Present MaCro           B M
Opcode to call a macro.
SC:  Not needed, macros are called by their own names.

PR#    Select printer slot     T
SC:  Select before assembly begins using DOS "PR#slot" command, or SC
"PRT" command.

REL    RELocatable object      K
Causes assembler to generate a relocation dictionary at the end of the
object file, for use by Apple's relocating loader.
SC:  Not available.

REM    REMark                  W
Used to indicate a comment line.
SC:  Use "*" in first column of label field.

REP    REPeated character      K
Generates a string of repetitions of the current CHR value on the
output listing.  Used to save space in the source file.
SC:  Not needed, because SC automatically compresses repeated
characters.

SAV    SAVe object code        B M
SC:  Use .TF directive.

SBTL   SuBTitLe                K
Provides a title line for the top of each page of the output listing.
SC:  Use .TI directive.

SKP    SKiP lines              K B M
Leaves a specified number of blank lines in the output listing.
SC:  Not available.
```

```
SPD    ScratchPaD              W
```
Works with ESP to bracket a data section.
SC:  Not needed.

```
STR    STRing                  L
```
Similar to Lisa's ASC except the first byte output is the length of the string.
```
SC:  labela .DA #labelb
            .AS /string/
     labelb .EQ *-labela-1
```

```
SYM    SYMbols                 T
```
Produces a symbol cross-reference table at end of assembly.
SC:  Not available, but can use Rak-Ware's XREF utility program.

```
TITL   TITLe
TTL    TiTLe                   L
```
Generates title line at top of each page of listing.
SC:  Use .TI directive.

```
TR     TRuncate object listing B M
```
Limit listing of object code to 3 bytes per source line.
SC:  Not available.

```
USR    USeR directive          L
```
An extra entry in the directive table for the user to use as he sees fit.
SC:  Use .US directive.

```
;      comment indicator       L
```
SC:  If ";" was in first column, use "*" instead.  If in later column, no special character is needed.

```
=      equate                  B M others
```
If written with label on left, this is the same as EQU and .EQ directives.  If written with "*" on the left, it is the same as ORG and .OR directives.

```
<<<                            B M
```
Alternate syntax for EOM.
SC:  Use .EM directive.

```
>>>                            B M
```
Alternate syntax for PMC.
SC:  Not needed, because macros are called by their own names.

Directives in Roger Wagner's Book

If you have been trying to learn using the S-C Assembler with Roger's book "Assembly Lines:  The Book", you may have been frustrated by his use of several assembler directives.  He discusses directives on pages 16-18, and 55.

On page 16, the first example of the use of directives has two errors. Lines 6 and 7 are written:

```
6  OBJ EQU $300
7  ORG EQU $300
```

```
But they should be:  6    OBJ $300
                     7    ORG $300
```

That is, OBJ and ORG are directives, not labels. The top two lines on page 21 are also incorrect, in that the ORG and OBJ directives were typeset to look like labels; they should be moved over to the opcode column, and the "$300" values to the operand column.

In all, Roger uses only five directives in his book: OBJ, ORG, EQU, ASC, and HEX. To use his programs in the S-C assembler, change:

```
 From                    To
----------------       ----------------
label EQU value        label .EQ value
label HEX hexdigits    label .HS hexdigits
      HEX hexdigits          .HS hexdigits
label ASC "characters" label .AS -"characters"
      ASC "characters"       .AS -"characters"
      OBJ $300 or $302     omit this line
      ORG $300 or $302       .OR $300 or $302
```

Note that the normal translation of "OBJ" is ".TA"; however, when the address is the same as the ORG/.OR address, it is not necessary to use OBJ/.TA. Furthermore, in the S-C Assemblers you must put the ".OR" line BEFORE the ".TA" line. In Roger's examples these two lines are reversed.

==============================================================================
DOCUMENT :AAL-8209:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 2 -- Issue 12 September, 1982

In This Issue...



Current Advertising Rates

Sorry, it is going up again. For the October 1982 issue the price will be $75 for a full page, $40 for a half page. To be included, I must receive your camera-ready copy by September 20th.

What if you move?

We mail the Apple Assembly Line by bulk mail, unless you have paid for First Class or Overseas postage. If you move, the post office will NOT forward AAL to your new address. Please let us know your new address as soon as you find out what it will be, so you will not miss a single issue!

Quarterly Disks

As you no doubt know, every three months we gather all the source code printed during the quarter on one disk. You can save countless hours of typing and proofreading for only $15 per quarter. Some have elected to establish a standing order with their credit card, or even to prepay for a year at a time.

=============================================================================
DOCUMENT :AAL-8209:Articles:Hardcore.txt
=============================================================================

About Hardcore Magazine....................Bob Sander-Cederlof

I have received several calls by subscribers who wonder about the ad from Hardcore magazine. The ad prices a subscription at $20, but does not say clearly what $20 buys.

To my knowledge, HARDCORE has published two issues so far: the first one about a year ago, and the second about six months ago.

Inside the front cover of the first issue you will find the following message:

"Attention Subscribers: Although presently only a quarterly magazine, HARDCORE Computing will go bimonthly and then monthly as soon as possible. Meanwhile, your one-year subscription is for the 4 quarterly issues plus 8 UPDATEs (printed on the other 8 months) and all ALERT Bulletins sent out whenever we feel information is too important to wait. The UPDATEs will be reprinted in part or in whole in the next magazine. The magazines, UPDATEs, and ALERT Bulletins comprise the subscription package."

I have talked with the publisher, Chuck Haight, several times on the phone. I believe he intends to fulfill every subscription, but he is having trouble getting the magazine out on a regular schedule. I asked him how often the magazine is published, and he answered "Very infrequently". He did re-assure me that a subscription buys four issues.

Note that Softkey Publishing is another company with the same people. Two callers indicated they are quite satisfied with the software they bought from Softkey.

=============================================================================
DOCUMENT :AAL-8209:Articles:New.Products.txt
=============================================================================

New Products

ES-CAPE: For really painless Applesoft programming, you need a complete line editor, global search and replace, automatic line numbers, and keyboard macros. At least. ES-CAPE gives you all these and more!

The retail price is $60, but AAL subscribers can get it for only $40 until the end of September. Hurry!

We wrote a nice little reference manual of about 22 pages, but ES-CAPE is so easy to use and remember that you won't need the book very long!

If you already purchased AED II (the earlier version of this editor), Bill Linn has an upgrade offer: Send him your disk plus $10, and you will get the new versions (both regular and language card), the manual, and the reference card.

68000 Macro Cross Assembler: Not content with producing only three cross assemblers based on the S-C Macro Assembler, Bobby Deen has now completed the biggest one of all! This one costs $50, and allows you to assemble Motorola 68000 source programs in your Apple, with all the friendly features of the S-C package.

SYNASSEMBLER: Synapse Software has just started marketing a conversion of the S-C Assembler II Version 4.0 for the Atari 800 or 400. You need 48K RAM and at least one disk drive. The conversion was done by Steve Hales, of Livermore, California. He added global replace and copy commands, so this version falls somewhere between the Apple version 4.0 and the new Macro version. It assembles at about 6500 lines per minute, which is from 50 to over 100 times faster than the Atari ASM/ED program.

Since the Atari does not have nice monitor commands built-in, like the Apple does, Steve added a complete set of monitor commands to SYNASSEMBLER. They look exactly like the Apple monitor commands, except that he added some new ones to allow reading and writing a range of disk sectors, delete the tape I/O commands, and included the old Step and Trace commands which were in Apples before the Autostart ROM.

The price is only $49.95 on disk. A ROM version is available by special order for $89.95. I will carry these, if you want to order from me.

An Apple Bibliography

Bob Broedel has been keeping track of all the books, magazines, etc. that are of interest to Apple owners. The last time I saw the list

(May 1982), it was ten pages, two columns. Each entry includes all the bibliographic data Bob knows, so that you can find the items you want.

This is the most complete list I have ever seen. If you want a copy, he will send you one for $2. Write to Bob Broedel, P. O. Box 20049, Tallahassee, FL 32304.

==============================================================================
DOCUMENT :AAL-8209:Articles:Read.Paddles.txt
==============================================================================

No More Paddle Interaction..........................Mike Laumer

While working on the FLASH! Integer BASIC Compiler I ran into a nasty little problem because the compiled code ran too fast!   That's right, too fast.  The old problem with reading the game paddles too soon after one another rose to byte (punny huh!) me once again.

Basically the game paddle problem is that they are read with a variable time delay loop.  Because one paddle may read significantly faster than another, and the paddles have only one trigger to fire all four of the paddles, you might process the data fast enough to be ready to read the next paddle before it has finished its previous time delay.  This problem is real and occurs in many of the game programs to be found on the Apple.  Even Raster Blaster has the problem in its jittery ball release thrust adjuster.

In the example below paddle 0 and 1 are triggered by the $C070 paddle I/O trigger address.  But because paddle 0 has a smaller value, it finishes before paddle 1.  If you read one paddle after another with little other processing then one paddle seems to affect the value of the other one.  Many programmers have shown this problem to their dealer thinking that they have found a new bug in the Apple but the only problem (if one exists) is the lack of independent paddle triggers for each of the four paddles.

The problem appears if you use the following BASIC program and play with the paddle adjustments.  Turn paddle 1 to the middle of its scale and paddle 0 to the low end of its scale and you will see changing paddle 0 affects the value read for paddle 1.  You will find that paddle 1 will vary by 20-40 counts without even touching it.

```
   10 PRINT PDL(0),PDL(1) : GOTO 10

      +-------+
     -|       |----------------     paddle 0

      +--------------+
     -|              |----------    paddle 1
     ^               ^
     :               : paddle expires
     :
     paddles are triggered at this time
```

So what can be done about the problem?  What I did is design a routine that reads the paddle without triggering it and waits for the paddle to shut off.  This is easily done by calling the monitor paddle read routine at $FB21, skipping the trigger instruction at $FB1E.  This takes care of much of the problem, but I still found it necessary to add a tiny delay loop before triggering the paddle.  The extra delay is probably due to the remaining charge in the internal capacitor in the timer chip.

The assembly language routine which follows is basically what I added to the FLASH! compiler runtime package to take care of its being too fast for its own good!  This explains 14 of the 36,000 bytes of object code in the FLASH! Compiler system.  There is also a DEMO program which reads both paddles and displays the values in hexadecimal so you can test the routine.

```
==============================================================================
DOCUMENT :AAL-8209:Articles:Screen.Tricks.txt
==============================================================================
```

Some Fast Screen Tricks....................Bob Sander-Cederlof

Sometimes the standard Apple Monitor screen functions are too slow. No reflection on Steve Wozniak, because he wrote them to be general and compact rather than quick.

I am thinking particular of the screen clear (HOME to Applesoft users) and the screen scroll subroutines. They were both written to operate on a text window, not necessarily the whole screen. But most of the time you do want to clear or scroll the whole screen.

The primary text screen memory is mapped into the addresses from $400 through $7FF, but not in an obvious or straightforward way. This table shows the actual memory addresses for each screen line:

```
Line Addresses     Line Addresses     Line Addresses
  0  $400-$427       8  $428-$44F      16  $450-$477
  1  $480-$4A7       9  $4A8-$4CF      17  $4D0-$4F7
  2  $500-$527      10  $528-$54F      18  $550-$577
  3  $580-$5A7      11  $5A8-$5CF      19  $5D0-$5F7
  4  $600-$627      12  $628-$64F      20  $650-$677
  5  $680-$6A7      13  $6A8-$6CF      21  $6D0-$6F7
  6  $700-$727      14  $728-$74F      22  $750-$777
  7  $780-$6A7      15  $7A8-$7CF      23  $7D0-$7F7
```

Note that 120 consecutive bytes are used for three text lines spaced at an 8-line interval. Then 8 bytes are not used. Then the next 120, and so on. Those 8 sets of 8 bytes that are not used by the screen mapping are used by peripheral cards and DOS for temporary storage. In the standard Apple Monitor subroutines, a subroutine named BASCALC at $FBC1 calculates the starting address for a specified line. Then the various screen functions use that address, which is kept up-to-date in BASL,BASH ($28,29).

In the listing that follows, I have included fast subroutines to clear the entire text screen (CLEAR); to set the entire text screen to whatever character is in the A-register (SET); to clear the entire Lo-Res Graphics screen (GCLEAR); and to scroll the entire text screen up one line. For demonstration purposes, I also wrote routines to set the entire screen to each value from $00 through $FF; to alternate the screen between solid black and solid white until a key is pressed; to scroll end-around, placing the old top line on the bottom of the screen while moving the rest of the lines up; and to continuously scroll end-around until a key is pressed.

For comparison, I counted that the Wozniak's screen clear takes 15537 microseconds; mine takes only 5410 microseconds. The fastest possible would be one LDA #$A0 followed by 960 "STA $xxx" and an RTS; that would take 3848 microseconds.  (All these times round off the Apple's cycle time to one microsecond; actually it is a little faster.)

==============================================================================
DOCUMENT :AAL-8209:Articles:Underline.Fix.txt
==============================================================================

A Note on the Underline Cursor.............Bob Sander-Cederlof

Bill Linn's "Blinking Underline Cursor" program generated a lot of interest. However, Allan Blackburn from Fort Worth had a problem with it:

"It works just fine, until you hit RESET or re-boot...then it must be BRUN again to get it back. You can't enter monitor and type 300G, or use CALL 768 from Applesoft. Why doesn't calling the routine reset KSWL and KSWH? It should, but I always end up with $9E81 there. Even though lines 1210-1250 store $09 in $38 and $03 in $39, it seems they never get there. Can you explain this? Please?

Sure, Allan. Line 1250 needs to be changed from RTS to JMP $3EA.

This is a common problem. I had it myself back when DOS first came out. For the first year or so we only had a tiny preliminary manual, and the subject wasn't covered. Now the DOS manual is so large we forget to read it or where to find the information. Look on pages 100-105 of the DOS manual and you will find a full explanation.

Briefly, here is what happens. Lines 1210-1250 DO store the address $309 into #38 and $39. But the next time you print a character, DOS gets control and stores its own input address right on top of yours. DOS's input address is $9E81.

The same thing happens in Applesoft programs if you use IN#1 (for example) instead of PRINT CHR$(4)"IN#1", and then print a character. Note 7b on page 105 tells about CALL 1002, which is $3EA.

```
==============================================================================
DOCUMENT :AAL-8209:Articles:VidexPatchPatch.txt
==============================================================================

4020 *--------------------------------
4025 V.BASEL .EQ $478+SLOTNUM
4030 V.BASEH .EQ $4F8+SLOTNUM
4035 V.CHORZ .EQ $578+SLOTNUM
4040 V.XSAV1 .EQ $402
4045 V.OLDCHAR .EQ $678
4050 *
4055 V.DEV0  .EQ SLOTNUM*16+$C080
4060 V.DISP0 .EQ $CC00
4065 V.DISP1 .EQ $CD00
4070 *--------------------------------
4075 *
4080 RDKEY  LDA KEYBOARD
4085        BPL RDKEY
4090        STA KEYSTROBE
4095        ORA #$80
4100        CMP #$81     Shift lock?
4105        BNE .1
4110        .DO LCVERSION
4115        JSR UNPROTECT.LC.RAM
4120        .FIN
4125        LSR SCM.SHIFT.FLAG
4130        BPL .2       Return with errant key
4135 .1     CMP #$9A     Shift unlock?
4140        BNE CTRLU    No, return with key
4145        .DO LCVERSION
4150        JSR UNPROTECT.LC.RAM
4155        .FIN
4160        SEC
4165        ROR SCM.SHIFT.FLAG
4170 .2     LDA #$96     Return with errant key
4175        .DO LCVERSION
4180        BIT $C080    Reprotect LC RAM
4185        RTS
4190 *
4195 UNPROTECT.LC.RAM
4200        BIT $C083    Enable Bank 2
4205        BIT $C083
4210        .FIN
4215        RTS
4220 *
4225 CTRLU  CMP #$95     CTRL-U COPY KEY
4230        BNE .3
4235        STX $400
4240        STY $401
4245        LDA V.CHORZ
4250        JSR PSNCALC
4255        BCS .1
4260        LDA V.DISP0,X
```

```
4265        BCC .2
4270 .1     LDA V.DISP1,X
4275 .2     ORA #$80
4280        STA V.OLDCHAR
4285        LDX $400
4290        LDY $401
4295 .3     RTS
4300 *
4305 PSNCALC CLC
4310        ADC V.BASEL
4315        STA V.XSAV1
4320        LDA #0
4325        ADC V.BASEH
4330        LSR
4335        PHP
4340        AND #3
4345        ASL
4350        ASL
4355        TAY
4360        LDA V.DEV0,Y
4365        PLP
4370        LDX V.XSAV1
4375        RTS
4380 *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8209:Articles:VidexRtArrow.txt
==========================================================================

Right arrow for the VIDEX patches................Mike Laumer

The VIDEX 80 column board patches for the S-C Macro Assembler in last months Apple Assembly Line was a welcome article for me.  You see I bought a VIDEX board last November but have no software to run it. I've been planning to write a program development editor similar to the one I used at Texas Instruments, but so far I haven't had the time between the FLASH! compiler, MIKE'S MAGIC MATRIX and the American Heart association CPR Training system.

The patches were very usable, but a major problem still existed to prevent my use on a regular basis.  The right arrow key would not copy characters from the VIDEX screen.  Try to copy a file name from your catalog with that limitation!

I knew it could be done, because the VIDEX software in ROM has to do that function.  Don Taylor mentioned last month that he didnt know the right routine to call and his ROM differed from the listing in the VIDEX manual.  My listing was a little off also from my ROM, but I didn't care becase I wasn't going to call the ROM routines.

I used the VIDEX manual's listings to locate the section that performed the copy-character-from-screen function and used similar code in the RDKEY routine of last month's VIDEX patches for the Macro assembler.  The 'BNE' to '.3' was changed to go to 'CTRLU' and the copy function coded to process the right arrow key for the VIDEX 80 column board.

I needed two temporary variables to save the X- and Y- registers, so I used the first two bytes of the normal Apple text screen at $400 and $401.  Another temporary variable is at $402.  Since the normal Apple text display is not operative while the VIDEX is enabled you can use it for temporary variable space without it affecting the screen display.  If you try a trick like this some time, you must be careful because some of the monitor routines like HOME and SCROLL can easily zap your storage when you least expect it.

With this new capability of the right arrow key functioning as expected, I am able to use the VIDEX patches often in my software development work.  But there are a few problems left yet to solve that I didn't get to look into before writing this article.  They are:

1.  A RETURN key should clear to the end of line on line input, but not EDIT input.

2.  The control character display features are not handled very well by the VIDEX patches.

3.  The patches blow up on Reset.  (I think.)

4. The patches blow up on INT or FP commands.

5. The patches don't work very well when you use MNTR command.

6. All calls to $FC9C (the Monitor clear to end of line routine) should send $9D to the VIDEX board.

7. Right arrow, left arrow, and any printing key cause the entire EDIT line to be redisplayed. The flicker is somewhat annoying.

The listing that follows should replace lines 4020 through 4420 of the listing on pages 21 and 22 of the August 1982 issue.

The source code on the AAL Quarterly Disk #8 will have these lines already merged with Don Taylor's patches.

```
==========================================================================
DOCUMENT :AAL-8209:DOS3.3:S.CatalogArr.txt
==========================================================================

 1000  *SAVE S.CATALOG ARRANGER
 1010  *--------------------------------
 1020         .OR $803
 1030         .TF CATALOG ARRANGER
 1040  *--------------------------------
 1050  POINTER     .EQ 0
 1060  *
 1070  MON.CV      .EQ $25
 1080  PREG        .EQ $48
 1090  *
 1100  DOS.RESTART .EQ $3D0
 1110  DOS.RWTS    .EQ $3D9
 1120  *
 1130  CORNER      .EQ $7D0
 1140  *
 1150  KEYBOARD    .EQ $C000
 1160  KEYSTROBE   .EQ $C010
 1170  *
 1180  DOS.SIZEOUT .EQ $AE42
 1190  DOS.PRNTERR .EQ $A702
 1200  DOS.TYPTABL .EQ $B3A7
 1210  IOB         .EQ $B7E8
 1220  IOB.SLOT    .EQ $B7E9
 1230  IOB.DRIVE   .EQ $B7EA
 1240  IOB.VOLUME  .EQ $B7EB
 1250  IOB.TRACK   .EQ $B7EC
 1260  IOB.SECTOR  .EQ $B7ED
 1270  IOB.BUFFER  .EQ $B7F0,F1
 1280  IOB.COMMAND .EQ $B7F4
 1290  IOB.ERROR   .EQ $B7F5
 1300  IOB.OSLOT   .EQ $B7F7
 1310  IOB.ODRIVE  .EQ $B7F8
 1320  *
 1330  * MONITOR CALLS
 1340  *
 1350  MON.VTAB    .EQ $FC22
 1360  MON.CLREOP  .EQ $FC42
 1370  MON.HOME    .EQ $FC58
 1380  MON.PRBYTE  .EQ $FDDA
 1390  MON.COUT1   .EQ $FDF0
 1400  MON.SETINV  .EQ $FE80
 1410  MON.SETNORM .EQ $FE84
 1420  *
 1430  *  SYMBOLIC CONSTANTS
 1440  *
 1450  ZERO         .EQ 0
 1460  READ         .EQ 1
 1470  WRITE        .EQ 2
 1480  LINE.COUNT   .EQ 22
```

```
1490  ENTRY.LENGTH .EQ 35
1500  RETURN       .EQ $8D
1510  SPACE        .EQ $A0
1520  *--------------------------------
1530  SETUP
1540         LDA IOB.OSLOT     SET SLOT AND
1550         STA SLOT          DRIVE TO WHERE
1560         LDA IOB.ODRIVE    WE CAME FROM
1570         STA DRIVE
1580
1590         LDA #ZERO         INITIALIZE
1600         STA VOLUME        VARIABLES
1610         STA NUMBER.OF.ELEMENTS
1620         STA MOVING.FLAG
1630         LDA #$FF
1640         STA ACTIVE.ELEMENT
1650
1660         JSR BUILD.ARRAY.TABLE
1670         JSR READ.CATALOG
1680         JSR MON.HOME
1690  *--------------------------------
1700  DISPLAY.AND.READ.KEY
1710         JSR DISPLAY.ARRAY
1720
1730  .1     LDA KEYBOARD
1740         BPL .1
1750         STA KEYSTROBE
1760         CMP #$95             -->
1770         BEQ HANDLE.RIGHT.ARROW
1780         CMP #$88             <--
1790         BEQ HANDLE.LEFT.ARROW
1800         CMP #$9B             ESC
1810         BEQ HANDLE.ESC
1820         CMP #RETURN
1830         BEQ HANDLE.RETURN
1840         CMP #$C2             B
1850         BEQ HANDLE.B         BEGINNING
1860         CMP #$C5             E
1870         BEQ HANDLE.E         END
1880         CMP #$D2             R
1890         BEQ HANDLE.R         READ CATALOG
1900         CMP #$D7             W
1910         BEQ HANDLE.W         WRITE CATALOG
1920         JMP .1            NONE OF THE ABOVE
1930  *--------------------------------
1940  HANDLE.RIGHT.ARROW
1950  *  MOVE UP ONE ELEMENT
1960         JSR CHECK.FOR.END.OF.ARRAY
1970         BCS .2            DO NOTHING IF ALREADY AT END
1980         BIT MOVING.FLAG   SKIP SWAP IF
1990         BPL .1            NOT MOVING
2000         JSR MOVE.ELEMENT.UP
2010  .1     INC ACTIVE.ELEMENT  FOLLOW IT UP
2020  .2     JMP DISPLAY.AND.READ.KEY
```

```
2030 *--------------------------------
2040 HANDLE.LEFT.ARROW
2050 *  MOVE DOWN ONE ELEMENT
2060        JSR CHECK.FOR.BEGINNING.OF.ARRAY
2070        BCS .2            IF AT BEGINNING, DO NOTHING
2080        BIT MOVING.FLAG   IF NOT MOVING,
2090        BPL .1            SKIP SWAP
2100        JSR MOVE.ELEMENT.DOWN
2110 .1     DEC ACTIVE.ELEMENT
2120 .2     JMP DISPLAY.AND.READ.KEY
2130 *--------------------------------
2140 HANDLE.B
2150 *  MOVE CURSOR TO BEGINNING OF ARRAY
2160 .1     JSR CHECK.FOR.BEGINNING.OF.ARRAY
2170        BCS .3    DO NOTHING IF AT BEGINNING
2180        BIT MOVING.FLAG
2190        BPL .2
2200        JSR MOVE.ELEMENT.DOWN
2210 .2     DEC ACTIVE.ELEMENT
2220        BPL .1
2230 .3     JMP DISPLAY.AND.READ.KEY
2240 *--------------------------------
2250 HANDLE.E
2260 *  MOVE CURSOR TO END OF ARRAY
2270 .1     JSR CHECK.FOR.END.OF.ARRAY
2280        BCS .3
2290        BIT MOVING.FLAG
2300        BPL .2
2310        JSR MOVE.ELEMENT.UP
2320 .2     INC ACTIVE.ELEMENT
2330        BPL .1       ...ALWAYS
2340 .3     JMP DISPLAY.AND.READ.KEY
2350 *--------------------------------
2360 HANDLE.W
2370 *  WRITE CATALOG TO DISK
2380        JSR WRITE.CATALOG
2390        JMP DISPLAY.AND.READ.KEY
2400 *--------------------------------
2410 HANDLE.RETURN
2420 *  TOGGLE MOVING FLAG
2430 *  =FF IF MOVING
2440 *  =0  IF NOT
2450        LDA MOVING.FLAG
2460        EOR #$FF
2470        STA MOVING.FLAG
2480        JMP DISPLAY.AND.READ.KEY
2490 *--------------------------------
2500 HANDLE.ESC
2510 *  EXIT PROGRAM
2520        JMP DOS.RESTART
2530 *--------------------------------
2540 HANDLE.R
2550 *  READ NEW CATALOG
2560        JMP SETUP  RESTART PROGRAM
```

```
*--------------------------------
READ.CATALOG
       JSR READ.VTOC
       JSR POINT.TO.FIRST.CATALOG.SECTOR
.1     JSR READ.CATALOG.SECTOR
       BCS .4                  .CS. IF END OF CHAIN

*  MOVE CATALOG SECTOR INTO ARRAY
*   X STEPS THROUGH BUFFER, $B-$FF
*   Y STEPS THROUGH ENTRY, 0-$23
       LDX #$B
.2     LDA CATALOG.BUFFER,X
       BEQ .4                  END OF CATALOG?
       INC ACTIVE.ELEMENT      NO, WE HAVE
       INC NUMBER.OF.ELEMENTS  A NEW ENTRY
       LDA ACTIVE.ELEMENT
       JSR POINT.TO.A          SET POINTER
       LDY #ZERO
.3     LDA CATALOG.BUFFER,X
       STA (POINTER),Y
       INX
       BEQ .1            END OF BUFFER?
*                        IF SO, READ NEW SECTOR
       INY
       CPY #ENTRY.LENGTH END OF ENTRY?
       BCC .3            NO, KEEP GOING
       BCS .2            YES, GET NEXT ONE

.4     LDA ACTIVE.ELEMENT
       CLC          GO ONE PAST
       ADC #1       LAST ELEMENT
       ASL          AND STORE
       TAY          TWO ZEROES
       LDA #ZERO
       STA ARRAY.TABLE,Y
       STA ARRAY.TABLE+1,Y
       STA ACTIVE.ELEMENT
       RTS
*--------------------------------
READ.VTOC
       LDA #ZERO
       STA SECTOR
       LDA #$11
       STA TRACK
       LDA #VTOC.BUFFER
       STA BUFFER
       LDA /VTOC.BUFFER
       STA BUFFER+1
       LDA #READ
       STA COMMAND
       JMP RWTS.CALLER
*--------------------------------
READ.CATALOG.SECTOR
       LDA CATALOG.BUFFER+1   GET NEXT TRACK
```

```
3110        BEQ .1       END OF CATALOG CHAIN?
3120        STA TRACK
3130        LDA CATALOG.BUFFER+2   GET NEXT SECTOR
3140        STA SECTOR
3150        LDA #CATALOG.BUFFER
3160        STA BUFFER
3170        LDA /CATALOG.BUFFER
3180        STA BUFFER+1
3190        LDA #READ
3200        STA COMMAND
3210        JSR RWTS.CALLER
3220        CLC
3230        RTS
3240
3250 * SET CARRY TO SHOW END-OF-CHAIN
3260 .1     SEC
3270        RTS
3280 *--------------------------------
3290 WRITE.CATALOG
3300        LDA #$FF
3310        STA ACTIVE.ELEMENT
3320        JSR POINT.TO.FIRST.CATALOG.SECTOR
3330 .1     JSR READ.CATALOG.SECTOR
3340        LDX #$B
3350 .2     INC ACTIVE.ELEMENT
3360        LDA ACTIVE.ELEMENT
3370        JSR POINT.TO.A
3380        BCS .5            .CS. IF AT END OF TABLE
3390        LDY #ZERO
3400 .3     LDA (POINTER),Y
3410        STA CATALOG.BUFFER,X
3420        INX
3430        BEQ .4            END OF BUFFER?
3440        INY
3450        CPY #ENTRY.LENGTH END OF ENTRY?
3460        BCC .3            NO, KEEP GOING
3470        BCS .2            YES, GET NEXT ONE
3480
3490 .4     JSR WRITE.CATALOG.SECTOR
3500        JMP .1            AND READ THE NEXT SECTOR
3510
3520 *  FILL THE REST OF THE BUFFER WITH ZEROES
3530 .5     LDA #ZERO
3540 .6     STA CATALOG.BUFFER,X
3550        INX
3560        BNE .6
3570
3580        JSR WRITE.CATALOG.SECTOR
3590        LDA #ZERO
3600        STA ACTIVE.ELEMENT
3610        JMP DISPLAY.AND.READ.KEY
3620 *--------------------------------
3630 WRITE.CATALOG.SECTOR
3640        LDA #WRITE        WRITE THE SECTOR
```

```
3650        STA COMMAND       BACK JUST WHERE
3660        JMP RWTS.CALLER   IT CAME FROM
3670 *--------------------------------
3680 POINT.TO.FIRST.CATALOG.SECTOR
3690 *  GET THE FIRST TRACK AND SECTOR FROM THE VTOC
3700        LDA VTOC.BUFFER+1
3710        STA CATALOG.BUFFER+1
3720        LDA VTOC.BUFFER+2
3730        STA CATALOG.BUFFER+2
3740        RTS
3750 *--------------------------------
3760 DISPLAY.ARRAY
3770        LDA #ZERO         START AT
3780        STA MON.CV        TOP OF
3790        JSR MON.VTAB      SCREEN
3800        LDA ACTIVE.ELEMENT
3810        SEC
3820        SBC #LINE.COUNT/2
3830        BPL .1            IF RESULT IS +, USE IT
3840        LDA #ZERO         OTHERWISE, USE ZERO
3850 .1     TAX               X KEEPS TRACK OF
3860 .2     TXA               WHERE WE ARE
3870        CMP ACTIVE.ELEMENT
3880        BNE .3
3890        PHA
3900        JSR MON.SETINV    INVERT ACTIVE ELEMENT
3910        PLA
3920 .3     JSR POINT.TO.A    SET POINTER
3930        BCS .5            .CS. IF AT END OF TABLE
3940        JSR INTERPRET.ENTRY  WRITE A LINE
3950        LDA #RETURN
3960        JSR MON.COUT1
3970        JSR MON.SETNORM   RESTORE NORMAL
3980        INX
3990        LDA MON.CV
4000        CMP #LINE.COUNT   END OF SCREEN?
4010        BCC .2            IF NOT, DO ANOTHER LINE
4020 .5     JSR MON.CLREOP    CLEAR TO END OF PAGE
4030        JSR MON.SETNORM   JUST IN CASE
4040        BIT MOVING.FLAG
4050        BPL .6
4060        JSR SHOW.MOVING.FLAG  IF MOVING
4070 .6     RTS
4080 *--------------------------------
4090 INTERPRET.ENTRY
4100        LDY #ZERO
4110        LDA (POINTER),Y   DELETED?
4120        BPL .1            MINUS IF YES
4130        LDA #$AD     -
4140        BMI .3       ...ALWAYS
4150 .1     LDY #2
4160        LDA (POINTER),Y   LOCKED?
4170        BPL .2            MINUS IF YES
4180        LDA #$AA     *
```

```
4190        BMI .3        ...ALWAYS
4200  .2    LDA #SPACE        NEITHER DELETED NOR LOCKED
4210  .3    JSR MON.COUT1
4220        LDY #2
4230        LDA (POINTER),Y   GET FILE TYPE
4240        AND #$7F          MAKE POINTER
4250        LDY #7            INTO DOS'S
4260        ASL               TYPE TABLE
4270  .4    ASL               (ROUTINE BORROWED
4280        BCS .5            FROM DOS, $ADE8-ADF9)
4290        DEY
4300        BNE .4
4310  .5    LDA DOS.TYPTABL,Y
4320        JSR MON.COUT1     DISPLAY TYPE
4330        LDA #SPACE
4340        JSR MON.COUT1
4350        LDY #$21
4360        LDA (POINTER),Y    SET UP FOR
4370        STA $44            ROUTINE TO
4380        INY                DISPLAY FILE
4390        LDA (POINTER),Y    SIZE
4400        STA $45
4410        JSR DOS.SIZEOUT    DO IT
4420        LDA #SPACE
4430        JSR MON.COUT1
4440        LDY #3
4450  .6    LDA (POINTER),Y    GET A CHARACTER
4451        CMP #SPACE         CONTROL?
4452        BCS .7             NO, GO ON
4453        AND #$7F           YES, MAKE IT INVERSE
4460  .7    JSR MON.COUT1      DISPLAY IT
4470        INY
4480        CPY #33            DONE WITH FILE NAME?
4490        BCC .6
4500        RTS
4510  *---------------------------------
4520  SHOW.MOVING.FLAG
4530        LDY #5        PUT INVERSE
4540  .1    LDA QMOVING,Y "MOVING" AT
4550        STA CORNER,Y  BOTTOM OF
4560        DEY           SCREEN
4570        BPL .1
4580        RTS
4590  *---------------------------------
4600  RWTS.CALLER
4610        LDA SLOT     TRANSFER
4620        STA IOB.SLOT  VALUES
4630        LDA DRIVE      INTO
4640        STA IOB.DRIVE   IOB
4650        LDA VOLUME
4660        STA IOB.VOLUME
4670        LDA TRACK
4680        STA IOB.TRACK
4690        LDA SECTOR
```

```
4700         STA IOB.SECTOR
4710         LDA COMMAND
4720         STA IOB.COMMAND
4730         LDA BUFFER
4740         STA IOB.BUFFER
4750         LDA BUFFER+1
4760         STA IOB.BUFFER+1
4770         LDA #$00
4780         STA IOB.ERROR
4790  *
4800         LDY #IOB     LOAD IOB
4810         LDA /IOB     ADDRESS
4820         JSR DOS.RWTS     CALL RWTS
4830         LDA #$00
4840         STA PREG    SOOTHE MONITOR
4850         BCS ERROR.HANDLER
4860         RTS
4870  *--------------------------------
4880  ERROR.HANDLER
4890         LDA #$87     BELL
4900         JSR MON.COUT1     RING
4910         JSR MON.COUT1        ING
4920         JSR MON.COUT1          ING
4930         LDA #23
4940         STA MON.CV        USE LINE BELOW DISPLAY
4950         JSR MON.VTAB
4960         LDX #8
4970         JSR DOS.PRNTERR   DISPLAY "I/O ERROR"
4980         JMP DOS.RESTART   EXIT PROGRAM
4990  *--------------------------------
5000  BUILD.ARRAY.TABLE
5010         LDA #CATALOG.ARRAY  SET FIRST ENTRY
5020         STA ARRAY.TABLE     TO POINT TO
5030         LDA /CATALOG.ARRAY  START OF
5040         STA ARRAY.TABLE+1   ARRAY
5050         LDX #2
5060  .1     LDA ARRAY.TABLE-2,X MAKE EACH
5070         CLC                 SUCCESSIVE
5080         ADC #ENTRY.LENGTH   ENTRY $23
5090         STA ARRAY.TABLE,X   LARGER THAN
5100         LDA ARRAY.TABLE-1,X THE LAST
5110         ADC #ZERO
5120         STA ARRAY.TABLE+1,X
5130         INX
5140         INX
5150         CPX #$FE            127 ENTRIES YET?
5160         BNE .1
5170         LDA #ZERO           END TABLE
5180         STA ARRAY.TABLE,X   WITH TWO
5190         STA ARRAY.TABLE+1,X ZEROES
5200         RTS
5210  *--------------------------------
5220  POINT.TO.A
5230         ASL                 MAKE (A) INTO INDEX
```

```
5240        TAY
5250        LDA ARRAY.TABLE,Y   CHECK FOR TWO
5260        ORA ARRAY.TABLE+1,Y CONSECUTIVE
5270        BEQ .1              ZERO BYTES
5280        LDA ARRAY.TABLE,Y
5290        STA POINTER         PUT TABLE ENTRY
5300        LDA ARRAY.TABLE+1,Y INTO POINTER
5310        STA POINTER+1
5320        CLC
5330        RTS
5340
5350 .1     SEC                 END OF TABLE
5360        RTS
5370 *---------------------------------
5380 CHECK.FOR.END.OF.ARRAY
5390 *  RETURNS CARRY SET IF AT END
5400 *     "      "   CLEAR IF NOT
5410        LDA ACTIVE.ELEMENT
5420        CLC
5430        ADC #1
5440        CMP NUMBER.OF.ELEMENTS
5450        RTS
5460 *---------------------------------
5470 CHECK.FOR.BEGINNING.OF.ARRAY
5480        LDA ACTIVE.ELEMENT
5490        BNE .1
5500        SEC          ACTIVE = 0, WE ARE AT BEGINNING
5510        RTS
5520
5530 .1     CLC          NONZERO, WE'RE OKAY
5540        RTS
5550 *---------------------------------
5560 MOVE.ELEMENT.UP
5570        LDA ACTIVE.ELEMENT
5580        ASL          MAKE INDEX INTO TABLE
5590        TAX
5600        LDY #2       DO THIS TWICE, FIRST LO, THEN HI
5610 .1     LDA ARRAY.TABLE,X
5620        PHA
5630        LDA ARRAY.TABLE+2,X
5640        STA ARRAY.TABLE,X
5650        PLA
5660        STA ARRAY.TABLE+2,X
5670        INX          NOW DO HIGH BYTES
5680        DEY
5690        BNE .1       DONE?
5700        RTS
5710 *---------------------------------
5720 MOVE.ELEMENT.DOWN
5730        LDA ACTIVE.ELEMENT
5740        ASL
5750        TAX
5760        LDY #2
5770 .1     LDA ARRAY.TABLE,X
```

```
5780        PHA
5790        LDA ARRAY.TABLE-2,X
5800        STA ARRAY.TABLE,X
5810        PLA
5820        STA ARRAY.TABLE-2,X
5830        INX
5840        DEY
5850        BNE .1
5860        RTS
5870 *--------------------------------
5880 QMOVING
5890 *  INVERSE "MOVING"
5900        .HS 0D0F16090E07
5910 *--------------------------------
5920 SLOT    .BS 1  (USUALLY 6)
5930 DRIVE   .BS 1  (USUALLY 1)
5940 VOLUME  .BS 1  (0 = ANY)
5950 TRACK   .BS 1  (USUALLY $11)
5960 SECTOR  .BS 1  (0 TO F)
5970 BUFFER  .BS 2  (VARIES)
5980 COMMAND .BS 1  (1 OR 2)
5990
6000 NUMBER.OF.ELEMENTS .BS 1  (1 TO N)
6010 ACTIVE.ELEMENT     .BS 1  (0 TO N-1)
6020 MOVING.FLAG        .BS 1  (0 OR FF)
6030 *--------------------------------
6040 END.OF.PROGRAM
6050
6060 VTOC.BUFFER     .EQ END.OF.PROGRAM
6070 CATALOG.BUFFER  .EQ END.OF.PROGRAM+$100
6080 ARRAY.TABLE     .EQ END.OF.PROGRAM+$200
6090 CATALOG.ARRAY   .EQ END.OF.PROGRAM+$300
```

==================================================================
DOCUMENT :AAL-8209:DOS3.3:S.PdlWOIntAct.txt
==================================================================

```
 1000  *--------------------------------
 1010  * READ PADDLES
 1020  * PADDLE NUMBER IN A REGISTER
 1030  * USES A,X,Y REGISTERS
 1040  * RETURNS PADDLE VALUE IN Y REGISTER
 1050  *--------------------------------
 1060  * THIS PADDLE READ ROUTINE
 1070  * WILL PREVENT ALMOST ALL PADDLE
 1080  * INTERACTION PROBLEMS DUE TO
 1090  * ONLY 1 PADDLE TRIGGER FOR
 1100  * ALL PADDLES.
 1110  *--------------------------------
 1120  MON.PREAD .EQ $FB1E
 1130  *--------------------------------
 1140  READP  AND #3           PDL 0 - 3
 1150         TAX
 1160         JSR MON.PREAD+3  MAKE SURE PADDLE IS READY
 1170         LDY #0
 1180  .1     DEY              KLUDGE DELAY FOR
 1190         BNE .1              CIRCUIT READY
 1200         JMP MON.PREAD    TRIGGER AND READ
 1210  * PADDLE RESULT IN Y REGISTER
 1220  *--------------------------------
 1230  DEMO   LDA #0       READ PADDLE 0
 1240         STA $24      HTAB COLUMN 1
 1250         JSR READP
 1255         TYA          VALUE TO A
 1260         JSR $FDDA    PRINT VALUE IN HEX
 1270         INC $24      LEAVE SPACE ON SCREEN
 1280         LDA #1       READ PADDLE 1
 1290         JSR READP
 1295         TYA          VALUE TO A
 1300         JSR $FDDA    PRINT VALUE IN HEX
 1310         JMP DEMO     AGAIN AND AGAIN...
```

```
================================================================================
DOCUMENT :AAL-8209:DOS3.3:S.RelocAmperMac.txt
================================================================================

 1000        .MA VECTOR
 1010        JSR $FF58
 1020  :1    TSX
 1030        LDY $100,X
 1040        DEX
 1050        LDA $100,X
 1060        CLC
 1070        ADC #:3-:1+1
 1080        BCC :2
 1090        INY
 1100  :2    .DO ']1='Y   CTRL-Y?
 1110        STA $3F9
 1120        STY $3FA
 1130        LDA #$4C
 1140        STA $3F8
 1150        .ELSE        OR &?
 1160        STA $3F6
 1170        STY $3F7
 1180        LDA #$4C
 1190        STA $3F5
 1200        .FIN
 1210        RTS
 1220  :3
 1230        .EM
```

```
==========================================================================
DOCUMENT :AAL-8209:DOS3.3:S.RelocAmpersnd.txt
==========================================================================

 1000  *--------------------------------
 1010  STACK        .EQ $100
 1020  AMPER.VECTOR .EQ $3F5
 1030  RETURN       .EQ $FF58
 1040  HOME   .EQ $FC58
 1050  *--------------------------------
 1060         .OR $300
 1070         .TF B.AMPEXAMPLE
 1080  *--------------------------------
 1090  SETUP  JSR RETURN        PUT CURRENT ADDR ON STACK
 1100  .1     TSX               GET STACK POINTER FOR OFFSET
 1110         LDY STACK,X       MSB OF ADDR ON STACK
 1120         DEX
 1130         LDA STACK,X       LSB
 1140         CLC
 1150         ADC #START-.1+1   OFFSET TO ENTRY POINT
 1160         BCC .2
 1170         INY               (Y) IS HI BYTE
 1180  .2     STA AMPER.VECTOR+1  LSB OF ENTRY ADDRESS
 1190         STY AMPER.VECTOR+2  MSB
 1200         LDA #$4C            JMP OPCODE
 1210         STA AMPER.VECTOR
 1220         RTS
 1230  *--------------------------------
 1240  START  JSR HOME     CLEAR SCREEN
 1250         NOP          DO WHATEVER
 1260         NOP          YOU LIKE
 1270         RTS
```

```
==============================================================================
DOCUMENT :AAL-8209:DOS3.3:S.Screen.Tricks.txt
==============================================================================

 1000  * S.SCREEN TRICKS
 1010  *--------------------------------
 1020  *   FAST SCREEN CLEAR SUBROUTINE
 1030  *--------------------------------
 1040  GCLEAR LDA #0
 1050         .HS 2C       SKIP OVER NEXT TWO BYTES
 1060  CLEAR  LDA #$A0
 1070  SET    LDY #119
 1080  .1     STA $400,Y   LINES:  0  8 16
 1090         STA $500,Y           2 10 18
 1100         STA $600,Y           4 12 20
 1110         STA $700,Y           6 14 22
 1120         STA $480,Y           1  9 17
 1130         STA $580,Y           3 11 19
 1140         STA $680,Y           5 13 21
 1150         STA $780,Y           7 15 23
 1160         DEY
 1170         BPL .1
 1180         RTS
 1190  *--------------------------------
 1200  *      SET SCREEN TO ALL VALUES
 1210  *--------------------------------
 1220  SETALL LDX #0
 1230  .1     TXA
 1240         JSR SET
 1250         INX
 1260         BNE .1
 1270         RTS
 1280  *--------------------------------
 1290  *   ALTERNATE SCREEN UNTIL KEY PRESSED
 1300  *--------------------------------
 1310  ALTER  LDA #$20     INVERSE BLANK
 1320         JSR SET
 1330         JSR CLEAR
 1340         LDA $C000
 1350         BPL ALTER
 1360         STA $C010
 1370         RTS
 1380  *--------------------------------
 1390  *   FAST SCROLL UP SUBROUTINE
 1400  *--------------------------------
 1410  SCROLL LDY #119
 1420  .1     LDA $400,Y   SAVE LINES: 0 8 16
 1430         PHA
 1440         LDA $480,Y   MOVE 1>0, 9>8, 17>16
 1450         STA $400,Y
 1460         LDA $500,Y   MOVE 2>1, 10>9, 18>17
 1470         STA $480,Y
 1480         LDA $580,Y   MOVE 3>2, 11>10, 19>18
```

```
       STA $500,Y
       LDA $600,Y   MOVE 4>3, 12>11, 20>19
       STA $580,Y
       LDA $680,Y        ET CETERA
       STA $600,Y
       LDA $700,Y
       STA $680,Y
       LDA $780,Y
       STA $700,Y
       PLA          MOVE 8>7, 16>15
       CPY #40
       BCC .2       DISCARD OLD LINE 0
       STA $780-40,Y
.2     DEY
       BPL .1
       RTS
*---------------------------------
*   SCROLL AROUND, MOVING TOP LINE TO BOTTOM
*---------------------------------
SCR    LDY #39      SAVE TOP LINE ON STACK
.1     LDA $400,Y
       PHA
       DEY
       BPL .1
       JSR SCROLL   SCROLL SCREEN UP ONE LINE
       LDY #0       STORE OLD TOP LINE
.2     PLA             ON BOTTOM OF SCREEN
       STA $7D0,Y
       INY
       CPY #40
       BCC .2
       RTS
*---------------------------------
*   ROTATE SCREEN UNTIL KEY PRESSED
*---------------------------------
S      JSR SCR      SCROLL AROUND ONCE
       LDA $C000    ANY KEY PRESSED?
       BPL S        NO, SCROLL AGAIN
       STA $C010    YES, CLEAR STROBE
       RTS          ...AND RETURN
```

================================================================
DOCUMENT :AAL-8209:DOS3.3:S.Tookit.Conv.txt
================================================================

```
 1000  "TOOLKIT CONVERTER
 1010  "
 1020  "
 1030  "COMMENTS
 1040  REP/ SKP 1/*/A
 1050  REP/ SKP 2/L/A
 1060  REP/ ;/       /A
 1070  REP/;/*/A
 1080  "DIRECTIVES
 1090  REP/ EQU /    .EQ /A
 1100  REP/ DW /     .DA /A
 1110  REP/ ORG /    .OR /A
 1120  REP/ DS /     .BS /A
 1130  REP/ DCI /    .AT /A
 1140  REP/ ASC /    .AS /A
 1150  REP/ DFB /    .DA #/A
 1160  REP/ CHN /*** CHN /A
 1170  "OPCODE TABS
 1180  REP/ ADC /    ADC /A
 1190  REP/ AND /    AND /A
 1200  REP/ ASL/     ASL/A
 1210  REP/ BIT /    BIT /A
 1220  REP/ CMP /    CMP /A
 1230  REP/ CPX /    CPX /A
 1240  REP/ CPY /    CPY /A
 1250  REP/ DEC /    DEC /A
 1260  REP/ EOR /    EOR /A
 1270  REP/ INC /    INC /A
 1280  REP/ LDA /    LDA /A
 1290  REP/ LDX /    LDX /A
 1300  REP/ LDY /    LDY /A
 1310  REP/ LSR/     LSR/A
 1320  REP/ ORA /    ORA /A
 1330  REP/ ROL/     ROL/A
 1340  REP/ ROR/     ROR/A
 1350  REP/ SBC /    SBC /A
 1360  REP/ STA /    STA /A
 1370  REP/ STX /    STX /A
 1380  REP/ STY /    STY /A
 1390  REP/ BPL /    BPL /A
 1400  REP/ BMI /    BMI /A
 1410  REP/ BEQ /    BEQ /A
 1420  REP/ BNE /    BNE /A
 1430  REP/ BVS /    BVS /A
 1440  REP/ BVC /    BVC /A
 1450  REP/ BCC /    BCC /A
 1460  REP/ BCS /    BCS /A
 1470  REP/ JMP /    JMP /A
 1480  REP/ JSR /    JSR /A
```

```
1490  REP/ BRK/     BRK/A
1500  REP/ CLC/     CLC/A
1510  REP/ CLD/     CLD/A
1520  REP/ CLV/     CLV/A
1530  REP/ DEX/     DEX/A
1540  REP/ DEY/     DEY/A
1550  REP/ INX/     INX/A
1560  REP/ INY/     INY/A
1570  REP/ NOP/     NOP/A
1580  REP/ PHA/     PHA/A
1590  REP/ PLA/     PLA/A
1600  REP/ PLP/     PLP/A
1610  REP/ RTS/     RTS/A
1620  REP/ SEC/     SEC/A
1630  REP/ TAX/     TAX/A
1640  REP/ TAY/     TAY/A
1650  REP/ TSX/     TSX/A
1660  REP/ TXA/     TXA/A
1670  REP/ TXS/     TXS/A
1680  REP/ TYA/     TYA/A
1690  "ADDRESS MODES
1700  REP" #>" #"A
1710  REP" #<" /"A
1720  "WRITE ON TEMP FILE
1730  TEXT#F
```

```
================================================================================
DOCUMENT :AAL-8209:DOS3.3:S.Usr.Week.Fn.txt
================================================================================

 1000  *SAVE S.USR WEEK FUNCTION
 1010  *--------------------------------
 1020  *      USR (X) = PEEK(X)+256*PEEK(X+1)
 1030  *--------------------------------
 1040         .OR $300     OR WHEREVER YOU WISH
 1050  *--------------------------------
 1060  USR    LDA $9D      CHECK RANGE
 1070         CMP #$91
 1080         BCS .1       ERROR
 1090         JSR $EBF2    CONVERT TO INTEGER IN $A0,A1
 1100         LDA $A0      PUT HIGH BYTE AFTER LOW BYTE
 1110         STA $A2
 1120         LDY #1
 1130         LDA ($A1),Y  HIGH-ORDER BYTE
 1140         STA $9E      HIGH BYTE OF MANTISSA
 1150         DEY
 1160         LDA ($A1),Y  LOW-ORDER BYTE
 1170         STA $9F      NEXT BYTE OF MANTISSA
 1180         SEC          SIGN IS POSITIVE
 1190         LDX #$90     EXPONENT 2^16
 1200         JMP $EBA0    FINISH CONVERSION
 1210  .1     JMP $E199    "ILLEGAL QUANTITY" MESSAGE
 1220  *--------------------------------
```

```
================================================================================
DOCUMENT :AAL-8209:DOS3.3:TEST.USR.txt
================================================================================

dπ11,0:π12,3(iÑX:¢,(37):∫X":  ";Fn∫'(X)" = ",(X)»256 ,(X»1)Ox´105
```

```
==================================================================
DOCUMENT :AAL-8209:DOS3.3:Toolkit.Conv.txt
==================================================================

"TOOLKIT CONVERTER
"
"
"COMMENTS
REP/ SKP 1/*/A
REP/ SKP 2/L/A
REP/ ;/   /A
REP/;/*/A
"DIRECTIVES
REP/ EQU /      .EQ /A
REP/ DW /       .DA /A
REP/ ORG /      .OR /A
REP/ DS /       .BS /A
REP/ DCI /      .AT /A
REP/ ASC /      .AS /A
REP/ DFB /      .DA #/A
REP/ CHN /*** CHN /A
"OPCODE TABS
REP/ ADC /      ADC /A
REP/ AND /      AND /A
REP/ ASL/       ASL/A
REP/ BIT /      BIT /A
REP/ CMP /      CMP /A
REP/ CPX /      CPX /A
REP/ CPY /      CPY /A
REP/ DEC /      DEC /A
REP/ EOR /      EOR /A
REP/ INC /      INC /A
REP/ LDA /      LDA /A
REP/ LDX /      LDX /A
REP/ LDY /      LDY /A
REP/ LSR/       LSR/A
REP/ ORA /      ORA /A
REP/ ROL/       ROL/A
REP/ ROR/       ROR/A
REP/ SBC /      SBC /A
REP/ STA /      STA /A
REP/ STX /      STX /A
REP/ STY /      STY /A
REP/ BPL /      BPL /A
REP/ BMI /      BMI /A
REP/ BEQ /      BEQ /A
REP/ BNE /      BNE /A
REP/ BVS /      BVS /A
REP/ BVC /      BVC /A
REP/ BCC /      BCC /A
REP/ BCS /      BCS /A
REP/ JMP /      JMP /A
REP/ JSR /      JSR /A
```

```
REP/ BRK/      BRK/A
REP/ CLC/      CLC/A
REP/ CLD/      CLD/A
REP/ CLV/      CLV/A
REP/ DEX/      DEX/A
REP/ DEY/      DEY/A
REP/ INX/      INX/A
REP/ INY/      INY/A
REP/ NOP/      NOP/A
REP/ PHA/      PHA/A
REP/ PLA/      PLA/A
REP/ PLP/      PLP/A
REP/ RTS/      RTS/A
REP/ SEC/      SEC/A
REP/ TAX/      TAX/A
REP/ TAY/      TAY/A
REP/ TSX/      TSX/A
REP/ TXA/      TXA/A
REP/ TXS/      TXS/A
REP/ TYA/      TYA/A
"ADDRESS MODES
REP" #>" #"A
REP" #<" /"A
"WRITE ON TEMP FILE
TEXT#F
```

==============================================================================
DOCUMENT :AAL-8210:Articles:Autocat.For.LC.txt
==============================================================================

Automatic CATALOG in the Language Card ...........Bill Morgan

It has been pointed out to me (loudly!) that I failed to mention the Language Card version of the Macro Assembler in my Automatic CATALOG routine last June.  Well, here's what you need to do:

Assemble the patch with an origin of $DF00.  This is a blank page inside the assembler.

BLOAD PATCH.

$D46D:FF DE  In the Language Card version, the Escape code jump table is at $D467-D482.

BSAVE S-C.ASM.MACRO.LC.MOD,A$D000,L$231F

That should take care of it!

==============================================================================
DOCUMENT :AAL-8210:Articles:CatalogArranger.txt
==============================================================================

Catalog Arranger ...................................Bill Morgan

We all have the problem: a disk starts getting full, we delete some files to make space, and our new files (from our latest project) end up scattered all through the catalog. A disk that has been used for a few months ends up with a thoroughly shuffled catalog.

There are programs available to alphabetize a catalog, but that's not always what I want to do. I want HELLO at the beginning, utilities next (assembler, text editor, ES-CAPE, disk zap, etc.), then various projects. The files for each project should all be grouped together, with the current job at the end of the catalog.

I decided that what I want to be able to do is to "pick up" one entry in the catalog, move it to exactly where I want it, put it down, then go get another one and put that one in its place, and so on. Here's my program to do just that.

Using Catalog Arranger

First BLOAD CATALOG ARRANGER, then insert the disk you want to modify. When you type CALL 2051 (or 803G from the monitor) the disk will spin for a little while as the catalog is read into a sort of string array. The first 22 entries in the catalog will then be displayed, with the first entry shown in inverse. You may notice that deleted files are also displayed, with a minus sign before the file type and a stray inverse character out at the end of the file name. Control characters are also displayed in inverse.

The inverted entry is a cursor showing the "active entry". If you press the arrow keys, this cursor will move up and down the display. When the cursor reaches the center of the screen, it will stop moving and the display will scroll up and down around it.

When you have the cursor on an item you wish to move, press RETURN. The word "MOVING" will appear in inverse in the lower left corner of the screen. When this "moving flag" is on, the entry in the cursor will be carried wherever the cursor goes. When it reaches the place where you want to put it, press RETURN again. The moving flag will disappear and that entry will stay where you just put it.

There are a couple of other commands as well. Pressing the "B" key moves the cursor to the beginning of the catalog, and the "E" key moves it to the end. If the moving flag is on, the item in the cursor will be carried right along. There is also an "R" command, to read in a new catalog. This is useful if you want to reread the current catalog and start all over again, or to move on to another disk.

When you have the catalog arranged just the way you want it, press the "W" key to write the revised catalog onto the disk. Press ESC when you want to exit the program.
Catalog Organization

If you are familiar with the internal structure of an Apple DOS catalog, you can skip ahead to the section labelled "How Catalog Arranger Works".

The first step in reading a disk catalog is to read the VTOC (Volume Table of Contents), which is always located at track $11, sector 0. The second and third bytes in the VTOC (offsets 1 and 2) contain the track and sector of the start of the catalog. On a standard DOS 3.3 disk these always point to track $11, sector $0F, and the rest of the catalog is always on track $11. These locations can be changed, however. For example, some programs to convert DOS 3.2 disks to DOS 3.3 leave the first catalog sector at track $11, sector $0C. Therefore, it is safest to follow the pointers rather than assuming that the catalog will always be in its usual place.

In a catalog sector, the first byte is not used. The second and third bytes point to the next track and sector of the catalog. If the track byte is zero, it means that there is no next sector, this is the end of the catalog. The fourth through the eleventh bytes are not used. The actual catalog information starts at the twelfth byte of the sector (offset $B) and fills the rest of the sector. Each catalog entry takes 35 bytes, so there are 7 entries in each sector.

The first two bytes of an entry contain the track and sector of the file's track/sector list. If the first byte is $FF, the file has been deleted. In that case, the track number has been moved to the end of the file name. If the first byte is zero, this entry has never been used, and we are past the end of the catalog.

The third byte tells the file type and whether the file is locked. Here are the type codes:

```
00 -- TEXT file
01 -- INTEGER BASIC program
02 -- APPLESOFT BASIC program
04 -- BINARY file
08 -- S file -- not used
10 -- R file -- DOS Toolkit relocatable object file
20 -- A file -- Lazer Pascal source file
40 -- B file -- Lisa assembler source file
```

If the file is locked, $80 is added to the type code.

The next 30 bytes are the file name. If the name is less than 30 characters long, it is filled out with spaces (ASCII $A0).

The last two bytes are the length of the file, in sectors. This is expressed as first low byte, then high byte.

Here's a diagram:

```
    0       1       2     3 . . . . 32      33         34
 --------------------------------- -------------------------
| Track | Sector | Type | Name . . . . | Len - Lo | Len - Hi |
 --------------------------------- -------------------------
```

You can find more information on the structure of the catalog in the DOS Manual on pages 129-134 or in the book Beneath Apple DOS on pages 4-4 through 4-7. It is impossible to recommend Beneath Apple DOS too highly; if you have any interest in the internals of DOS, get that book.

How Catalog Arranger Works

After initialization, the program builds a table of pointers into the storage area. We can build this table in advance because we know that all entries will be the same length. The catalog is then read into the array, each entry being placed according to the next pointer in the table. The end of the table is then marked with two zero bytes after the last element used.

The next step is to display the entries in the array. The display routine starts by checking ACTIVE.ELEMENT to see whether to start the display with the first element, or somewhere in the middle. It then scans up the table, displaying each catalog entry and inverting the one corresponding to ACTIVE.ELEMENT. The routine that actually displays each line borrows a couple of subroutines in DOS to decode the file type and display the file size.

When MOVING.FLAG is off, the arrow, B, and E commands simply change the value of ACTIVE.ELEMENT. When MOVING.FLAG is on, the arrows swap entries in the table up or down, and B and E repeatedly swap entries to move ACTIVE.ELEMENT to the beginning or end.

When writing the catalog back onto the disk we have to be careful to put the catalog sectors back in the same place they came from, since we can't assume that the catalog came from track $11. We do this by reading the first catalog sector into the buffer, scanning up the pointer table and moving the indicated entries into the catalog buffer, and then writing the buffer to the same disk sector it came from. We then get the track and sector pointers (which haven't been changed) from the buffer and use them to read the next sector. This whole process ends when we run out of entries in the pointer table.

Limitations and Additional Features

There are a few points that need more work:

Disk error handling. The program just prints "I/O ERROR" and stops. It needs a real error handler.

This program will handle catalogs with up to 127 entries. A standard disk has no more than 105, but some catalogs are modified to have more.

I plan to add several more commands to Catalog Arranger:

Alphabetic sort. This would be useful too, maybe just from the cursor to the end of the catalog. That would keep the utilities in place at the beginning.

Sort by file type.

Delete and undelete files.

Move deleted files to the end.

Rename files by editing the file name in the cursor. That would be a lot easier than typing entire file names twice, as RENAME requires.

Display and allow changing the values of SLOT and DRIVE.

Display the value of ACTIVE.ELEMENT and NUMBER.OF.ENTRIES, and maybe free sectors on the bottom line.

Write the catalog out to the disk as a text file.

Adding many of these features would also require reworking the command structure (which wouldn't hurt anyway!)

Here is a summary of the commands:

```
     B -- Move the cursor to the beginning of the catalog.
     E -- Move the cursor to the end of the catalog.
     R -- Read the catalog from the disk.
     W -- Write the catalog to the disk.
   --> -- Move the cursor down one item.
   <-- -- Move the cursor up one item.
RETURN -- Toggle the moving flag on or off.
   ESC -- Exit the program.
```

==============================================================================
DOCUMENT :AAL-8210:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 3 -- Issue 1                              October, 1982

In This Issue...



This issue of AAL is late.  No sooner do I warn you of one magazine that is behind in their publication schedule, than I get behind myself!  We plan to catch up with the next issue.

I just returned from 8 days in California.  Some of you know that I am on the board of directors of the International Apple Core.  After the board meeting I contacted a few long-time customers.  I also attended the San Francisco Apple Corps Swap Meet.

I looked up Peter Meyer, author of SDS's "Routine Machine"; together we had dinner at the home of Pat Caffrey, co-author of "Doubletime Printer".  Peter is now working on the fourth volume of additional Applesoft-extenders for his Routine Machine.  I also spent two half-days with Henry Spragens, well known for his early contributions to Apple graphics lore.  He bought his (first) Apple long ago in Kentucky, where he was one of the original members of LAUGHS (Louisville Apple User's Group for Hardware and Software!).  Now he works at Beck-Tech in Berkeley, doing exotic things in the world of synthesized video graphics with the Apple and other machines.

A few weeks earlier I spent the afternoon with DeWayne Van Hoozer in Houston, at the HAAUG meeting (you're right, it is pronounced "hog"!). DeWayne's Genasys project is nearing the publication stage, so you'll probably be hearing more about it soon.  I am looking forward to some more time in Houston around Halloween, at the AppleFest there.  Look me up if you are there, in or near the International Apple Core booth.

==========================================================================
DOCUMENT :AAL-8210:Articles:SC.LC.Patch.txt
==========================================================================

Another Lower Case Patch for S-C Macro.....Bob Sander-Cederlof

Graeme Scott pointed out another oversight of mine.  All lower case characters inside macro definitions are currently converted to upper case, whether or not you want it that way.  The following patches will fix it, assuming you have already installed the patches from AAL August 1982 page 28.

Motherboard version:  $275E:BA 31

Language Card version:  $E8AA:06 F3

I found another problem:  ".EM" and ".eM" work, but ".em" and ".Em" do not.  The following patches make them work too.

Motherboard version:

```
$31DB:B9 00 02 C9 60 90 02 29 5F 60
$2979:20 DB 31
```

Language Card version:

```
$F327:B9 00 02 C9 60 90 02 29 5F 60
$EAC5:20 27 F3
```

```
=============================================================================
DOCUMENT :AAL-8210:Articles:Scroll.Correx.txt
=============================================================================
```

Correction to Bob's Fast Screen Scroll............Jim Church

If you tried the fast scroll from Bob Sander-Cederlof's article "Some Fast Screen Tricks" from the September issue, you might have been surprised. Bob goofed!

He copied characters from line 16 into line 15 before moving line 15 to 14; ditto with lines 8, 7, and 6. This in spite of his special attempt to save lines on the stack. The problem is that he ran the loop backwards from 119 to 0. If you change it to run from 0 up to 119, the scroll works correctly.

Change lines 1410, 1620, and 1630, and add line 1625:

```
1410 SCROLL LDY #0

1620 .2     INY
1625        CMP #120
1630        BCC .1
```

Bob, you are going to get a lot of mail (unless they are asleep)!

```
=======================================================================
DOCUMENT :AAL-8210:Articles:SQ.Macro.txt
=======================================================================
```

So You Never Need Macros!..................Bob Sander-Cederlof

I have said it many times myself, "I don't need macros!"  But now that I have them, I seem to find more and more uses for them.  Not the traditional uses, to generate common sequences of opcodes.  I am using them to build tables of data, rather than typing in line after line of very similar stuff.

I have been working some more on the Prime Number Generator program. You may remember the series:  first the articles in BYTE Magazine, then my faster version in an early Apple Assembly Line, then Charles Putney's version at double my speed.  Now Tony Brightwell has cut Charlie's time nearly in half.  (His program will probably appear next month.)  Anyway, I have done some more investigation.

One approach required a precomputed table of the squares of the odd numbers from 1 to 127.  An easy way to enter this table might be:

```
.DA 1*1,3*3,5*5,7*7,9*9
.DA 11*11,13*13,15*15,17*17
et cetera
```

I had type about that much when I said, "There has to be an easier way."  I made up the following macro definition:

```
       .MA SQ
:0     .EQ ]1
:1     .EQ ]1+2
:2     .EQ ]1+4
:3     .EQ ]1+6
:4     .EQ ]1+8
:5     .EQ ]1+10
:6     .EQ ]1+12
:7     .EQ ]1+14
       .DA :0*:0,:1*:1,:2*:2,:3*:3
       .DA :4*:4,:5*:5,:6*:6,:7*:7
       .DO ]2<8
       >SQ ]1+16,]2+1
       .FIN
       .EM
```

Then the single line of code

```
     2200         >SQ 1,1
```

generated all 64 squares for me.

How does it work?  Good question....  The eight .EQ lines create 8 private labels with the values of 8 consecutive odd numbers starting with whatever the first parameter from the call line happens to be. Line 2200 has the first parameter "1", so the private labels will have values of 1, 3, 5, 7, 9, 11, 13, and 15 respectively. The two .DA lines generate the squares of these 8 values.

The next three lines are the tricky part. If the second parameter has a value less than 8 then the line between .DO and .FIN is assembled. It is a nested call on the SQ macro. Only this time the first parameter is 16 greater than it was, and the second parameter is one greater. After going through this nesting process 7 times, we will have generated 8 sets of 8 values each. When the second parameter has worked its way up to 8, the nested calls will exit in turn, and the table is finished.

If you have the macro expansion listing option on during assembly, the expanded form takes 2 1/2 pages

==============================================================================
DOCUMENT :AAL-8210:Articles:Toolkit.2.SC.txt
==============================================================================

Converting ToolKit Source to S-C...........Bob Sander-Cederlof

I had the source code for FIG-FORTH on the Apple, entered by some members of the Dallas Apple Corps. For some reason they decided to use the DOS ToolKit Assembler when they typed in all those lines. Naturally, I had a strong desire to convert the source files to the format of my S-C Macro Assembler.

The first step, and one of the easiest in this case, is to figure out how to read the ToolKit source files into S-C. ToolKit source files are standard DOS text files (type "T"). There are no line numbers. S-C allows such files to be read in by typing the following commands:

```
NEW
AUTO
<<<<<EXEC filename   (where "<" stands for backspace)
<<<<<MANUAL
```

"NEW" makes sure there are no lingering program lines from a previous load. "AUTO" starts generating automatic line numbers. The first line number generated will be 1000. Five backspaces will back up the cursor to the beginning of the input line, so the EXEC command can be typed. As the file is EXECing, each line will be read in with a prefixed line number. After the whole file has been read, five backspaces allow you to type the "MANUAL" command, thereby turning off the AUTO mode.

At this point you can LIST the program in memory and see what a ToolKit file looks like when you are using the S-C editor. You could use the EDIT and REPLACE commands to make all the necessary changes, and SAVE the converted program on a new file.

I was able to automate much of the conversion process, using an EXEC file of REPLACE commands. Several of you readers, including Graeme Scott of DFX fame, have sent me similar EXEC files for converting LISA source code.

Before I lay out the whole file, lets look at a simple case. The people who typed in the ToolKit source decided to separate individual sections of code with "SKP 1" lines. This causes a blank line on the assembly listing. S-C does not have an equivalent directive, but then again I personally don't like blank lines on my listings. (They always make me think my printer is broken!) Anyway, the command REP / SKP 1/*/A replaces all of the skips with empty comment lines. If you don't even want to see the asterisk on the line, use REP / SKP 1/ /A.

Notice that there is one space before "SKP" in the command above. ToolKit uses space as a tab character, and so the source file does not have nice neat columns for each field. If you list it with a regular text editor, the opcode field winds around like a snake; it always starting one space after the label, or in column 2 if there is no label. S-C uses control-I for a tab character, because control-I is the ASCII tab character. More on this later.

There were also a number of " SKP 2" lines. I decided to turn these into "*--------" lines, to indicated a greater separation than a mere empty comment line would.

ToolKit uses the semi-colon in column 1 to indicate a comment line; S-C uses an asterisk. ToolKit also uses a semi-colon to begin a comment field on a source line; S-C does not require any such character. The following two replace commands will make the necessary changes:

```
REP / ;/  /A
REP /;/*/A
```

The commands have to be in that order, or else you end up with an asterisk starting comment fields when they aren't necessary. Two lines had ";" in as ASCII literal constant. I had to hand-re-correct them later.

The most important changes are the directives. The files I was converting needed the following changes:

```
REP / EQU / .EQ /A
REP / DW / .DA /A
REP / ORG / .OR /A
REP / DS / .BS /A
REP / DCI / .AT /A
REP / DFB / .DA #/
REP / ASC / .AS /
```

Immediate address mode also presented a problem. ToolKit uses the form "LDA #<SSS" to indicate the high byte, and "LDA #>SSS" to indicate the low byte. S-C uses "LDA /SSS" for the high byte, and "LDA #SSS" for the low byte. I fixed them with:

```
REP " <#" /"A
REP " >#" #"A
```

Now about those snaky columns.... I wanted to somehow put a tab before each opcode field, and before each comment field. I thought, "Why not just use the replace command to put in a control-I?":

```
REP / EQU /^I.EQ /A    (where ^I means I typed control-I)
et cetera
```

My first problem was that typing controll-I when entering the REPLACE command made a tab. I overcame that by typing the sequence "control-O control-I". Control-O makes the next character become part of the input line regardless of its normal meaning. That worked, but....

My second problem was that getting a control-I into the source program did not make it a tab. Somehow the control-I had to be "executed". So I wrote the converted program on a text file, this time with line numbers, and then EXECed it back in.

```
TEXT# filename
EXEC filename
```

That "executed" the control-I's, and I had tabs. But....

My third problem was that I wanted to save all the REPLACE commands as an EXEC file, so that I did not have to manually retype them for every file to be converted. When I EXECed the REPLACE command file, the control-I's were executed immediately! I had to change my replace commands to include both a control-O and a control-I, so that the control-I in the REPLACE command would be read in from the EXEC file but not executed until it was later EXECed from the temporary source text file.

Still with me? If not, keep reading anyway, because I will show you what I mean.

Using S-C, I entered the following "program":

In the listing above, I have used "^O" to mean "control-O"; "^I" to mean "control-I"; and "^[" to mean "ESCAPE key". In order to get "control-O control-I" in a line, I had to type "control-O control-O control-O control-I".

Lines 1000-1030, 1080,1130, 1170,1690, and 1720 begin with a quotation mark. These are comment lines to the S-C input routine; they print on the screen when they are read from the EXEC file, but are otherwise ignored.

I saved the file as is using "SAVE TOOLKIT CONVERTER", in case I might want to modify it again. And I did, again and again and again. Then I wrote it on a text file without line numbers using "TEXT TKC".

Here is the sequence of steps I went through for each source file:

```
NEW
AUTO
1000 *    filename
<<<<<EXEC filename,D2   (where "<" means "backspace")
<<<<<MAN
EXEC TKC,D1
EXEC F
LIST 1000               (to see what filename to use)
SAVE filename
```

There are three EXEC commands above; the first reads on the ToolKit source file; the second executes all the REPLACE commands, and writes the resulting source on a temporary text file named "F"; the third reads in that temporary text file to "execute" the control-I tabs and the "ESC-L" lines.

After all the files were converted, I built a little assembly control file like this:

```
1000           .IN FILE1
1010           .IN FILE2
  et cetera
```

I also added a ".TF" directive after the ".OR" line, to put the assembled code on a DOS binary file.

The first assembly did not go smoothly, because of lines containing "ROR A". In the four shift instructions, ToolKit requires the symbol "A" to signify Accumulator mode. S-C uses a blank operand field to signify Accumulator mode, and thinks "ROR A" means to shift the memory location labeled "A".

Once I was able to assemble with no errors, I compared the object code produced with that produced by ToolKit. They did not match! There were two lines in the ToolKit source causing the problem:

```
         DW LIT,$FFFF
 L1495   DFB  $C1,$DB
```

The "DW" directive in ToolKit does not recognize multiple items separated by commas; therefore the ",$FFFF" was ignored. The following line in the source was "DW $FFFF". The S-C form ".DA LIT,$FFFF" does assemble both items, so the $FFFF constant was duplicated.

The "DFB" directive in ToolKit recognizes multiple items. The conversion I did rendered the line into "L1495  .DA #$C1,$DB", so the $DB item became a 16-bit value. I changed the line to "L1495  .DA #$C1,#$DB" and all was well.

If you have a large Toolkit source to convert, chances are that you will find one or two more things to change that are not included above. Let me know what you come up with.

As I mentioned earlier, the same general techniques work when you have a LISA file to convert. If you can get the source code on a text file, with all tokens expanded, then you can read it into S-C and begin converting. If you want a challenging assignment, how about writing a program which will read LISA type-B source files and convert them to S-C type-I source files, all automatically!

==============================================================================
DOCUMENT :AAL-8210:Articles:USR.Week.txt
==============================================================================

Using USR for a WEEK.......................Bob Sander-Cederlof

The "&" and CALL statements are not the only ways to use machine language to enhance the Applesoft Language. USR is a third way, and provides an easy way to return a single value.

How many times have you seen the Applesoft code "PEEK(X) + 256*PEEK(X+1)"? It is used over and over again. What it does is look in memory at X and X+1 for a 16-bit value (stored low-byte first as are most 16-bit values in the 6502 environment). The high byte is multiplied by 256, and the low byte added in. Wouldn't it be nice to have a USR function which would convert a two-byte value directly? This function is sometimes called "WEEK", meaning "Word pEEK" (hence the awful pun in the title above).

When I was in California last week someone categorically and unequivocally assured me that it is impossible to use the USR function with a value of 32768. I tried it with the WEEK function, and it works fine. So much for the assurances! I think his problem was that he followed the instructions in the Applesoft manual, which are somewhat incomplete.

Here is the USR code, set up to run at $300. However, it is "run-anywhere" code, because there are no internal references. You do have to tell Applesoft where it starts, though. Line 100 in the example shows how to do that. Location 11 and 12 must be set to the low- and high-bytes of the address of the USR code.

```
==============================================================================
DOCUMENT :AAL-8210:Articles:Writing.4.AAL.txt
==============================================================================
```

Writing for AAL............................Bob Sander-Cederlof

More and more of you are expressing interest in contributing articles to this newsletter.  Fine with me!

I accept them in almost any form.  It is by far the best if any source programs are on disk in S-C format, so I don't have to type them in. Other formats are OK, but more trouble.

I use my own word processor, which accepts standard DOS text files or Applewriter files.  If you have a large article, a copy on disk saves a lot of time here.

I receive more articles than I can use, but if yours is as good as you think it is, I will probably print it.  I usually spend a lot of time checking the programs and editing the articles before I print them.

Of course, I will return any disks you send.

==============================================================================
DOCUMENT :AAL-8211:Articles:Apple.Talker.txt
==============================================================================

A Sight of Sound.............Herbert A. & Herbert L. McKinstry

The Apple-Talker program that came on our disk for the S-C Assembler II Version 3.2 does some interesting things that go beyond what it was designed to do. When we tried it out we played a recorded message from our cassette recorder into the Apple memory and were amazed at the computer rendering of the original words.

The actual quality of reproduction leaves something to be desired, so when someone said "Let's see what it sounds like," and another said "Let's hear what it looks like," we snooped around the program listing and found that what we were hearing was stored on Hi-Res graphics page one. We looked at it by typing in $C050:0 and $C057:0. The sight of the sound was not too loud, nor was it even obvious that what we were looking at was the sound that we had heard.

If there were a pitch there, we should see some kind of pattern. We recorded a pitch, and saw that the sound was noisy. So then we entered some sense into memory by creating a repeating pattern, and listened to the patterns. We tried some like this:

```
*2000:FF FF 00 00 N 2004<2000.27FFM
*2800:FF 00 N 2802<2800.2FFFM
*3000:F0 N 3001<3000.37FFM
*3800:CC N 3801<3800.3FFEM
```

and

```
*2000:FC 0F C0 00 N 2004<2000.27FFM
*2800:F8 3F 03 E0 N 2804<2800.2FFFM
*3000:AA N 3001<3000.37FFM
*3800:CC N 3801<3800.3FFEM
```

We liked what we saw and we saw what we heard, so our thanks to Bob Sander-Cederlof and to Victor Borge for his recognition of the sight of sound.

Your Apple Can Talk.......................Bob Sander-Cederlof

Back in the summer of 1978, I spent two weeks in California with my kids. I visited a couple of computer stores with my brother, to show him what my Apple looked like. In one of them, I think the Byte Shop in Westminster on Beach Blvd., the proprietor mentioned in passing an astonishing event. He told me, "A high schooler was in here a few weeks ago with a program that produced speech out of the Apple speaker!" "Impossible," I mumbled.

A few weeks later I heard rumors of a program by Bob Bishop which did indeed make the Apple talk. I think it was in the September meeting of the Dallas Apple Corps that I overheard his program running. From amazement to insight took only a few seconds...I almost RAN home to write a program to do the same thing!

A month or two later I handed out copies of the program and gave a talk on the subject of speech synthesis and recognition. When Version 3.2 of the S-C Assembler was released, I included the same program as an example. Then again on version 4.0

Meanwhile, Bob Bishop released several neat tapes through Softape, including a talking calculator, "Apple Talker", and "Apple Listener". The latter program did some limited speech recognition through the cassette port, with no additional hardware. I bought the last two, and I still have the tape somewhere, but I have never loaded it.

About two years later Muse Software started marketing a program on disk to evoke speech from the Apple. I believe they included some sort of "editor" to allow you to make your own programs talk. I never saw or heard it, so I don't know.

As far as I know, the basic idea behind all of these programs is the same: approximate the waveform of spoken words by toggling the Apple speaker. You can hook a microphone up to the cassette input port and toggle the output speaker whenever the input port changes. Or you can record a message on tape, and "play" in into the Apple.

My program samples the cassette input port about 6000 times per second. If the input byte is $80 or larger, I store a "1"; if less than $80, I store a "0". I pack eight bits in a byte, and store the bytes in a buffer from $4000 through $5FFF. The buffer is 8192 bytes long, so that is 65536 samples or about 10 seconds of stored sound. You could store more samples or less samples, according to your own needs.

The playback loop looks at the stored bits at the same rate, and toggles the speaker whenever there is a change from 1 to 0 or 0 to 1. The result is actually understandable, though somewhat scratchy.

As the McKinstry's pointed out, my choice of buffer coincides with the Hi-Res Graphics page. In the copy of the program they have, I used $2000-3FFF, which is Hi-Res page one. Now I use $4000-5FFF, Hi-Res page two, so it will not erase the last half of the S-C Assembler when I test the program. Taking their suggestions to heart, I added the code to turn on the Hi-Res display during recording and playback, and to turn it off when finished.

When looking at the display you need to bear in mind the complex way the bytes are arranged on the Hi-Res screen. For starters, the bits are backwards in each byte. And remember that only seven bits of each byte show up on the screen -- the 8th bit shifts the other seven one half dot position. And the big confuser is the way the lines are arranged. (See Mike Laumer's article in the July ,1982 issue of AAL, page 15ff, for a discussion of the line arrangement.)

I prepared some EXEC files which initialize the buffer to various patterns, including the ones the two Herberts suggested. (It is amazing how handy it is to be able to create/modify little text files like these using the editor in the S-C Macro Assembler!)

Sound #1

```
$4000:FF FF 00 00 N 4004<4000.47FFM
$4800:FF 00 N 4802<4800.4FFFM
$5000:F0 N 5001<5000.57FFM
$5800:CC N 5801<5800.5FFEM
```

Sound #2

```
$5800:CC N 5801<5800.5FFEM
$5000:AA N 5001<5000.57FEM
$4800:F8 3F 03 E0 N 4804<4800.4FFCM
$4000:FC 0F C0 00 N 4004<4000.47FCM
```

Sound #3

```
$4000:00 01 03 07 0F 1F 3F 7F FF FE FC F8 F0 E0 C0 80
$4010<4000.5FEFM
```

Sound #4

```
$4000:00 FF 00 00 FF FF 00 00 00 00 FF FF FF FF
$400E<4000.5FFFM
```

Sound #5

```
$4000:00 88 00 00 88 88 00 00 00 00 88 88 88 88
$400E<4000.5FFFM
```

To play back one of the sound above, simple EXEC or type in the monitor commands, and then "MGO TALK".

Looking at the program which follows, you find three main routines. ECHO (lines 1180-1300) samples the cassette port about 6000 times per second; if it has changed, the speaker is toggled. After each toggle the keyboard strobe is examined, so that typing any key can stop the program and return to the caller.

RECORD (lines 1560-1710) stores 65536 samples in the buffer. TALK (lines 1750-1950) play back the buffer contents. You can play with the sample rate and playback rate by modifying the constant 30 in lines 1590 and 1790. It is amusing to play back a message faster or slower than it was recorded.

Both RECORD and TALK use a monitor subroutine called NXTA to control the loop. This is the same subroutine used by the monitor memory display and memory move commands. NXTA tests the current value of A1L,A1H ($3C,$3D) against A2L,A2H ($3E,$3F), and sets carry if A1 is greater than or equal to A2. Then it increments A1.

I tried various schemes for packing the bits in the buffer, to save space for more speech. None of them were effective enough to bother with, but you might run on to one that is. I also experimented with isolating words and individual phonemes, and with trying to filter out the scratchiness. I was not satisfied with any of my results. If you are successful, I would like to hear about it.

[ A later note: I just received Dec-82 Creative Computing, and there are reviews of several speech synthesis systems. One, called "Software Automatic Mouth (SAM)", is claimed to be a "high quality speech synthesizer created entirely in software." SAM costs $125 ($99 from Huntington Computing from now until the end of the year). In spite of the claim, it is not entirely software. There is also a small board containing a digital-to-analog converter (DAC), an audio amplifier, and a volume control. You can hook it up to the speaker in the Apple, or supply an external speaker. The ad claims it enables you to add speech to your programs with ease, but bear in mind that the software takes 9K of RAM, and 6K more if you want to automatically translate straight English text to speech. ]

==============================================================================
DOCUMENT :AAL-8211:Articles:Changing.Lomem.txt
==============================================================================

Moving the Symbol Table .............................Bill Morgan

Do you use the language card version of the S-C Macro Assembler? Have you ever tried to create more space for your object code by patching $D01D to move the symbol table up from $1000? Got a MEM PROTECT ERROR, didn't you? Here's what went wrong, and how to fix it.

The problem is the private label table for macros. This table is also protected during assembly, and starts at $FFF and grows downward. The base of the table is defined by a LDA #$10 instruction at $E564. When the table is searched during assembly, the check for the end of the table is a CMP #$10 at $E6A0. Both of these must also be patched to allow the $D01D patch to work. Here are the commands to correct the assembler:

```
:$C083 C083 N E564:A5 4B N E6A0:C5 4B N C080
:BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F
```

This changes the LDA #$10 to a LDA LOMEM+1 and the CMP #$10 to a CMP LOMEM+1. Now, whenever you want to move the symbol table, just type the following (where XX is the page you want the tables to start with):

```
:$C083 C083 D01D:XX N C080
:NEW
```

The NEW command is necessary to reset the page-zero pointers.

If you are using a target file and don't care about object code space, you can move the symbol table down. This creates more source code and symbol table space. You can move the table base all the way down to $800, if you are not using private labels. If you are using them, remember that each private label occurence uses 5 bytes of table space, so be sure to leave enough room under the table base.

Here's a map that shows how things got this way:

```
 -----------
|   Symbol  |
|   Table   |
 -----------  LOMEM
|           |          ----------
| Assembler |         |   Symbol |
|           |         |   Table  |
 -----------  $1000    ----------  LOMEM
|  Private  |         |  Private |
|  Labels   |         |  Labels  |
 -----------           ----------
```

```
   Normal               Language Card
   Version                 Version
```

The normal version of the assembler has to start at $1000, so the private label table also has to be there.  The language card version didn't get changed to reflect the fact that the private labels could now be moved.

```
==============================================================================
DOCUMENT :AAL-8211:Articles:Exec.WO.End.txt
==============================================================================
```

EXEC without END from Applesoft...........Bob Sander-Cederlof

I have been working on a project with Lee Meador which requires a binary file to be loaded into the second $D000 bank of a 16K RAM card. It is just a little tricky to do this!

You cannot just use a simple BLOAD, because you have to be sure the RAM card is selected and write-enabled. You cannot do it from a running Applesoft program, or even as a direct command after the Applesoft prompt, because if the RAM card is enabled the Applesoft ROMs are not. We wanted to do it from within the running Applesoft program.

The typical answer is to create an EXEC file with the commands to call the monitor, select the RAM card, BLOAD the file, reselect the motherboard ROMs, and bounce back to Applesoft. For example:

```
CALL-151               call Apple monitor
C089 C089              write-enable RAM with 2nd bank
F800<F800.FFFFM        copy of monitor in RAM card
BLOAD B.BOBANDLEE      load the file
C081                   back to Applesoft ROMs
3D0G                   back to Applesoft, softly
```

You can nicely EXEC this file from the direct mode, or from a running Applesoft program. However, in order to use it from a running program, the program must END or STOP. Do it like this:

```
100 PRINT CHR$(4)"EXEC LOAD 2ND BANK":END
```

If you don't END the program, the EXEC file will probably just become part of the input to your Applesoft program, rather than being executed.

HOWEVER.... You can beat the system. Change the EXEC file to this form:

```
C089 C089
F800<F800.FFFFM
BLOAD B.BOBANDLEE
C081
D7D2G
```

And the Applesoft code to this:

```
100 PRINT CHR$(4)"EXEC LOAD 2ND BANK":CALL-151
```

Note the two changes in the EXEC file: the CALL-151 is not there, and 3D0G has become D7D2G. And in the Applesoft code instead of END we have CALL-151.

The CALL-151 starts up the Apple monitor, which reads the commands from the EXEC file. The last command jumps to $D7D2, the running entry into Applesoft. This continues execution of the Applesoft program from the next statement after the CALL-151.

```
==============================================================================
DOCUMENT :AAL-8211:Articles:Front.Page.txt
==============================================================================
```

$1.50

Volume 3 -- Issue 2                                   November, 1982

In This Issue...



Apple/Fest in Houston

Although only about one-third the size of the Boston original (last May), it was still worth the trip.  I met an orthopedic surgeon from Lille, France, who flew down Saturday from New York just for the show. Also a professor from Des Moines.  There was not a lot to see that could be called NEW, but it was valuable to meet and get to know the people.  I brought along my son David, almost ten now; he loved playing all the new games, and was a big help in the IAC booth.

If you are in an area where there is no club of Apple owners, you might like to contact the International Apple Core at 908 George St, Santa Clara, CA 95050.  They have a start-up kit for new clubs that will help you organize your own club.

Bill Morgan also came on Saturday.  We have kidded Bill in the past that he looked a lot like Paul Lutus...well, three people were almost positive on Saturday!

Another Christmas Special

And this one is in December!  Subscribers have until the end of 1982 to get Laumer Research's FLASH Integer Basic Compiler at only $49. That is a savings of nearly 38%!

Advertising in AAL

Once again, the price per page of advertising in Apple Assembly Line is going up.  The December issue will run $90 for a full page, $50 for a half page.

```
==============================================================================
DOCUMENT :AAL-8211:Articles:Locator.txt
==============================================================================
```

Applesoft Program Locator............................Bill Morgan

Have you ever wanted to know exactly where your Applesoft program and variables are in memory? How much space is code and how much is variables? How close you're getting to the Hi-Res display space? FRE(0) will tell you how much space you have, but not where it is. You can PEEK the Applesoft pointers, or go into the monitor to check them, but that means you have to remember where all the pointers are.

In the October, 1982 issue of Big Apple Users Digest I saw a program by Frank Weinberg to build an EXEC file called FPSTAT, which displays the Applesoft pointers to program and variable locations. (That program was credited as being reprinted from The Grapevine, August, 1982.) Now that was pretty neat, but EXEC is so slow, and the adresses were printed in decimal. I'm more comfortable thinking of addresses in hex notation. Bob suggested writing a BRUNnable program which would execute in page 2 (the input buffer), thus avoiding conflict with any page 3 routines that might be present. Here's what I came up with.

Using LOCATOR

Whenever you want to know the memory situation, just BRUN LOCATOR. It will display something like this:

```
PROGRAM: $0801 TO $0923
 SIMPLE: $0923 TO $0A35
 ARRAYS: $0A35 TO $1B3C
STRINGS: $9435 TO $9600
```

PROGRAM shows the location of the actual text of your program. SIMPLE is the simple variables, both numeric and string pointers. ARRAYS is the array variables, both numeric and string. STRINGS is the area used by the actual text of the strings.

Notice that the upper addresses are all one too large. Applesoft's end-of-program and end-of-variables pointers actually point to the next available location, rather than the last location used. Similarly, the end-of-strings pointer is HIMEM, which is one past the last location available. I wrote another version of LOCATOR which automatically decremented the second address in each line, but that got cumbersome, and returned silly values if the Applesoft program had not yet been RUN. (For example, SIMPLE: $0923 TO $0922.)

If you want to CALL LOCATOR from within an Applesoft program, change line 1320 from JMP $3D0 to RTS, and change the origin to $294. Then you can CALL 660, if you're not using very long input lines. Or, you can put LOCATOR in page 3, if you're not already using that area.

It is also interesting to RUN a program, BRUN LOCATOR, then type FRE(0) and call LOCATOR again. This lets you see just how much wasted string space you have had, and gives you some idea how long the garbage collector takes to clear how much space.

I'm looking forward to using LOCATOR together with EXAMINER (from AAL June, 1982) to study Applesoft's variable structure. You can find more information on Applesoft variables and their pointers on pages 126-127 and 137 of the Applesoft manual.

How LOCATOR Works

Since we are printing eight addresses, the X-register is used to count from 0-7. In lines 1140-1190 that count is converted into a value of $0, $8, $10, or $18, to determine which title line to print. If the titles hadn't been a convenient 8 bytes long, we could have inserted a title offset at the beginning of each of the .DA statements in lines 1570-1600, and loaded Y from there.

The heart of the program is the table of Applesoft pointers at lines 1570-1600. In lines 1420-1440 the Y-register is loaded with a value from the table, then used to load the A- and X-registers with the address pointed to. The program then calls MON.PRNTAX, which displays first the A- and then the X-register.

====================================================================
DOCUMENT :AAL-8211:Articles:More.Speech.txt
====================================================================

Speaking of Speech.................................Bill Morgan

Just thought I'd tell you a little about the way I played around with a speech program like Bob's.  I couldn't find the disk with the exact code, but here's what I remember about it.  I wanted a brief Applesoft program which would say the numbers 0 through 9 when a number key was pressed.

To do this, first record your voice on tape, reciting the ten numbers. Then play the tape into your Apple, using the RECORD routine in Bob's program.  Now, by using the system monitor to examine memory, it's easy to scan through the buffer and see where each word begins and ends.  The gaps between words will be long stretches of "00 ... 00", with a few stray bytes of noise along the way.  Words will be stretches of random-looking values.  It's interesting to see the difference between a word like "two", which starts abruptly and trails off, and one like "eight", which starts more slowly and ends suddenly.

Now you can use the monitor move command to remove the gaps between words.  Move the data for "one" to the very beginning of the buffer, and note its start and end addresses.  Then move "two" down to the space just after "one", and note the addresses.  Carrying on like this, you can compress the number data into about half the space of the original recording.

Assemble the playback portion of Bob's program at $300.  All you should need is lines 1760-1950 (plus the needed .EQ's), with an RTS substituted for the JMP FINISH at line 1920.  To say a number, all your Applesoft program has to do is get the starting and ending addresses of a word from an array, POKE the addresses into locations 60-63, and CALL 768.

```
=================================================================================
DOCUMENT :AAL-8211:Articles:My.Ad.txt
=================================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
S-C Macro Cross Assembler Modules
    6800/6801/6802 Version........................................$32.50
    6809 Version..................................................$32.50
    Z-80 Version..................................................$32.50
    68000 Version.................................................$50.00
    Requires ownership of S-C Macro Assembler.
    Each disk includes regular and language card versions.
Upgrade from Version 4.0 to MACRO.................................$27.50
Source code of Version 4.0 on disk................................$95.00
    Fully commented, easy to understand and modify to your own tastes.

ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks.......................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)......(regular $79) $49.00
    Special price to AAL readers only, until 12/31/82!
Source Code for FLASH! Runtime Package............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware)......................$45.00
Apple White Line Trace (Lone Star Industrial Computing)...........$50.00
    (A unique learning tool)

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $4.50
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
    (Messy, but effective!)
Diskette Mailing Protectors........................10-99:  40 cents each
                                              100 or more:  25 cents each
    Corrugated folder specially designed for mailing mini-floppy
    diskettes.  Fits in standard 6x9-inch envelope.  (Envelopes
    5-cents each, if you need them.)

Ashby Shift-Key Mod...............................................$15.00
Paymar Lower-Case Adapter.........................................$37.50
    For Apples before Revision 7 only
Lower-Case Display Encoder ROM....................................$25.00
    Works only Revision level 7 Apples.  Replaces the encoder ROM.

Books, Books, Books.........................compare our discount prices!
```

```
"Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
"Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
"Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
"Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
"What's Where in the Apple", Second Edition..........($24.95)  $23.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

=============================================================================
DOCUMENT :AAL-8211:Articles:Repeat.Until.txt
=============================================================================

REPEAT and UNTIL for Applesoft.....................Bobby Deen

The following program adds three statements to Applesoft: &REPEAT, &UNTIL, and &POPR. With these you can write Pascal-like loops in your Basic programs.

You start the loop with &REPEAT, and end it with &UNTIL <exp>. The loop will be repeated until the <exp> evaluates to non-zero (true). As long as the value of <exp> is zero (false), the loop will keep going.

I use the system stack for saving the line number and the program pointer, just like Applesoft does with FOR-NEXT loops. A special code is used to identify the stuff on the stack, so you can have FOR-NEXT loops inside REPEAT-UNTIL loops and vice versa.

The statement &POPR removes one REPEAT block from the stack, in case you want to jump out of a loop rather than completing it. (This is not generally a good practice, even with FOR-NEXT loops, but you can do it if you feel you must.) The statement "&UNTIL 1" will do the same thing as &POPR, but &POPR takes less space and time.

If &POPR or &UNTIL is executed when there is not an UNTIL block on the top of the stack, you will get "NEXT WITHOUT FOR" error.

Applesoft parses the word "REPEAT" as four letters "REPE" and the token "AT". This makes the listings look weird, but never mind. Likewise, "UNTIL" looks like a variable name during tokenization, so the expression runs into the letter "L"; but at execution time all is understood.

Here is a sample program which shows a pair of REPEAT loops:

```
100  REM TEST REPEAT/UNTIL
110 D$ =  CHR$ (4): PRINT D$"BLOAD B.REPEAT/UNTIL": CALL 768
120 I = 0: & REPE AT
130 I = I + 1: PRINT I":  ";
135 J = 0: & REPE AT :J = J + 1: PRINT J" ";: & UNTILJ > 14:
    PRINT
140  & UNTILI = 10
```

Lines 1200-1250 install the ampersand vector. I assumed the JMP opcode is already stored at $3F5, since DOS does that. After BLOADing the file, CALL 768 will executed these lines.

When the "&" is executed, the 6502 jumps to AMPER.PARSE at line 1270. Lines 1270-1420 search through a table of keywords, matching one if possible with the characters after the "&" in your Applesoft program.

This is a general routine, which you can use for any keywords, just by making the appropriate entries in the table (lines 1500-1590).

The table contains a string and an address for each keyword. The string is shown as a hex string, and includes the exact hexadecimal values expected. For example, for "REPEAT" I have entered the ASCII codes for "REPE" and the token value or "AT". After the keyword there is a 00 byte to flag the end, and a two byte address. The address will be pushed onto the stack so that an RTS instruction will branch to the processing program for that keyword. Since RTS adds one, the address in the table have "-1" after them.

The last entry in the table has a null keyword, so it will match anything and everything. If the search goes this far, we have a syntax error; therefore the branch address is to the Applesoft syntax error code.

When a keyword is matched, the Y-register contents need to be added to TXTPTR. A subroutine in the Applesoft ROMs does this, called AS.ADDON. Since both REPEAT and POPR require the next character to be end-of-line or a colon, a JMP to AS.CHRGOT gets the next character and tests it. The RTS at the end of AS.CHRGOT actually branches to the processing code for the keyword.

Lines 1600-1840 process the REPEAT command. A five-byte block is pushed onto the stack, consisting of the current line number, the TXTPTR, and a code value $B8.

Lines 1850-2070 process the UNTIL command. First the expression is evaluated. If the value turns out to be zero, the byte at FAC.EXP will be zero. If it is zero, we need to keep looping; if non-zero, the loop is finished. Looping involves copying the line number and text pointer from the stack back into CURLIN and TXTPTR, and then going to AS.NEWSTT. The REPEAT block is left on the stack, and execution resumes just after the &REPEAT that started this loop.

If the expression is true (non-zero), the loop is terminated. Termination is trivial: just pop off the REPEAT block, and go to AS.NEWSTT to continue execution after the UNTIL statement. I could pop the block off with seven PLA's, but I used the technique of adding 7 to the stack pointer instead.

Naturally, this package was assembled to sit in page 3, along with 99 other machine language things you use. You can easily move it to another location, just by changing the origin (line 1180). Or you can use the routines with Amper-Magic or the Routine Machine. Note that the routines themselves are relocatable run-anywhere code (no data references, JSR's, or JMP's to points within the routines). You will have to shorten the routine names to four or less characters to use them with Amper-Magic.

Pascal has some other looping constructs which you might like to see in Applesoft. Now that you see how I did this one, why not try your hand at coding REPEAT WHILE?

```
=======================================================================
DOCUMENT :AAL-8211:Articles:TonyFasterPrime.txt
=======================================================================
```

Even Faster Primes..........................Anthony Brightwell

Is this the last word on prime number generation?

I modified Charles Putney's program from the February issue, and cut the time from 330 milliseconds down to 183 milliseconds!  Here is what I did:

*  I sped up the zero-memory loop by putting more STA's within the loop.

*  I removed the CLC from the main loop.  After all, why CLC withing the loop if you're looping on a BCC condition?

*  I removed the LDA #$FF from the main loop.  It was there to be sure a non-zero value gets stored in non-prime slots, but why LDA #$FF if the accumulator never contains $00 within the loop?

*  I changed the way squares of primes are computed.  Charlie did it using a quick 8-bit by 8-bit multiply.  I took advantage of a little number theory, and shaved off some time.

The method I use for squaring may appear very round-about, but it actually is faster in this case.  Look at the following table:

| Odd #'s | square | neat formula |
|---|---|---|
| 1 | 1 | 0 * 8 + 1 |
| 3 | 9 | 1 * 8 + 1 |
| 5 | 25 | 3 * 8 + 1 |
| 7 | 49 | 6 * 8 + 1 |
| 9 | 81 | 10 * 8 + 1 |

The high byte of the changing factor in the "neat formula" is stored in the LDA instruction at line 1550, and the low byte in the ADC instruction at line 1900.  The factor is the sum of the numbers from 1 to n:  1+2=3, 1+2+3=6, 1+2+3+4=10, etc.  In all, 31 primes are squared, and the total time for all the squaring is less than 3 milliseconds.

Here is a driver in Applesoft to load the program and then print out primes from the data array.

```
10  REM DRIVER FOR TONY'S FAST PRIME FINDER
20  PRINT CHR$ (4)"BLOAD B.TONY'S SUPER-FAST PRIMES"
30  HOME : PRINT "HIT ANY KEY TO START"
40  GET A$: PRINT " GENERATING PRIMES . . ."
50  CALL 32768
60  FOR A = 8195 TO 24576 STEP 2
70  IF  PEEK (A) = 0 THEN  PRINT A - 8192;" ";
```

```
80  NEXT
```

A few more cycles can probably still be shaved....  Any takers?

```
==============================================================================
DOCUMENT :AAL-8211:DOS3.3:S.LOCATOR.txt
==============================================================================
```

```
*SAVE S.LOCATOR
*--------------------------------
       .OR $292     HIGH END OF INPUT BUFFER
*      .TF LOCATOR
*--------------------------------
ZERO       .EQ 0
MON.PRNTAX .EQ $F941
MON.COUT   .EQ $FDED
MON.CROUT  .EQ $FD8E
*--------------------------------
START  LDX #0
LOOP   JSR MON.CROUT     NEW LINE
       TXA
       LSR               MAKE (X)
       ASL               INTO
       ASL               TITLE
       ASL               INDEX
       TAY
.1     LDA TITLES,Y      SHOW TITLE
       JSR MON.COUT
       INY
       CMP #':+$80       ":" ?
       BNE .1
       LDY #1            FILL WITH " $"
       JSR PRINT.ADDRESS
       LDY #4            FILL WITH " TO $"
       JSR PRINT.ADDRESS
       CPX #7            DONE YET?
       BCC LOOP          NO, GO ON
       JMP $3D0          YES, EXIT TO DOS
*--------------------------------
PRINT.ADDRESS
.1     LDA FILLER,Y      Y TELLS HOW
       JSR MON.COUT      MUCH FILLER
       DEY               TO PRINT
       BPL .1
       TXA
       PHA               SAVE X
       LDY TABLE,X       GET POINTER
       LDA ZERO+1,Y      GET HIGH BYTE
       LDX ZERO,Y        GET LOW BYTE
       JSR MON.PRNTAX    DISPLAY ADDRESS
       PLA
       TAX               RESTORE X
       INX               AND GET READY
```

```
1490        RTS                FOR NEXT PASS
1500  *---------------------------------
1510  TITLES
1520        .AS -/PROGRAM:/
1530        .AS -/ SIMPLE:/
1540        .AS -/ ARRAYS:/
1550        .AS -/STRINGS:/
1560  *---------------------------------
1570  TABLE  .DA #$67,#$AF     START OF PROGRAM, END OF PROGRAM
1580         .DA #$69,#$6B     START OF VARIABLES, START OF ARRAYS
1590         .DA #$6B,#$6D     START OF ARRAYS, END OF NUMERICS
1600         .DA #$6F,#$73     START OF STRINGS, HIMEM
1610  *---------------------------------
1620  FILLER .AS -/$ OT /
```

```
==============================================================================
DOCUMENT :AAL-8211:DOS3.3:S.NewAplTalker.txt
==============================================================================

 1000  *--------------------------------
 1010  *       APPLE-TALKER FROM S-C SOFTWARE CORP.
 1020  *--------------------------------
 1030  MON.NXTA .EQ $FCBA     BUMP AND TEST A1
 1040  *--------------------------------
 1050  CASSETTE .EQ $C060     CASSETTE INPUT LEVEL
 1060  SPEAKER  .EQ $C030     SPEAKER OUTPUT
 1070  STROBE   .EQ $C010
 1080  KEYBOARD .EQ $C000
 1090  *--------------------------------
 1100  LAST     .EQ $2F       LAST CASSETTE INPUT LEVEL
 1110  A1L      .EQ $3C       MONITOR A1L, A1H, A2L, A2H
 1120  *--------------------------------
 1130  BUFFER   .DA $4000     FWA OF BUFFER
 1140           .DA $5FFF     LWA OF BUFFER
 1150  *--------------------------------
 1160  *    ECHO CASSETTE THRU SPEAKER
 1170  *--------------------------------
 1180  ECHO   LDY #30       150 USEC DELAY
 1190  .1     DEY
 1200         BNE .1
 1210         LDA CASSETTE
 1220         EOR LAST      SEE IF TOGGLED
 1230         BPL ECHO      NO
 1240         EOR LAST      YES
 1250         STA LAST
 1260         LDA SPEAKER   TOGGLE SPEAKER
 1270         LDA KEYBOARD
 1280         BPL ECHO
 1290         STA STROBE
 1300         RTS
 1310  *--------------------------------
 1320  *    SET UP BUFFER ADDRESSES
 1330  *--------------------------------
 1340  SETUP  LDX #3
 1350  .1     LDA BUFFER,X
 1360         STA A1L,X
 1370         DEX
 1380         BPL .1
 1390         STX LAST
 1400         LDA $C050    SELECT HGR2 FOR VIEWING
 1410         LDA $C052
 1420         LDA $C055
 1430         LDA $C057
 1440         RTS
 1450  *--------------------------------
 1460  *    RESTORE NORMAL SCREEN AND EXIT
 1470  *--------------------------------
 1480  FINISH LDA $C051
```

```
1490        LDA $C053
1500        LDA $C054
1510        LDA $C056
1520        RTS
1530 *---------------------------------
1540 *    STORE SPEECH IN BUFFER
1550 *---------------------------------
1560 RECORD JSR SETUP     SET UP BUFFER ADDRESSES
1570 .1     LDX #8        EIGHT BITS
1580 .2     PHA           PUSH BYTE WE ARE FILLING
1590        LDY #30
1600 .3     DEY           150 USEC DELAY
1610        BNE .3
1620        LDA CASSETTE  READ CASSETTE LEVEL
1630        ASL           LEVEL INTO CARRY BIT
1640        PLA
1650        ROL           MERGE LEVEL INTO BYTE
1660        DEX
1670        BNE .2        BYTE NOT FULL YET
1680        STA (A1L,X)   STORE NEXT WORD IN BUFFER
1690        JSR MON.NXTA      BUMP & TEST POINTER
1700        BCC .1        NOT THRU
1710        JMP FINISH
1720 *---------------------------------
1730 *    PLAYBACK SPEECH FROM BUFFER
1740 *---------------------------------
1750 TALK   JSR SETUP     SET UP BUFFER ADDRESSES
1760 .1     LDX #0
1770        LDA (A1L,X)   GET NEXT WORD FROM BUFFER
1780        LDX #8        EIGHT BITS
1790 .2     LDY #30
1800 .3     DEY           150 USEC DELAY
1810        BNE .3
1820        EOR LAST      TEST IF LEVEL CHANGED
1830        BPL .5        NO
1840        EOR LAST      YES, RESTORE (A)
1850        STA LAST      UPDATE LEVEL
1860        LDY SPEAKER   TOGGLE SPEAKER
1870 .4     ASL
1880        DEX
1890        BNE .2
1900        JSR MON.NXTA      BUMP & TEST POINTER
1910        BCC .1        NOT THRU
1920        JMP FINISH
1930 .5     EOR LAST      RESTORE (A)
1940        JMP .6        EVEN OUT TIMING
1950 .6     JMP .4
```

==============================================================
DOCUMENT :AAL-8211:DOS3.3:S.Repeat.Until.txt
==============================================================

```
1000  *SAVE S.REPEAT/UNTIL
1010  *--------------------------------
1020  *   BY BOBBY DEEN
1030  *      629 WINCHESTER DR
1040  *      RICHARDSON,TX. 75080
1050  *      (214) 235-4391
1060  *--------------------------------
1070  AMPERSAND.VECTOR .EQ $3F5
1080  AS.FRMEVL  .EQ $DD7B     EVALUATE A FORMULA
1090  AS.CHRGOT  .EQ $00B7     GET CHAR AT TXTPTR
1100  AS.TXTPTR  .EQ $00B8     POINT TO PROGRAM TEXT
1110  AS.SYNERR  .EQ $DEC9     SYNTAX ERROR
1120  AS.ADDON   .EQ $D998     ADDS (Y) TO TXTPTR
1130  AS.CURLIN  .EQ $75       CURRENT LINE NUMBER
1140  FAC.EXP    .EQ $9D       EXPONENT OF FAC
1150  AS.BADFOR  .EQ $DD0B     NEXT WITHOUT FOR ERROR
1160  AS.NEWSTT  .EQ $D7D2     EXECUTE NEW STATEMENT
1170  *--------------------------------
1180         .OR $300
1190         .TF B.REPEAT/UNTIL
1200  *--------------------------------
1210  START  LDA #AMPER.PARSE
1220         STA AMPERSAND.VECTOR+1
1230         LDA /AMPER.PARSE
1240         STA AMPERSAND.VECTOR+2
1250         RTS
1260  *--------------------------------
1270  AMPER.PARSE
1280         LDX #-1      START OF TABLE
1290  .1     LDY #-1      START OF AMPER-CALL
1300  .2     INX
1310         INY
1320         LDA TABLE,X  NEXT CHAR FROM TABLE
1330         BEQ .4       END OF KEYWORD, MATCHED
1340         CMP (AS.TXTPTR),Y   COMPARE WITH AMPER-CALL
1350         BEQ .2       MATCHES SO FAR
1360  *---SKIP TO NEXT TABLE ENTRY-----
1370  .3     INX          ...TO END OF KEYWORD
1380         LDA TABLE,X
1390         BNE .3
1400         INX          ...OVER THE ADDRESS
1410         INX
1420         BNE .1       ...ALWAYS
1430  *---MATCHED A KEYWORD-----------
1440  .4     JSR AS.ADDON      ADJUST TXTPTR PAST KEYWORD
1450         LDA TABLE+2,X     GET ADDRESS AND BRANCH
1460         PHA
1470         LDA TABLE+1,X
1480         PHA
```

```
1490        JMP AS.CHRGOT     GET CHAR AT TXTPTR
1500 *--------------------------------
1510 TABLE
1520        .HS 52455045C500    "REPEAT"
1530        .DA REPEAT-1
1540        .HS 554E54494C00    "UNTIL"
1550        .DA UNTIL-1
1560        .HS A15200          "POPR"
1570        .DA POPR-1
1580        .HS 00              ANYTHING
1590        .DA AS.SYNERR-1
1600 *--------------------------------
1610 * REPEAT COMMAND
1620 *--------------------------------
1630 REPEAT
1640        BNE SYNERR       NOT THERE
1650        PLA          SAVE RETURN ADDRESS
1660        TAX
1670        PLA
1680        TAY
1690        LDA AS.CURLIN+1 PUSH CURRENT LINE NUMBER
1700        PHA
1710        LDA AS.CURLIN
1720        PHA
1730        LDA AS.TXTPTR+1 PUSH TEXT POINTER
1740        PHA
1750        LDA AS.TXTPTR
1760        PHA
1770        LDA #$B8 IDENTIFIER FOR REPEAT LOOP
1780        PHA      SO THIS ISN'T MISTAKEN FOR FOR/NEXT
1790 *               OR GOSUB/RETURN
1800        TYA       PUT RETURN ADDRESS ON STACK
1810        PHA
1820        TXA
1830        PHA
1840        RTS       AND GO BACK
1850 *--------------------------------
1860 * PROCESS UNTIL COMMAND
1870 *--------------------------------
1880 UNTIL
1890        JSR AS.FRMEVL GET EXPRESSION
1900        LDA FAC.EXP  GET EXPONENT
1910        BNE POP.IT   TRUE,END LOOP
1920        TSX          KEEP LOOPING
1930        LDA $103,X
1940        CMP #$B8     IS IT A REPEAT?
1950        BNE BADFOR   NO,ERROR
1960        LDA $104,X   GET THE DATA
1970        STA AS.TXTPTR AND TELL APPLESOFT
1980        LDA $105,X
1990        STA AS.TXTPTR+1
2000        LDA $106,X
2010        STA AS.CURLIN
2020        LDA $107,X
```

```
2030        STA AS.CURLIN+1
2040        INX          WE DON'T NEED THE RETURN ADDRESS
2050        INX
2060        TXS          KILL SUB CALL
2070        JMP AS.NEWSTT  NEW STATEMENT
2080 *-------------------------------
2090 * POP A REPEAT LOOP OFF STACK
2100 *-------------------------------
2110 POPR
2120        BNE SYNERR
2130 POP.IT TSX          EXP TRUE,SO END LOOP
2140        LDA $103,X   MAKE SURE IT IS A REPEAT
2150        CMP #$B8
2160        BNE BADFOR
2170        TXA
2180        CLC
2190        ADC #7       PULL 7 THINGS
2200        TAX
2210        TXS
2220        JMP AS.NEWSTT
2230 *-------------------------------
2240 BADFOR JMP AS.BADFOR
2250 SYNERR JMP AS.SYNERR
2260 *-------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8211:DOS3.3:S.TonyFasterPrm.txt
==============================================================================

 1000  *SAVE S.TONY'S SUPER-FAST PRIMES
 1010         .OR $8000    SAFELY OUT OF WAY
 1020         .TF B.TONY'S SUPER-FAST PRIMES
 1030  *---------------------------------
 1040  BASE   .EQ $2000    BASE OF PRIME ARRAY
 1050  BEEP   .EQ $FF3A    BEEP THE SPEAKER
 1060  *--------------------------------
 1070         .MA ZERO
 1080         STA ]1+$001,X
 1090         STA ]1+$101,X
 1100         STA ]1+$201,X
 1110         STA ]1+$301,X
 1120         STA ]1+$401,X
 1130         STA ]1+$501,X
 1140         STA ]1+$601,X
 1150         STA ]1+$701,X
 1160         .DO ]1<$5800
 1170         >ZERO ]1+$800
 1180         .FIN
 1190         .EM
 1200  *---------------------------------
 1210  *      MAIN CALLING ROUTINE
 1220  *
 1230  MAIN   LDA #100     DO 100 TIMES SO WE CAN MEASURE
 1240         STA COUNT    THE TIME IT TAKES
 1250         JSR BEEP     ANNOUNCE START
 1260  .1     JSR PRIME
 1270         DEC COUNT    CHECK COUNT
 1280         BNE .1       DONE ?
 1290         JMP BEEP     SAY WE'RE DONE
 1300  *---------------------------------
 1310  *      PRIME ROUTINE
 1320  *      SETS ARRAY STARTING AT BASE
 1330  *      TO $FF IF NUMBER IS NOT PRIME
 1340  *      CHECKS ONLY ODD NUMBERS > 3
 1350  *      INC = INCREMENT OF KNOCKOUT
 1360  *      N = KNOCKOUT VARIABLE
 1370  *--------------------------------
 1380  PRIME
 1390         LDX #1
 1400         STX SHCNT+1  STARTING MULTIPLIER FOR SQUARE
 1410         STX MULT+1
 1420         DEX
 1430         STX SQUARE+1
 1440         TXA          CLEAR WORKING ARRAY
 1450  .1     >ZERO BASE
 1460         INX          EVERY ODD LOCATION
 1470         INX
 1480         BEQ .2
```

```
1490        JMP .1       NOT FINISHED CLEARING
1500 *--------------------------------
1510 .2     LDA #3
1520        STA START+1
1530 MAINLP ASL          INC = START * 2
1540        STA INC+1
1550 SQUARE LDA #*-*     MOVE MULT TO N
1560        STA N+2
1570        LDA MULT+1
1580        ASL          MULTIPLY BY 8
1590        ROL N+2
1600        ASL
1610        ROL N+2
1620        ASL
1630        ROL N+2
1640        TAX
1650        INX          AND ADD 1
1660        BNE .1
1670        INC N+2
1680 .1     CLC          ADD BASE TO N
1690        LDA N+2
1700        ADC /BASE
1710        STA N+2
1720        TAY
1730        TXA
1740 LOOP
1750 N      STA $FF00,X  REMEMBER THAT N IS REALLY AT N+2
1760 INC    ADC #*-*     N = N + INC
1770        TAX
1780        BCC LOOP     DONT'T BOTHER TO ADD, NO CARRY
1790        INY          INC HIGH ORDER
1800        STY N+2
1810        CPY /BASE+$4000  IF IS GREATER THAN $6000
1820        BCC LOOP     NO, REPEAT
1830 START  LDX #*-*     GET OUR NEXT KNOCKOUT
1840 NEXT   INX
1850        INX          START = START + 2
1860        BMI END      WE'RE DONE IF X>$7F
1870        INC SHCNT+1  INCREMENT SQUARE MULTIPLIER
1880 SHCNT  LDA #*-*     AND ADD TO MULTIPLIER
1890        CLC
1900 MULT   ADC #*-*
1910        STA MULT+1
1920        BCC .1
1930        INC SQUARE+1
1940 .1     LDA BASE,X   GET A POSSIBLE PRIME
1950        BNE NEXT     THIS ONE HAS BEEN KNOCKED OUT
1960        STX START+1
1970        TXA
1980        BNE MAINLP   ...ALWAYS
1990 END    RTS
2000 *--------------------------------
2010 COUNT  .DA #*-*     COUNT FOR 100 TIMES LOOP
```

===============================================================================
DOCUMENT :AAL-8211:DOS3.3:SOUND.1.txt
===============================================================================

```
$4000:FF FF 00 00 N 4004<4000.47FFM
$4800:FF 00 N 4802<4800.4FFFM
$5000:F0 N 5001<5000.57FFM
$5800:CC N 5801<5800.5FFEM
```

===============================================================================
DOCUMENT :AAL-8211:DOS3.3:SOUND.2.txt
===============================================================================

```
$5800:CC N 5801<5800.5FFEM
$5000:AA N 5001<5000.57FEM
$4800:F8 3F 03 E0 N 4804<4800.4FFCM
$4000:FC 0F C0 00 N 4004<4000.47FCM
```

```
==========================================================================
DOCUMENT :AAL-8211:DOS3.3:SOUND.3.txt
==========================================================================

$4000:00 01 03 07 0F 1F 3F 7F FF FE FC F8 F0 E0 C0 80 N
4010<4000.5FEFM
```

```
=====================================================================
DOCUMENT :AAL-8211:DOS3.3:SOUND.4.txt
=====================================================================

$4000:00 FF 00 00 FF FF 00 00 00 00 FF FF FF FF N 400E<4000.5FFFM
```

```
=============================================================================
DOCUMENT :AAL-8211:DOS3.3:SOUND.5.txt
=============================================================================

$4000:00 88 00 00 88 88 00 00 00 00 88 88 88 88 N 400E<4000.5FFFM
```

================================================================================
DOCUMENT :AAL-8211:DOS3.3:Talk.A.Test.txt
================================================================================

( DTC removed -- lots of garbage characters )

```
================================================================================
DOCUMENT :AAL-8211:DOS3.3:TestRepeatUntil.txt
================================================================================
```

```
d≤TEST REPEAT/UNTILCnD$–Á(4):∫D$"BLOAD B.REPEAT/UNTIL":å768RxI –
0:ØREPE≈eÇI–I»1:∫I":  ";éáJ–0:ØREPE≈:J–J»1:∫J"
";:ØUNTILJœ14:∫ùåØUNTILI–10
```

```
==========================================================================
DOCUMENT :AAL-8211:DOS3.3:TONY.S.DRIVER.txt
==========================================================================

*
≤DRIVER FOR TONY'S FAST PRIME FINDER\(D$–Á(4):∫D$"BLOAD B.TONY'S
SUPER-FAST PRIMES"Ç2ó:¢10:ñ10:∫"HIT ANY KEY TO
START"ú<π49168,0:æA$:π49168,0ßZå32768“_â:ÅA–8195¡24576«2:≠,(A)–
0ƒ∫A…8192;" ";ÿbÇ
```

================================================================================
DOCUMENT :AAL-8212:Articles:AS.Src.Code.txt
================================================================================

Applesoft Source, Completely Commented.....Bob Sander-Cederlof

For several years I have been working on this. I even bought an assembler from another company just to get the promised source code of Applesoft that came in the package, but I was very disappointed. (No complaints, it was intended as a "freebie" with their assembler.) I wanted LOTS of comments, MEANINGFUL labels, and I wanted it in the format of the S-C Macro Assembler.

Now I have it. And you can have a copy, on two diskettes, for only $50. It comes in an encrypted form, with a driving program which creates the source code files on your machine with the aid of the Applesoft image in ROM or RAM. The encryption is meant to protect the interests of Apple and Microsoft.

You need two disk drives to assemble Applesoft, a printer to get a permanent listing, and of course you need the S-C Macro Assembler. The source code is split into more than a dozen source files, assembled together using a control file full of .IN directives. After assembling and printing, you will have well over 100 pages of the best documentation of Applesoft internals available anywhere.

In the process of writing the comments, I discovered some very interesting bugs. Some have been published before, and others I have never heard of. Just for fun, try this:

```
]A%=-32768     (you get an error message, correctly)
]A%=-32768.00049    (No error message!)
]PRINT A%           (You get 32767!)
```

Or open your disk drive doors, just in case, and type:

```
]LIST 440311
```

That last one can be disastrous! Any use of a six-digit line number (illegal, but not caught by Applesoft) between 437760 and 440319 will cause a branch to a totally incorrect place. For example, GOTO 440311 branches to $22D9!

The causes of these and other interesting phenomena, as well as some suggested improvements resulting in smaller/faster code, are documented in my comments.

=============================================================================
DOCUMENT :AAL-8212:Articles:Bit.Control.txt
=============================================================================

Add Bit-Control to Apple Monitor...........Bob Sander-Cederlof

The other day someone sent me a disk with an article for AAL on it as a binary file. The codes in the file were all ASCII characters, but they were nevertheless not compatible with any word processors I had around.

All blanks were coded as $A0 (hi-bit on), and all other characters were coded in the range $00-$7F (hi-bit off). Otherwise, everything was compatible with my favorite word processor (the one I am still in the process of finishing).

I need to have all the hi-bits set to one in the file, or in the memory image after BLOADing the file. That's the motivation for the following neat enhancement to the Apple monitor.

The enhancement hooks in through the control-Y user command vector. By merely typing:

```
     *80FF<2000.3FFF^Y    (^Y means control-Y)
```

I set all the hi-bits between $2000 and $3FFF.

The "80FF" in the command line above is the magic part of this enhancement. The last two digits are ANDed with every byte in the specified range, clearing every bit which is zero in those two digits. By using $FF, no bits are cleared. Other values for these two digits will clear whatever bits you wish.

The first two digits are ORed into every byte in the specified range, setting every bit which is one in the two digits. $80 in my example sets the top bit in every byte.

The code is designed to be BRUN from a binary file, and it is completely relocatable. You can BRUN it anywhere in memory that you have room for 36 bytes. That is why the SETUP code is longer than the actual control-Y code!

The SETUP routine first discovers where in memory it is running, and then sets up the control-Y vector in page 3 to point at the BITS code. The discovery is done in the usual way, by JSRing to a guaranteed RTS in the monitor ROM, at $FF58. This leaves a return address just below the stack pointer. I pick up that address and add the difference between that and BITS to get the appropriate control-Y vector pointer.

Line 1200 needs a little explanation. My assembler automatically switches to page zero addressing mode if the address is less than $100. STACK-1 is less than $100, so "ADC STACK-1,X" would assemble as though I wrote "ADC $FF,X". Indexed addressing in page zero mode stays in page zero, wrapping around. If X=3, "ADC $FF,X" would reference location $02 in page zero rather than $102. Therefore I had to use the ".DA #$7D" to force the assembler to use a full 16-bit address mode. (Some assemblers have a special way of forcing 16-bit addressing in cases like this; others require special marks to get zero-page modes.)

BITS itself is very straightforward code. The monitor leaves the starting address of the specified range in A1 ($3C,3D), the ending address in A2 ($3E,3F), and the mask in A4 ($42,43). The subroutine at $FCBA increments A1 and compares it to A2, leaving carry clear if the range is not yet complete.

=====================================================================
DOCUMENT :AAL-8212:Articles:ClearStrngArray.txt
=====================================================================

Save Garbage by Emptying Arrays............Bob Sander-Cederlof

As we all know, Applesoft programs which use a significant number of strings can appear to die for long periods of time while "garbage collection" is performed.  Many techniques have been published to reduce the frequency of garbage collection, or reduce the amount of garbage to be collected, or to speed up the collector.

Randy Wiggington published a much faster garbage collector in "Call-A.P.P.L.E.", January, 1981, pages 40-45.  The source code is available in S-C format on the Dallas Apple Corps disk of the month for March, 1981.  (Copies available for $10 from me.)  Randy's speed improvement is gained by searching for the highest 16 strings in memory at once, rather than just the highest one string.  It is much faster, but not the fastest.  The time for collection still varies quadratically with the number of strings in use.

Cornelius Bongers wrote the fastest collector I have seen.  It was published in "MICRO -- The 6502/6809 Journal", August, 1982, pages 90-97.  Corny's method is very straightforward, and has the advantage that execution time varies linearly with the number of strings in use. His method also has the disadvantage that it does not work if any strings contain any characters with the high bit on.  (Applesoft normally does not generate any strings with high-bit-set-characters, but you can do it with oddball code.)  I typed in the program from MICRO, made a few changes here and there, and found it to be lightning fast.

I installed the linear method in a client's program, and his garbage collection time went from 100 seconds after printing every four lines, to only 1/4 second.  Other changes, such as the one described below, cut the interval of collection to once every 12 lines.

It so happens that strings which are empty do not increase the garbage collection time.  Many times in Applesoft programs a string array is filled with data from a disk file, processed, and then filled with fresh data, and so on.  By emptying the array before each pass at reading the disk file, the number of active strings can be greatly reduced.

One of my programs was like this:  the one that prints your mailing label so that the AAL gets to you every month.  Before reading each file (the list is divided into about 12 different files, based on zip code and other factors), I wrote a loop that set each string in the array to a null string, and then forced garbage collection:

```
     FOR I = 1 TO NH : A$(I)="" : NEXT : F = FRE(0)
```

The only problem with this was that the loop takes so long! About ten seconds, anyway, enought to try my patience.

All the above motivated me to write the following little subroutine, which nulls out (or empties) a string array. Bongers included an array eraser in his article, which completely released an array; however, that requires re-DIMming the array each time. My program is faster in my case, and it was fun to write.

The program is listed below with the origin set at $300, so "CALL 768,arrayname" will activate it. It happens to be fully relocatable, so you can load it anywhere in memory you wish. You could easily add it to your Applesoft program with Amper-Magic or Amperware or The Routine Machine.

I used three subroutines inside the Applesoft ROMs. CHKCOM gives SYNTAX ERROR unless the next character is a comma. I use it to check for the comma that separates the call address from the array name. CHKSTR checks to make sure that the last-parsed variable is a string variable, and gives TYPE MISMATCH if not. GETARYPT scans an array name and returns the address of the start of the array in variable space.

If you look at page 137 of your Applesoft reference manual, you will see in the third column a picture of a string array. (Notice first the error: the signs of the first and second bytes of the string name are just the reverse of what the book says.) My program looks at the number of dimensions to determine the size of the preamble (the number of bytes before you get to the actual string pointers).

I use the preamble size to compute the starting address of the string pointers, and the number of bytes of string pointers that there are. Then a tight little loop stores zeros on top of all the descriptors.

The Applesoft program below illustrates the CLEAR subroutine in action.

================================================================
DOCUMENT :AAL-8212:Articles:Enhanced.6502.txt
================================================================

New Enhanced 6502 Nearly Here!.............Bob Sander-Cederlof

Nigel Nathan from Micro-Mixedware in Stow, MA, sent me some copies of reference material for the new 65C02 chip. This is the enhanced CMOS version, soon to be available from GTE and Rockwell.

Nigel's interest was that I might produced an enhanced S-C Macro Assembler to accommodate the new opcodes and addressing modes. I not only might...I did it right away! It is ready now, although you may have some difficulty getting the chips for a few more months.

Rockwell is sampling the 65C02 now, and scheduled for production in February. Rockwell is also readying an entire family of CMOS products to go with the 65C02, including 2Kx8 CMOS static RAM and multi-byte parallel interfaces.

The 65C02 is expected to be plug-compatible with the 6502 in your Apple II. In fact, Cliff Whitaker of Rockwell told me that the first chips they made were tested by plugging them into Apples. Hopefully you will be able to simply plug them in and start using the new opcodes. If true, then I will probably become a source for these chips.

What enhancements did they make? According to the GTE document, some "bugs" in the 6502 design were corrected:

* Indexed addressing across a page boundary no longer causes a false read at an invalid address.

* Invalid opcodes are now all NOPs, rather than doing exotic things such as I described in the March 1981 AAL.

* JMP indirect at a page boundary now operates correctly, at a cost of one additional cycle.

* Read/modify/write opcodes (like INC, DEC, and the shifts) now perform two reads and one write cycle rather than one read and two writes.

* The D-status bit is now set to binary mode (D=0) by reset; it used to be indeterminate.

* The N-, V-, and Z-status bits are now valid after ADC or SBC in decimal mode (D=1); they used to be invalid, requiring special tests. The cost is one additional cycle for all ADCs and SBCs in decimal mode.

* An interrupt after fetch of a BRK opcode defers to the BRK. It used to cause the BRK to be ignored.

The Rockwell literature makes reference to the following new opcodes, or new addressing modes for old opcodes:

```
80   BRA rel    Branch Always

12   ORA (zp)   )
32   AND (zp)   )
52   EOR (zp)   )  new addressing mode:
72   ADC (zp)   )
92   STA (zp)   )  zero-page indirect
B2   LDA (zp)   )
D2   CMP (zp)   )  without indexing
F2   SBC (zp)   )

04   TSB zp     Test and set bits
14   TRB zp     Test and reset bits
34   BIT zp,X   new addressing mode for BIT
64   STZ zp     Store Zero
74   STZ zp,X     "    "

07-77   RMB b,zp   Reset bit b in zp
87-F7   SMB b,zp   Set bit b in zp

89   BIT imm    new addressing mode for BIT

1A   INC        Increment A-register
3A   DEC        Decrement A-register
5A   PHY        Push Y
7A   PLY        Pull Y
DA   PHX        Push X
FA   PLX        Pull X

0C   TSB abs    Test and set bits
1C   TRB abs    Test and reset bits
3C   BIT abs,X  new addressing mode for BIT
7C   JMP (abs),X   new addressing mode for JMP
9C   STZ abs    Store zero

9E   STZ abs,X  Store zero

0F-7F   BBR b,zp,rel   Branch if bit b in zp is zero
8F-FF   BBS b,zp,rel   Branch if bit b in zp is one
```

Let your imagination run wild with all the great ways to use these new opcodes!  If you feel the need for the ability to assemble them now, the Cross Assembler upgrade for the 65C02 is available for $20 to subscribers of the Apple Assembly Line who already own the S-C Macro Assembler.

=============================================================================
DOCUMENT :AAL-8212:Articles:Enhancemnt.Rvw.txt
=============================================================================

Enhancing Your Apple II (A Review)..................Bill Morgan

Don Lancaster, the well-known author of electronics books for the hobbyist (and a subscriber to AAL), has now entered the Apple arena in a big way. His latest book, "Enhancing Your Apple II, Vol. 1", promises to be the start of a long series of easy-to-use guides to the important internal workings of the Apple.

The main enhancements he offers in this volume are simple modifications to the Apple's video circuitry, to allow EXACT software access to the video timing. This permits your program to play all sorts of tricks with the display modes. There is also a wealth of information on the Apple's techniques of video storage and output.

The basic hardware modification is a single wire from an IC in the video circuitry to either the cassette or the game input. With this wire and a little bit of code, it is easy to switch display modes between screen scans, avoiding a lot of messy glitches on the screen. With more code, and careful timing, you can lock the processor to the display timing and switch between text and graphic modes (hi-res or lo-res) in mid-line.

There is also a very good 60-page chapter on disassembling and understanding other people's programs. Don presents a novel technique of color-coding a printout of a monitor disassembly, to bring out the structure of a program and the function of each routine. The example program is Apple's High-Res Character Generator, from the DOS Toolkit. He later uses the information discovered about the character generator and the Hi-Res display to develop a slower and smoother scrolling routine for Hi-Res text.

He shows us other enhancements, as well. There are two different ways to attach a modulator's output line to your TV set, avoiding that clumsy little switch box that the manufacture gives us. How about a programmable color-killer circuit? With this one you can have software control of color vs. black-and-white display. There are sections about generating extra colors, in both Hi-Res and Lo-Res graphics.

In the back of the book are postcards for sending feedback and ordering other materials. All the code in the book (26 programs) can be ordered on diskette, for $14.95. He uses the DOS Toolkit Assembler, but we plan to talk to him about providing the programs in S-C format. You can also order a kit of the parts for all of the hardware modifications he describes. That kit costs only $11.95 + shipping, from a dealer in Oklahoma. Future plans include more volumes of enhancements and a possible bulletin board system for updates to the books.

All in all, "Enhancing Your Apple II" looks to be an important and useful book. Like all of Lancaster's books, it is published by Howard W. Sams. It is 232 pages long, size 8 1/2 X 11 inches, and sells for $15.95. We have ordered a stock here at S-C, and will sell them for $15.00 + postage.

For Volume 2 of the "Enhancing" series, he has promised us more video techniques, a keyboard enhancer, something called an "Adventure Emergency Toolkit", graphics software for daisy-wheel printers, a two-dollar interface for the BSR controller, and much more. I'm looking forward to it!

This is a good time to mention another of Don's books, which has received too little attention. I am speaking of "The Incredible Secret Money Machine". Despite the title, it is not a get-rich-quick pamphlet, but rather a very, very useful guide to starting and operating a free-lance technical or craft business. "Money Machine" is 160 pages of tightly packed information on strategy and tactics, getting started, and dealing with customers, suppliers, and the government.

There is enough practical advice on communication, both verbal and graphic, to make up several courses in advertising and technical writing. Bob and I refer to this book regularly, and have long felt that it is one of the best books around for the budding entrepeneur. We have also ordered "Money Machine", and will sell it for $7.50 + postage.

Last minute addition: We just received a review copy of another new book from Don Lancaster, Micro Cookbook Vol. 1 - Fundamentals. This one is a very basic introduction to microcomputer principles. He talks about how to learn and what to learn, and introduces some hardware fundamentals. He also promises Vol. 2, about Machine-Language Programming. It looks very good; I'll have more details next month. We especially like this sentence at the end of the Preface: This book is dedicated to the 6502.

```
==============================================================================
DOCUMENT :AAL-8212:Articles:Es.Cape.Patch.txt
==============================================================================
```

Add a New Feature to ES-CAPE.......................Bill Linn

Since most of the owners of ES-CAPE also subscribe to the Apple Assembly Line, I thought I would pass on some neat patches here.

The automatic line numbering feature in ES-CAPE can be enhanced by the following patches, which make the numbers track whatever you type. With these patches, each time an Applesoft line is changed via hitting return or via a CHANGE command, that line number plus the current increment becomes the next automatic line number to be generated when the space bar is pressed. (Now ES-CAPE works more like the S-C Macro Assembler.)

Here are the patches for the mother-board version:

```
LOAD ES-CAPE
CALL -151
E44:69 00 EA
102E:4C 51 9B
1C51:A5 51 8D 34 9B 18 A5 50 6D 35 9B 8D 33 9B
:90 03 EE 34 9B A5 25 F0 02 C6 25 4C 34 8F
3D0G
SAVE ES-CAPE REVISED
```

Here are the patches for the RAM-card version:

```
LOAD ES-CAPE LC
CALL -151
E41:69 00 EA
102B:4C 60 E1
1B60:A5 51 8D 51 E1 18 A5 50 6D 52 E1 8D 50 E1
:90 03 EE 51 E1 A5 25 F0 02 C6 25 4C 31 D6
3D0G
SAVE ES-CAPE LC REVISED
```

I believe these patches make ES-CAPE even easier to use. If any of you still have not upgraded your AED II copies to ES-CAPE, send me $10 and your old disk and I'll return a new version and the great new manual.

I am continuing to work on ES-CAPE, hoping for a new version around the middle of next year. What new features would you like? The main ones we have in mind are 80-column support, renumber, and merge. If you have some good ideas, drop me a line at 3199 Hammock Creek, Lithonia, GA 30058.

================================================================================
DOCUMENT :AAL-8212:Articles:Front.Page.txt
================================================================================

$1.50

Volume 3 -- Issue 3 December, 1982

In This Issue...



Cross Assemblers continue to appear. We now have ready a version for the Intel 8048, and one for the yet unreleased Rockwell 65C02. More on the latter inside.

Don Lancaster, famous author of many books published by Howard Sams, says the Apple II is probably going to have a greater impact on history than the automobile or television! Perhaps verging on Applolatry, but you will surely enjoy his new book. See Bill's reviews inside.

If I am trying to learn how to program in assembly language, or to increase my skill at it, what (besides AAL) should I read? I strongly recommend Softalk, Nibble, Micro, and Call APPLE. Every month they publish excellent examples of assembly language programs which you can study, modify, and use. As for books, Roger Wagner's, Lance Leventhal's, and Don Lancaster's are my favorites at this time.

```
==============================================================================
DOCUMENT :AAL-8212:Articles:Lancaster.Addtn.txt
==============================================================================
```

Last last-minute addition:  Don sent me a copy of the program disk, and I am now converting the source files to S-C format.  By the time you read this, he will have the S-C code.  When you order the diskette from Synergetics (Lancaster's company), just mention that you want the version with the S-C files.

================================================================================
DOCUMENT :AAL-8212:Articles:ListOnTXTFile.txt
================================================================================

Assembly Listings on TEXT Files............Bob Sander-Cederlof

Today Jules Gilder called, asking for patches to allow sending the assembly listing to a TEXT file. He is in the process of writing a book, and wanted the listings on TEXT files so they could be automatically typeset.

My first response was a familiar one: "It is a very difficult problem, because of the interaction of .IN and .TF files."

"But I don't need .TF or .IN files!", he swiftly parried.

Suddenly in a flash of insight I saw that it could be EASILY done. All that is needed is to patch the .TF directive so that it opens a TEXT file, but does not set the TF-flag. Then all listing output will go to the text file, and the object code will go to the current origin or target address.

Here are the patches for the motherboard version:

```
       :$2998:A5 60 D0 03 20 6A 2A 4C 04 1F
```

And the language card version:

```
       :$C083 C083 EAE4:A5 60 D0 03 20 B6 EB 4C 50 E0
```

Note that the two "C083's" above write-enable the language card so the patches will be effective.

After the patches are installed, a .TF directive will direct the listing to your text file. Here is an example:

```
       .TF T.LISTING
SAMPLE LDA #3
       STA $06
       RTS
```

Just remember that the normal use of .TF is not available when this patch is in place; also that .IN should not be used. Using .IN would turn off the listing output, directing it back to the screen.

==============================================================================
DOCUMENT :AAL-8212:Articles:LoadRAMCard.txt
==============================================================================

Clarification on Loading the RAM Card...........Paul Schlyter

Last month Bob S-C told how to use an EXEC command file without ENDing an Applesoft program. His example may have obfuscated the process of loading a file into a RAM card, because it really is not necessary to use an EXEC file for this purpose.

You can BLOAD into the RAM card without leaving Applesoft, contrary to Bob's information, by merely write-enabling it. The soft-switches $C081 and $C089 write-enable the RAM card (with bank 2 or bank 1 at $D000, respectively), leaving the motherboard ROMs read-enabled. This means everything you write goes into the RAM card, and everything you read comes from the motherboard ROMs. Thus you can simply BLOAD into the RAM card, and BLOAD will write to those addresses.

Here is a short program that loads the whole 16K RAM card, all from within a running Applesoft program, without EXEC files.

```
100 D$ = CHR$ (4)
110 B2 = 49281 : REM $C081 -- SELECT BANK TWO
120 B1 = 49289 : REM $C089 -- SELECT BANK ONE
130 P = PEEK(B2) + PEEK(B2) : REM WRITE ENABLE BANK TWO
140 PRINT D$"BLOAD LC.BANK 2"
150 P = PEEK(B1) + PEEK(B1) : REM WRITE ENABLE BANK ONE
160 PRINT D$"BLOAD LC.BANK 1"
```

[ Note from Bob S-C: My face is red! Paul will note that I modified his program above in lines 130 and 150. He wrote "130 POKE B2, PEEK(B2)" and similarly for line 150. However, some RAM cards, such as my Andromeda board, will disable write-enable if the soft-switches are addressed during a write-cycle. The POKE does just that; therefore, I changed 130 and 150 to do two PEEKs in a row. Further, I recall when working with a Corvus drive last year that BLOADing from a Corvus into the RAM card did not work unless a monitor had already been copied into the space from $F800-$FFFF. ]

```
==================================================================================
DOCUMENT :AAL-8212:Articles:My.Ad.txt
==================================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
S-C Macro Cross Assembler Modules
    65C02 Version.................................................$20.00
    6800/6801/6802 Version........................................$32.50
    6809 Version..................................................$32.50
    Z-80 Version..................................................$32.50
    68000 Version.................................................$50.00
    Requires ownership of S-C Macro Assembler.
    Each disk includes regular and language card versions.
Upgrade from Version 4.0 to MACRO.................................$27.50
Source code of Version 4.0 on disk................................$95.00
    Fully commented, easy to understand and modify to your own tastes.

Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.

ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)......(regular $79) $49.00
    Special price to AAL readers only, until 12/31/82!
Source Code for FLASH! Runtime Package............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware).......................$45.00
Apple White Line Trace (Lone Star Industrial Computing)............$50.00
    (A unique learning tool)

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $4.50
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                                100 or more:  25 cents each

Ashby Shift-Key Mod...............................................$15.00
Paymar Lower-Case Adapter.........................................$37.50
    For Apples before Revision 7 only
Lower-Case Display Encoder ROM....................................$25.00
    Works only Revision level 7 Apples.  Replaces the encoder ROM.

Books, Books, Books........................compare our discount prices!
```

```
"Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
"Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
"Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
"Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
"Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
"Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
"Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
"What's Where in the Apple", Second Edition..........($24.95)  $23.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

=====================================================================
DOCUMENT :AAL-8212:Articles:Quickies.txt
=====================================================================

Quickie No. 1..............................Bob Sander-Cederlof

To merge selected bits from one byte with the rest of the bits of another byte:

```
  Code           Example
LDA MASK         00011111
EOR #$FF         11100000
ORA BYTE1        111xxxxx
STA TEMP
LDA BYTE2        yyyzzzzz
ORA MASK         yyy11111
AND TEMP         yyyxxxxx
```

Quickie No. 2..............................Bob Sander-Cederlof

To test a byte in memory without disturbing any registers:

```
INC BYTE
DEC BYTE       Restore value, and test against zero
BEQ ....
```

Quickie No. 3..............................Bob Sander-Cederlof

To shift a two-byte value right one bit with sign extension:

```
LDA HI.BYTE
ASL            SIGN BIT INTO CARRY
ROR HI.BYTE    HI BYTE RIGHT ONE, CARRY (SIGN) INTO BIT 7
ROR LO.BYTE
```

Quickie No. 4..............................Bob Sander-Cederlof

To print a two byte value in hexadecimal:

```
LDA HI.BYTE
LDX LO.BYTE
JSR $F941
```

==============================================================================
DOCUMENT :AAL-8212:Articles:RelocJMPsMeyer.txt
==============================================================================

Making Internal JMPs and JSRs Relocatable.........Peter Meyer

A machine language routine is said to be relocatable if it can function properly regardless of its absolute location in memory. If a routine contains a JMP or a JSR to an INTERNAL address then it is not relocatable; if it is run in another part of memory then the internal JSR or JMP will still reference the former region of memory. JMPs and JSRs to subroutines at absolute locations (e.g. in the Monitor) do not impair the relocatability of a routine.

I will explain here a technique whereby you can, in effect, perform internal JSRs and JMPs without impairing the relocatability of your routine. There is a small price to pay for this: namely, an increase in the length of your routine. Your routine must be preceded by a 2-part Header Routine which is 43 bytes long. In addition, each internal JSR requires 8 bytes of code, and each internal JMP requires 11 bytes of code.

No tables or other data storage are required, except that three bytes must be reserved for a JMP instruction. These three bytes can be anywhere in memory, but must be at an absolute location. There are three bytes that normally are used only by Applesoft, namely, the ampersand JMP vector at $3F5 to $3F7. Since we are here concerned only with machine language routines in their own right, we can use the locations $3F5 to $3F7 for our own purposes. However, other locations would do just as well.

The technique is fully illustrated in the accompanying assembly language program. This routine consists of three parts:

(1) Header Part 1 (SETUP), which sets up a JMP instruction at VECTOR (at $3F5-$3F7, but could be different, as explained above) to point to Header Part 2.

(2) Header Part 2 (HANDLER), which is a 15-byte section of code whose task is to handle requests to perform internal JSRs and JMPs (more on this below).

(3) The main part of the routine, in which internal JSRs and JMPs (in effect) are performed using macro instructions.

When your routine (including the Header) is executed, the first thing that happens is that Header Part 1 locates itself (using the well-known JSR $FF58 technique), then places a JMP HANDLER at VECTOR. Thereafter a JMP VECTOR is equivalent to JMP HANDLER, and a JSR VECTOR is equivalent to a JSR HANDLER. The HANDLER routine handles requests from your routine for internal JSRs and JMPs. To perform a JSR to an internal subroutine labelled SUBR simply include the following code:

```
HERE   LDA #SUBR-HERE-7 low byte of offset
       LDY /SUBR-HERE-7 high byte of offset
       TSX
       JSR VECTOR
```

As explained above, the JSR VECTOR is in effect a JSR HANDLER. The Header Part 2 code takes the values in the A and Y registers and adds them to an address which it obtains from the stack to obtain the address of SUBR. It then places this address in INDEX ($5E,5F) and executes "JMP (INDEX)".

An internal JMP, from one part of your routine to another, is performed in a similar manner. Suppose you wish to JMP from HERE to THERE. It is done as follows:

```
HERE   LDA #THERE-HERE-7 low byte of offset
       LDY /THERE-HERE-7 high byte of offset
       TSX
       JSR $FF58
       JMP VECTOR
```

Since we are (in effect) performing a JMP, rather than a JSR, we do a JMP VECTOR rather than a JSR VECTOR. The other difference is that we have a JSR $FF58 following the TSX.

Clearly the sequence of instructions which allows you to perform a JMP or a JSR could be coded as a macro. The macros to use are shown in the accompanying program listing. By using macros an internal JMP or JSR can be performed with a single macro instruction bearing a very close resemblance to a real JSR or JMP instruction.

The following program, which consists of the Header Routine plus a demonstration routine, can be assembled to disk using the .TF directive. It can then be BRUN at any address and it will function properly. Thus it is relocatable, despite the fact that there are (in effect) an internal JMP and two internal JSRs performed.

When performing an internal JSR or JMP using my techniques, it is not possible to pass values in the registers, since these are required to pass information to the HANDLER routine. Nor is it advisable to try to pass parameters on the stack, even though the HANDLER routine does not change the value of the stack pointer. Better is to deposit values in memory and retrieve them after the transition has been made.

The HANDLER routine passes control to the requested part of your routine using a JMP indirect. (INDEX at $5E,5F, has been used in the accompanying program, but any other address would do as well, provided that it does not cross a page boundary.) This means that the section of your routine to which control is passed (whether or not it is a subroutine) may find its own location by inspecting the contents of the location used for the JMP indirect. This feature of this technique is also illustrated in the accompanying program, in the PRINT.MESSAGE subroutine.

The use of internal data blocks is something not normally possible in a relocatable routine, but it can be done if the techniques shown here are used.

This method of performing internal JSRs and JMPs in a relocatable routine may be simplified if the routine is intended to function as a subroutine appended to an Applesoft program.  If the subroutine is appended using my utility the Routine Machine (available from Southwestern Data Systems), then it is not necessary to include the 47-byte Header Routine.  Internal JMPs and JSRs can still be performed exactly as described above, except that the address of VECTOR must be $3F5-$3F7.  This technique is not described in the documentation to the Routine Machine.  A full explanation may be found in the Appendix to the documentation accompanying Ampersoft Program Library, Volume 4 (also available from Southwestern Data Systems).

```
==================================================================
DOCUMENT :AAL-8212:Articles:Split.txt
==================================================================
```

Splitting Strings to Fit Your Display......Bob Sander-Cederlof

Printing text on the screen, or even on a printer, is not as easy at it ought to be. The problem is splitting words at the right margin. Word processors handle it nicely, but what do you do in an Applesoft program?

You might write a subroutine, in Applesoft, which looks for the first space between words before a specified column position. The subroutine could split a string at the space into two substrings: one containing the next display line, the other the remainder of the original string.

You might. But believe me, it builds up a lot of garbage strings and takes a long time to execute. If you like the general approach, you might try coding the subroutine in assembly language. You can avoid garbage build-up and save lots of running time, so it is probably worth the effort. Especially since I already wrote the program for you!

The program is written to be called from an ampersand parser like the one in last month's article on REPEAT/UNTIL. Or, you can use it with Amper-Magic, Amperware, The Routine Machine, etc. It is fully relocatable, having no internal data or JMP/JSR addresses. I set the origin to $300, but it can be loaded and used anywhere without re-assembly.

Here is an Applesoft program to show how to call SPLIT:

```
100  POKE 1014,0: POKE 1015,3
120  FOR N = 5 TO 40 STEP 3: GOSUB 1000: NEXT : END
1000 A$ = "NOW IS THE TIME FOR ALL GOOD MEN TO COME
     TO THE AID OF THEIR PARTY."
1005  & ,A$,B$,N
1010  PRINT B$
1020  IF A$ <  > "" THEN 1005
1025  PRINT
1030  RETURN
```

Call SPLIT with three parameters. The first (A$ above) is the source string, which will be split. After SPLITting, the remainder string will be left in A$.

The second parameter, B$ above, will receive the left part, including the last complete word, up to N (the 3rd parameter) characters. If there is no space in the left N characters, exactly N characters will be split.

Here are some of the printouts from the test program:

```
N=5            N=11            N=20
-----          -----------     --------------------
NOW            NOW IS THE      NOW IS THE TIME FOR
IS             TIME FOR        ALL GOOD MEN TO COME
THE            ALL GOOD        TO THE AID OF THEIR
TIME           MEN TO COME     PARTY.
...etc.        ...etc.
```

==============================================================
DOCUMENT :AAL-8212:Articles:Toggle.Case.txt
==============================================================

Toggling Upper/Lower Case in the S-C Macro Assembler.........

Steven Mann
Psychology Dept.
Univ. of So. Dakota
P. O. Box 7187
Grand Forks, ND 58202

I have made the necessary modifications to the assembler (and to my Apple) that allow me to enter lower case characters in my source programs, but have found that I like to have all upper case in certain sections (the labels and opcodes) and mixed (mostly lower) case only in the comment field. In order to do accommodate this most effectively, I wanted to be able to toggle back and forth from upper to lower case while I was entering my source code.

The simplest solution for me was to patch the assembler to accept one of the escape key sequences as an upper/lower case toggle. From back issues of AAL I was able to determine that a table of address vectors for the escape keys A-L is maintained from $1467 thru $1482 ($D467 thru $D482 in the language card version). Each two-byte entry is the address-1 (low byte first) of the routine that will handle that particular escape sequence.

Certain of the sequences are already taken (e.g. ESC L loads a disk file; ESC I,J,K, and M move the cursor, etc.) Since I don't use the ESC A,B,C,D cursor moves, I selected the ESC A sequence as the code for the case toggle. I also used ESC C for "CATALOG", as suggested by Bill Morgan some months back in these pages.

Implementation of the toggle is accomplished with the following patches to the HELLO program (for the RAM version of the assembler.) First, line 50 should be changed to:

```
50 PRINT A: IF A=1 THEN PRINT CHR$(4)"BLOAD S-C.ASM.MACRO"
   :GOSUB 200: CALL 4096
```

The subroutine at 200 is as follows:

```
197 REM
198 REM ESC A TOGGLES UP/LOW CASE
199 REM
200 POKE 5229,109:POKE 5230,165
210 FOR I=1 TO 9:READ J:POKE 48350+I,J:NEXT
220 DATA 173,22,16,73,255,141,22,16,96
230 RETURN
240 REM
250 REM ROUTINE RESIDES AT $BCDF
260 REM
```

```
270 REM CODE IS AS FOLLOWS:
280 REM
290 REM     LDA $1016
300 REM     EOR #FF
310 REM     STA $1016
320 REM     RTS
330 REM
```

Note that I put the patch code at $BCDF, which is in an unused area inside DOS 3.3. If you have already used this area for other purposes, you can tack the patch on to the end of the assembler image instead.

The actual code is very simple. The upper/lower case flag is stored at $1016 and is either a $00 or a $FF (in binary all zeros or all ones.) Toggling the flag involves loading the current flag and EORing it with #$FF. This will cause all set bits to be cleared and all clear bits to be set, so the zeros become ones and the ones become zeros. Thus, an #$FF byte becomes a #$00 or a #$00 becomes an #$FF. The new flag value is then stored back in $1016.

For the language card version the program is basically the same, but slightly longer due to the need to first write enable the language card. The code looks like this:

```
PATCH   LDA $C083       Two of these in succession
        LDA $C083       write-enable the card
        LDA $D016       Get the flag
        EOR #$FF        Complement it
        STA $D016       Save the new flag
        LDA $C080       Write protect the card
        RTS
```

I put the code in the same place as in the RAM version ($BCDF) and put it into memory from the LOAD LCASM exec file which loads the assembler onto the card. Two lines need to be added to the file. Between lines 1070 and 1080 (assuming your version has not been modified) you need to place these two lines:

```
1072 D469:DE BC
1074 BCDF:AD 83 C0 AD 83 C0 AD 16 D0 49 FF 8D 16 D0 AD 80 C0 60
```

The first line places the address of the case toggle handler in the escape vector table and the second line contains the assembled source code listed above. If you are not sure how to modify the LOAD LCASM file see the step by step description given in the May 1982 AAL (page 3).

After you have made the patch, experiment with the toggle. One particularly valuable feature is that you can toggle the case within a single line as you enter the line. This means that you can enter the label and opcode in upper case, tab over to the comment field, toggle the case flag, and then enter your comment in lower case.

I have found using the case toggle to be easy while improving the appearance and readability of my source listings. The only problem I have encountered so far is that the flag can not be toggled from within the edit mode (either case can be used in the edit mode, but you can't change the case in the middle of editing.) If any of you find a way to add this to the assembler be sure to let me know.

[ P.S. If you haven't put in the automatic catalog yet, here is an easy way. Add the following line to your LOAD LCASM file:

```
     1076 D46D:6D A5
```

and then ESC C will do a catalog as long as you don't mind having to hit return at the end of the catalog. For the motherboard version, add:

```
     205 POKE 5225,222: POKE 5226,188
```

in the subroutine I've added to the HELLO program. ]

{Ouch! Why didn't I think of that? One caution: With this method ESC C will do a CATALOG even if you are in the middle of typing a line. . . . Bill Morgan}

```
==============================================================================
DOCUMENT :AAL-8212:DOS3.3:Meyers.Reloc.txt
==============================================================================

 1010         .TF B.MEYER.1
 1020  *SAVE S.MEYER.1
 1030  *---------------------------------
 1040  *   SETUP AND HANDLER ROUTINES
 1050  *   TO ALLOW INTERNAL JSRS AND
 1060  *   JMPS IN A RELOCATABLE MACHINE
 1070  *   LANGUAGE ROUTINE
 1080
 1090  *   BY PETER MEYER, 11/3/82
 1100  *---------------------------------
 1110  *   LOCATIONS
 1120
 1130  INDEX      .EQ $5E,5F
 1140  STACK      .EQ $100 - $1FF
 1150  VECTOR     .EQ $3F5 - $3F7
 1160  *---------------------------------
 1170  *   MACRO DEFINITIONS
 1180
 1190         .MA JSR
 1200  :1     LDA #]1-:1-7
 1210         LDY /]1-:1-7
 1220         TSX
 1230         JSR VECTOR
 1240         .EM
 1250
 1260         .MA JMP
 1270  :1     LDA #]1-:1-7
 1280         LDY /]1-:1-7
 1290         TSX
 1300         JSR $FF58
 1310         JMP VECTOR
 1320         .EM
 1330  *---------------------------------
 1340  *   HEADER PART 1
 1350
 1360  SETUP  JSR $FF58    FIND OURSELVES
 1370         TSX
 1380         CLC
 1390         LDA #HANDLER-SETUP-2
 1400         .DA #$7D,STACK-1   FORCE ABS,X MODE
 1410         STA VECTOR+1
 1420
 1430         LDA /HANDLER-SETUP-2
 1440         ADC STACK,X
 1450         STA VECTOR+2
 1460
 1470         LDA #$4C     "JMP"
 1480         STA VECTOR
 1490         BNE MAIN.ROUTINE   ALWAYS
```

```
1500 *--------------------------------
1510 *   HEADER PART 2
1520
1530 HANDLER
1540
1550 *   ON ENTRY A,Y HOLD OFFSET
1560 *   FOR JMP OR JSR FROM ROUTINE
1570 *   X IS STACK POINTER FROM BEFORE LAST JSR
1580
1590        CLC
1600        .DA #$7D,STACK-1   FORCE ABS,X MODE
1610        STA INDEX
1620        TYA
1630        ADC STACK,X
1640        STA INDEX+1
1650        JMP (INDEX)
1660 *--------------------------------
1670 *      MAIN ROUTINE, FOR EXAMPLE
1680 *--------------------------------
1690 MSG    .EQ $06 AND $07
1700 CH     .EQ $24
1710 CV     .EQ $25
1720 INVFLG .EQ $32
1730 COUNT  .EQ $3C
1740 SETTXT .EQ $FB39
1750 VTABZ  .EQ $FC24
1760 HOME   .EQ $FC58
1770 COUT   .EQ $FDED
1780 *--------------------------------
1790 MAIN.ROUTINE
1800        JSR SETTXT
1810        JSR HOME
1820 MAIN.LOOP
1830        LDA #190
1840        STA COUNT
1850 .1     LDA #AALQT-PRINT.MESSAGE
1860        STA MSG
1870        LDA /AALQT-PRINT.MESSAGE
1880        STA MSG+1
1890        >JSR PRINT.MESSAGE
1900        DEC COUNT
1910        BNE .1
1920        LDA #LONGQT-PRINT.MESSAGE
1930        STA MSG
1940        LDA /LONGQT-PRINT.MESSAGE
1950        STA MSG+1
1960        >JSR PRINT.MESSAGE
1970        >JMP FORWRD
1980
1990 *--------------------------------
2000 PRINT.MESSAGE
2010        CLC
2020        LDA MSG      CHANGE RELATIVE ADDRESS TO
2030        ADC INDEX    AN ABSOLUTE ADDRESS, BY
```

```
2040        STA MSG      ADDING THE OFFSET
2050        LDA MSG+1
2060        ADC INDEX+1
2070        STA MSG+1
2080        LDY #0       POINT AT FIRST CHAR OF MSG
2090 .1     LDA (MSG),Y  GET NEXT CHAR OF MSG
2100        BMI .2       IT IS LAST CHAR
2110        ORA #$80     MAKE APPLE VIDEO FORM
2120        JSR COUT     PRINT IT
2130        INY          ADVANCE POINTER
2140        BNE .1       ...ALWAYS
2150 .2     JMP COUT     PRINT AND RETURN
2160 *--------------------------------
2170 *   256 BYTES TO JUMP OVER, JUST FOR ILLUSTRATION
2180
2190        .BS $100
2200 *--------------------------------
2210 *   TOGGLE INVERSE FLAG, AND HOME CURSOR
2220
2230 FORWRD LDA INVFLG
2240        EOR #$C0
2250        STA INVFLG
2260        LDA #0
2270        STA CH
2280        STA CV
2290        JSR VTABZ
2300        >JMP MAIN.LOOP
2310 *--------------------------------
2320 AALQT  .AT /AAL /
2330 LONGQT .HS 0D0D
2340        .AS / A P P L E   A S S E M B L Y   L I N E /
2350        .HS 0D02
2360        .AT /  S - C   S O F T W A R E   C O R P .  /
```

```
==========================================================================
DOCUMENT :AAL-8212:DOS3.3:S.BITS.txt
==========================================================================

 1000  *--------------------------------
 1010  *       MONITOR CTRL-Y COMMAND
 1020  *
 1030  *       TO SET AND CLEAR ANY COMBINATION
 1040  *       OF BITS IN A RANGE OF MEMORY
 1050  *
 1060  *       *MASK<ADR1.ADR2Y   (WHERE Y MEANS CTRL-Y)
 1070  *
 1080  *       MASK = XXYY  BITS = 0 IN YY WILL BE CLEARED
 1090  *                    BITS = 1 IN XX WILL BE SET
 1100  *--------------------------------
 1110  A1      .EQ $3C
 1120  A4L     .EQ $42
 1130  A4H     .EQ $43
 1140  STACK   .EQ $100
 1150  *--------------------------------
 1160  SETUP   JSR $FF58    FIND SELF FIRST
 1170          TSX
 1180          CLC
 1190          LDA #BITS-SETUP-2
 1200          .DA #$7D,STACK-1    FORCE ABS,X MODE
 1210          STA $3F9           MONITOR CTRL-Y VECTOR
 1220          LDA /BITS-SETUP-2
 1230          ADC STACK,X
 1240          STA $3FA
 1250          RTS
 1260  *--------------------------------
 1270  BITS    LDA (A1),Y   GET NEXT BYTE IN SPECIFIED RANGE
 1280          AND A4L      CLEAR BITS USING LO-BYTE OF MASK
 1290          ORA A4H      SET BITS FROM HI-BYTE OF MASK
 1300          STA (A1),Y   STORE MODIFIED BYTE
 1310          JSR $FCBA    ADVANCE POINTER
 1320          BCC BITS     MORE IN RANGE
 1330          RTS          FINISHED
 1340  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8212:DOS3.3:S.SPLIT.txt
==============================================================================

 1000  *SAVE S.SPLIT
 1010  *--------------------------------
 1020  *       & SPLIT,A$,B$,W
 1030  *
 1040  *       A$ -- SOURCE STRING
 1050  *       W  -- MAXIMUM WIDTH OF SPLIT
 1060  *
 1070  *       B$ -- LEFT W CHARS OF A$
 1080  *       A$ -- REST OF A$
 1090  *
 1100  *--------------------------------
 1110         .OR $300
 1120         .TF B.SPLIT
 1130  *--------------------------------
 1140  LINNUM     .EQ $50,51
 1150  DPTRA      .EQ $06,07
 1160  DPTRB      .EQ $08,09
 1170  SPTRA      .EQ $FE,FF
 1180  *--------------------------------
 1190  AS.CHKCOM  .EQ $DEBE
 1200  AS.PTRGET  .EQ $DFE3
 1210  AS.GETADR  .EQ $E752
 1220  AS.FRMNUM  .EQ $DD67
 1230  *--------------------------------
 1240  SPLIT  JSR AS.CHKCOM     GET COMMA
 1250         JSR AS.PTRGET     GET SOURCE STRING
 1260         STA DPTRA
 1270         STY DPTRA+1
 1280         JSR AS.CHKCOM     ANOTHER COMMA
 1290         JSR AS.PTRGET     GET TARGET STRING
 1300         STA DPTRB
 1310         STY DPTRB+1
 1320         JSR AS.CHKCOM     ANOTHER COMMA
 1330         JSR AS.FRMNUM
 1340         JSR AS.GETADR     GET MAXIMUM WIDTH
 1350         LDY #2
 1360         LDA (DPTRA),Y
 1370         STA SPTRA+1
 1380         STA (DPTRB),Y
 1390         DEY
 1400         LDA (DPTRA),Y
 1410         STA SPTRA
 1420         STA (DPTRB),Y
 1430         DEY
 1440         LDA LINNUM
 1450         CMP (DPTRA),Y
 1460         BCC .1
 1470         LDA (DPTRA),Y     A$ SHORTER THAN OR SAME AS W
 1480         STA (DPTRB),Y
```

```
1490        LDA #0
1500        STA (DPTRA),Y
1510        RTS
1520 *--------------------------------
1530 .1     TAY
1540 .2     LDA (SPTRA),Y     LOOK AT SPLIT BOUNDARY
1550        CMP #$20          FOR A BLANK
1560        BEQ .3            FOUND ONE
1570        DEY
1580        BNE .2            BACK UP THE SPLIT
1590 *---NO BLANK IN W CHARS----------
1600        LDA LINNUM
1610        BNE .4       ...ALWAYS
1620 *--------------------------------
1630 .3     TYA
1640        INY          SKIP OVER BLANK
1650        STY LINNUM
1660 .4     LDY #0       LENGTH OF B$
1670        STA (DPTRB),Y
1680        SEC
1690        LDA (DPTRA),Y     LENGTH OF A$
1700        SBC LINNUM
1710        STA (DPTRA),Y
1720        INY
1730        CLC
1740        LDA (DPTRA),Y
1750        ADC LINNUM
1760        STA (DPTRA),Y
1770        INY
1780        LDA (DPTRA),Y
1790        ADC #0
1800        STA (DPTRA),Y
1810        RTS
1820 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8212:DOS3.3:S.StrArrayClear.txt
==============================================================================

 1000  *SAVE S.STRING ARRAY CLEAR
 1010  *--------------------------------
 1020  *      CLEAR STRING ARRAY
 1030  *--------------------------------
 1040  GETARYPT   .EQ $F7D9
 1050  CHKSTR     .EQ $DD6C
 1060  CHKCOM     .EQ $DEBE
 1070  *--------------------------------
 1080  FAC    .EQ $9D
 1090  LOWTR  .EQ $9B,9C
 1100         .OR $300
 1110  *--------------------------------
 1120  CLEAR
 1130         JSR CHKCOM
 1140         JSR GETARYPT
 1150         JSR CHKSTR
 1160         LDY #4       COMPUTE SIZE OF PREAMBLE
 1170         LDA (LOWTR),Y     # OF DIMS
 1180         ASL               *2, CLEAR CARRY
 1190         ADC #5            +5
 1200         STA FAC      SAVE PREAMBLE SIZE
 1210         LDY #2       POINT AT OFFSET
 1220         SEC          COMPUTE ARRAY LENGTH
 1230         LDA (LOWTR),Y
 1240         SBC FAC
 1250         PHA          # BYTES IN PARTIAL PAGE
 1260         INY
 1270         LDA (LOWTR),Y
 1280         SBC #0
 1290         TAX          # WHOLE PAGES
 1300         CLC          POINT AT BEGINNING OF DATA
 1310         LDA LOWTR
 1320         ADC FAC
 1330         STA LOWTR
 1340         BNE .2
 1350         INC LOWTR+1
 1360  .2     LDY #0
 1370         TXA          # WHOLE PAGES
 1380         BEQ .4
 1390         TYA          SET A=0
 1400  .3     STA (LOWTR),Y
 1410         DEY
 1420         BNE .3
 1430         INC LOWTR+1
 1440         DEX          COUNT WHOLE PAGES
 1450         BNE .3
 1460  .4     PLA
 1470         BEQ .6       FINISHED
 1480         TAY
```

```
1490          LDA #0
1500  .5      DEY
1510          STA (LOWTR),Y
1520          BNE .5
1530  .6      RTS
```

```
=============================================================================
DOCUMENT :AAL-8212:DOS3.3:Test.Split.txt
=============================================================================
```

```
d∑1014,0:∑1015,3-x∑N»5∑40ø3:∑1000:∑:∑ÊÂA$»"NOW IS THE TIME FOR ALL
GOOD MEN TO COME TO THE AID OF THEIR PARTY.  WE HOPE THAT ALL OUR
CUSTOMERS ARE DISAPPOINTED ... THAT THEY WAITED SO LONG BEFORE THEY
BOUGHT S-C SOFTWARE."ıÍ∑,A$,B$,N˘Ô∑B$    ˙∑A$…«""∏1005       ∑    \∑
```

===============================================================================
DOCUMENT :AAL-8212:DOS3.3:Test.StrArrClr.txt
===============================================================================

```
dÜA$(25),B$(5)-iÅI–0¡5:B$(I)–"ABCD":ÇGnÅI–0¡25:N–”(¤(1) 5)»1ÅxÅJ–
1¡N:A$(I)–
A$(I)»Á(¤(1) 26»65):Ç:∫„(A$(I))":"A$(I),ãÇÇ:∫:∫óåå768,A$∏ñÅI –
0¡25:∫„(A$(I))":"A$(I),:Ç¿õ∫:∫’†ÅI–0¡5:∫B$(I),:Çfi™´110
```

```
=====================================================================
DOCUMENT :AAL-8301:Articles:Amper.Review.txt
=====================================================================
```

Amper-Magic, The Routine Machine, and Amperware

A Comparative Review......................Bob Sander-Cederlof

I have put off doing this for a long time. The authors and publishers of all three of these programs are friends of mine, and I don't like to go around comparing friends. On the other hand, all of you are my friends, and you have asked for my honest evaluations.

About two years ago, Bob Nacon visited me with an early version of Amper-Magic. He wanted me to consider marketing it for him. I wasn't ready at the time to market anything new, so I suggested he try Roger Wagner (Southwestern Data Systems) and Michael Heckman (then Aurora, now Anthro-Digital Software). Bob went to Roger, and within the same week Peter Meyer came to Roger with his package called "The Routine Machine". Roger opted for Peter's, and Mike took Bob's. Amper-Magic has been available in stores now for about a year. The Routine Machine took longer; I just got a review copy of the final release a few weeks ago.

About three months ago Amper-Ware appeared, and I received a pre-release copy for review. Since then about half a dozen more similar programs have been advertised, and some of them are actually available. Some of them sound very attractive, and I look forward to trying them out. The advertising copy for ELF IV from Sierra On-Line is particularly seductive.
 I would like to hear from you readers any comments you have on any of these Applesoft extension systems.

Both Amper-Magic (&-M) and The Routine Machine (TRM) serve the same function, which is to provide a convenient method for using machine language subroutines with Applesoft programs. Both use the same techniques, and both require the same "run-anywhere" subroutines. Very few if any changes must be made to a subroutine from one package to make it work with the other package.

Amperware (AMW) is a different breed. It is a fixed set of powerful &-subroutines which can be either loaded above the DOS buffers or below the Applesoft program.

TRM uses more memory, provides more features, has a better manual; &-M is more memory-efficient, includes the essential features, and costs $10 more. AMW, least expensive of the three, has some powerful abilities missing from the others.

```
The Package:
       &-M     8.5" x 11" Report Cover
               51 page manual
               Card summarizing operating procedures
```

Diskette not copy-protected
$75

TRM 6" x 9" Padded 3-ring binder
162 page manual
Summary of Subroutine Calls
Diskette copy-protected, but will self-copy up to three times. Locksmith with default parameters makes good copies. Each individual file is copyable to any other diskette.
$64.95

AMW 5.5" x 8.5" booklet, plastic-comb binding
68 page manual
Reference folder
Diskette not copy protected
$49.95

The Subroutine Management System: This is the main program for TRM and &-M, which does the work of appending your selection of machine language subroutines to an Applesoft program. AMW has no such program, being a fixed set of subroutines which are not relocatable.

| Menu Function | &-M | TRM |
|---|---|---|
| Add a subroutine | yes | yes |
| Remove a subroutine | yes | yes |
| Remove all subroutines | no | yes |
| Remove other code | N/A | yes |
| Save appended code | yes | yes |
| Load saved code | yes | yes |
| Report current subrs | yes | yes |
| Search for CALLs | no | yes |
| Search for &s | no | yes |
| Inspect A/S line | no | yes |
| Inspect subroutine | yes | no |
| Display memory map | yes | yes |
| Exit | yes | yes |
| | | |
| Restart after Exit | no | yes |

Subroutines Included in Package:

| | Number of Bytes | | |
|---|---|---|---|
| Function | &-M | TRM | AMW |
| Binary File Info | 253 | 443 | no |
| Delete Array | 47 | no | yes |
| Disassemble | 39 | no | no |
| Dump Variables | 80 | no | no |
| Find Substring | 120 | 140 | yes |
| Find Substr in Array | 285 | 456 | yes |
| GOTO expr | 43 | 17 | yes |

| | | | |
|---|---|---|---|
| GOSUB expr | 35 | 32 | yes |
| Hex Memory Dump | 26 | no | no |
| Input Anything | 92 | 41 | yes |
| Input Form Editor | no | no | yes |
| Screen Control | no | no | yes |
| Move Memory | 147 | 248 | no |
| Poke a List | 25 | no | no |
| Poke hex list | 70 | no | no |
| Print Hex | 26 | no | no |
| Formatted String Print | 380 | no | no |
| Formatted Number Print | no | 261 | yes |
| Print w/o word break | 118 | 197 | no |
| Prune String | 121 | no | no |
| Restore DATA line | 23 | 49 | no |
| Speed up Applesoft: | | | |
| &SPEED=SAVE | 28 | no | no |
| &SPEED=RESTORE | 15 | no | no |
| POKE two bytes | 29 | 150 | no |
| PEEK two bytes | no | 156 | no |
| Swap two variables | 58 | 58 | no |
| Sort string array | no | 250 | yes |
| Sort any array | no | no | yes |
| Sort array with index | no | no | yes |
| Tone (pitch, duration) | no | 56 | no |
| Sound effects | no | 28 | no |
| Fix Link Fields | no | 66 | no |
| ONERR Correction | no | 63 | no |
| Print Error Message | no | 150 | no |
| Restore DATA element | no | 129 | no |
| Convert hex/dec values | no | 214 | no |
| Restore Ampersand | no | 18 | no |
| Hires ASCII shapes | no | 1190 | no |
| Turtle Graphics | no | 612 | no |
| Turtle Graphics Plus | no | 988 | no |
| Super Fast BLOAD | no | 597 | no |
| Fast general disk I/O | no | no | yes |
| RESET vector = normal | no | 16 | no |
| RESET vector = ONERR | no | 42 | no |
| RESET vector = RUN | no | 24 | no |
| RESET vector = re-boot | no | 6 | no |
| Free Sector Count | no | 116 | yes |

Separate utilities on disk:

Amper-Magic

The Routine Machine
- Shape Table Gobbler
- Shape Table Viewer
- Binary File Copier
- Back-up Disk Copier

If you have a serious need for Amper-Magic or The Routine Machine, you might actually find it worthwile to buy both. The price is about equivalent to an hour or two of your own time, but your are getting hundreds of hours of Peter's and Bob's time in return.  Study the code in the machine language subroutines provided with &-M and TRM, until you understand how they work.

Once you have mastered the technique of writing "run-anywhere" subroutines that interact properly with Applesoft, both &-M and TRM are equally valuable tools for managing an ever-growing library of pre-coded modules.  Your own subroutines, together with Peter's and Bob's, and all the ones you find in AAL, Nibble, Call APPLE, etc., become the "IC's" of the programmer's world.

```
==============================================================================
DOCUMENT :AAL-8301:Articles:Arranger.Addtns.txt
==============================================================================
```

An Addition to CATALOG ARRANGER.................Dave Barkovitch

I really like Bill Morgan's CATALOG ARRANGER, from the October issue of AAL. There is something I want to change, though.

When you move the cursor to the end of a long catalog, the cursor stays in the middle of the screen and the catalog scrolls up, until only the top half of the screen is filled. Here are some patches to make the cursor move down to the end, and keep 22 files on the screen:

```
2931        LDA NUMBER.OF.ELEMENTS
2932        SEC
2933        SBC #LINE.COUNT
2934        BPL .5
2935        LDA #ZERO
2936 .5     STA LAST.ELEMENT

3830        BPL .7

3841        BEQ .1
3842 .7     CMP LAST.ELEMENT
3843        BCC .1
3844        LDA LAST.ELEMENT

5991 LAST.ELEMENT .BS 1
```

And Another Change.................................Bill Collins

CATALOG ARRANGER is a great utility. Here are a couple of things you might like to know:

1. Version 4.0 of the S-C Assembler will not accept division in the operand. If you have that version then change line 3820 to SBC #11.

2. If you have DOS relocated into a RAM card you need to add the following lines for bank switching purposes:

```
1165 MONREAD .EQ $C082
1167 DOSREAD .EQ $C083
```

Then add BIT MONREAD at these positions: Lines 1675, 3785, 3855, 3895, 4015 (".5" moved to this line), 4205 (".3" moved to this line), 4315, 4425, 4455 (".7" moved to this line).

And add BIT DOSREAD at these spots: Lines 1535-36, 1685-86, 3795-96, 3905-06, 3975-76, 4035-36, 4215-16, 4345-46, 4465-66, 4955-56.

Also, all DOS addresses must be moved up 16K (lines 1180-1310.) $Axxx addresses become $Exxx and $Bxxx become $Fxxx.

================================================================================
DOCUMENT :AAL-8301:Articles:Cookbook.Review.txt
================================================================================

Micro Cookbook, vol. 1 (Review).....................Bill Morgan

Here are some more details about Don Lancaster's other new book, "Micro Cookbook, vol. 1 -- Fundamentals." As I said last month, the focus of the book is what to learn and how to learn it. He emphasizes "what actually gets used", rather than an exhaustive coverage of all possibilities.

The best quick description of the book is an excerpt from the Preface:

Our aim is to show you how micros work, and how you can profit from and enjoy the micro revolution.

We start with the power and the underlying idea behind all micros. From there we build up the framework for all the important micro concepts and terms. The micro-
processor families are broken down into three simple and easily understood schools.

Chapter Two starts with a set of rules for winning the micro game. These rules have been thoroughly tested in the real world and are not at all what you might expect. After that, we check into many of the resources that are available to you as a micro user. A survey of micro trainers is included.

The Funny Numbers section (Chapter 3) shows you ways to use and understand the number systems involved in micros, particularly binary and hexadecimal. From there, we look at logic, both as hardware gates and as software commands.

The fourth chapter is all about codes. The important codes that are covered include straight binary, 2's complement binary, ASCII, BCD, instruction codes, user port codes, and various serial data-transmission codes and standards. The 2's complement codings are presented in a new and understandable way.

Chapter 5 tells us many things about memory. We go into electronic memory -- beginning with simple latches and progressing to clocked flip-flops. Mainstream microcomputer memory is attacked next -- from static RAMs up through dynamic RAM, ROM, PROM, EPROM, and EEPROM memories.

"Micro Cookbook -- Fundamentals" is just that: Fundamental. I am a complete novice on hardware. After reading Lancaster's book, I still can't design custom interfaces for my Apple, but I can now read the more technical books without getting totally lost. I have a better understanding of address decoding and of what the memory chips are really doing. The book is informative, enlightening, and entertaining. I recommend it.

This Cookbook is about 360 pages of text, plus appendices and index. There are many drawings and charts. List price is $15.95. We will be selling it for $15.00 + postage.

================================================================
DOCUMENT :AAL-8301:Articles:CROSS.AD.txt
================================================================

S-C Macro Cross Assemblers
The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

| | | | |
|---|---|---|---|
| Motorola: | 6800/6801/6802 | now | $32.50 |
| | 6805 | now | $32.50 |
| | 6809 | now | $32.50 |
| | 68000 | now | $50 |
| Intel: | 8048 | now | $32.50 |
| | 8051 | soon | $32.50 |
| | 8085 | soon | $32.50 |
| Zilog: | Z-80 | now | $32.50 |
| RCA: | 1802/1805 | soon | $32.50 |
| Rockwell: | 65C02 | now | $20 |

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support:  competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation   (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

=====================================================================
DOCUMENT :AAL-8301:Articles:Filename.Editor.txt
=====================================================================

A Filename Editor for CATALOG ARRANGER.............Bill Morgan

Many thanks to all of you who have called and written to say how much you like the CATALOG ARRANGER. I'm glad to hear that others find it as useful as I do. Here's my favorite addition to the program, the ability to edit the filename in the cursor. Now you can change a name by inserting or deleting characters, insert control characters, and place display titles in the catalog, using normal, inverse, flashing, or lower case text.

There are a couple of unique features in this editor. The cursor clearly indicates Insert, Overtype, or Override mode, and also shows whether the input will be Normal, Inverse, Flashing, or Lower Case. The display unambiguously shows all these types, plus Control. The price of all this clarity is three display lines for one text line, but that's no problem in this program. These concepts can easily be adapted to edit any line of forty or fewer characters. The principles also apply to longer lines, but the screen display would have to be handled carefully.

Installation

To add FILENAME EDITOR to CATALOG ARRANGER just type in S.FILENAME.EDITOR from this listing, and save it on the same disk with S.CATALOG.ARRANGER. Then LOAD S.CATALOG.ARRANGER and make the following changes and additions:

```
1030        .TF CATALOG.ARRANGER.NEW

1480 LINE.COUNT   .EQ 21

1915        CMP #$85    ^E
1920        BNE .1
1922        JSR RENAME.FILE
1924        JMP DISPLAY.AND.READ.KEY

5865        .IN S.FILENAME.EDITOR
```

Then SAVE the new S.CATALOG.ARRANGER and assemble it.

Operation

To rename a file, just use the arrow keys to move the cursor to the file you want, and type "CTRL-E" (for Edit). The name you selected will appear near the bottom of the screen, between square brackets. Any control characters in the name will have a bar above them. The caret below the first character of the name is the cursor. Any non-control characters you type will replace the characters on the screen. Control characters will have the effects shown in the command list below. Especially note that RETURN will enter the name in the lower buffer into the filename array, ESC will return you to the Arranger without altering the filename, and CTRL-R will restore the original filename.

One way to have fun with this program is to put dummy files in the catalog, for titles or just for decoration. In Applesoft, SAVE as many dummy programs (10 REM, for example) as you need. Then BRUN CATALOG ARRANGER, move the dummy programs to where you want them, and edit the names. If you start the new file name with six CTRL-H's, it will blank out the "A 002 " before the name. You can use inverse, flashing or lower case text in titles. If you insert CTRL-M's (RETURNS) after a name there will be blank lines in the catalog. Play with it for a while, and let me know if you come up with any especially neat tricks.

Here are the commands:

<-- -- Left Arrow. Move the cursor left one position.
--> -- Right Arrow. Move the cursor right one position.
RETURN -- Enter. Enter the changed name into the upper display.
ESC -- Escape. Return to arranging, without entering the changed name.
^B -- Beginning. Move the cursor to the beginning of the line.
^D -- Delete. Delete one character at the cursor.
^E -- End. Move the cursor to the end of the name.
^F -- Find. Move the cursor to a particular character. Type "^FA" to move the cursor to the next "A" in the name. Type another "A" to move to the following "A", and so on. Any character other than the search key will be entered or executed.
^I -- Insert. Turn on Insert Mode. Following characters will be inserted to the left of the backslash cursor. Any control character turns Insert off.
^O -- Override. Insert the next character typed "as is". This allows you to insert control characters into a name.
^R -- Restore. Restore the name to its original condition, as it appears in the upper display.
^S -- Shift Mode. Cycle between Normal, Inverse, Flashing, and Lower Case entry. The cursor changes to show the current mode.
^Z -- Zap. Remove all characters from the cursor to the end of the name.

How it All Works

When you type CTRL-E to enter the editor, line 1090 transfers the filename into an edit buffer located in the screen memory at $757-$774. The main loop of the editor is lines 1190-1320. All through the editor the Y-register is the cursor position in the line. The routine DISPLAY.EDIT.BUFFER shows the brackets before and after the name, puts bars over any control characters, displays the cursor, and gets the next keystroke. The main loop then checks whether that key was a control.

If it was not a control character, it is passed to the input section (lines 1340-1570), where the character is masked according to the current MASK.MODE (Normal, Inverse, Flashing, or Lower-case) and either inserted or just placed in the line.  The program then jumps back to E.START to redisplay the buffer and get the next key.

If you enter a control character, the program JSR's to the SEARCH.AND.PERFORM routine at lines 3250-3390 (taken straight from Bob's article in the August '82 AAL.)  Here we look up the command key in the table at lines 3420-3550 and do a PHA, PHA, RTS type branch to the appropriate command handler, or to the monitor's BELL, if the command didn't match anything in the table.

Almost all of the command handlers end with an RTS that returns control to line 1320.  The exceptions are OVERRIDE (lines 1590-1650) and RESTORE (lines 2150-2180), since they exit through internal JMP's, and RETURN/ESC (lines 2660-2720), since those return to the main program.  Another oddity is the FIND routine (lines 2420-2640), since it has two exits.  Line 2640 returns to line 1320 through the BELL routine.  Lines 2590-2620 are needed to process a keystroke that is not a repetition of the search key.

```
==============================================================================
DOCUMENT :AAL-8301:Articles:Front.Page.txt
==============================================================================
```

$1.50

Volume 3 -- Issue 4                    January, 1983

In This Issue...



Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

================================================================================
DOCUMENT :AAL-8301:Articles:Hardcore.Mag.txt
================================================================================

Still More on Hardcore Magazine............Bob Sander-Cederlof

I bought the latest, Vol. 1 No. 3, off the newsstand a few days ago. It is 72 pages, $3.50. I believe those 72 pages far surpass in usefulness the 600-odd pages of some familiar monthlies. A highlight for me was a complete assembly listing (in S-C format!) of HyperDOS, by John Bridges.

HyperDOS modifies the LOAD and BLOAD commands so that loading runs up to five times faster. This is the same improvement factor offered by a half dozen DOS-mods on the market, such as DOS Enhancer from S&H Software. (Of course, DOS Enhancer also speeds up SAVE and BSAVE, and include many other useful utilities with the package.)

If you are a nibble copier, you will be pleased with the listing of parameters for Locksmith and Nibbles Away II. As usual, there are a lot of hints on "how to unlock" those copy-protected disks: see "Controlling the I.O.B.", and "Boot Code Tracing".

Bev Haight (author of "Night Falls", among others) gives some excellent information on graphics, games, and even secrets to publishing. Bev describes, explains, and lists a new game called "Zyphyr Wars" for your pleasure and edification.

There is a lot more. Even an interview with Mike Markulla regarding Apple's position on software protection!

Issue number 4 promises to focus on graphics: novice-to-expert how-to's, complete graphic aid programs, tables, charts, reviews, etc.

=============================================================================
DOCUMENT :AAL-8301:Articles:Last.Minute.txt
=============================================================================

Many of you have expressed an interest in the new Rockwell R65C02 microprocessor. Well, we still haven't heard any more than I mentioned last month. We're as eager as you are to get a sample. We'll have a detailed report as soon as we know more.

Peter Bartlett just called from Chicago to report an unpublished ceiling on the number of Target Files that can be generated by one assembly. There can be only 32. If you need more files than that you should be able to patch $29EA from $1F to $3F. We haven't had time to test this thoroughly yet, so we'll tell you more next month.

```
=========================================================================
DOCUMENT :AAL-8301:Articles:My.Ad.txt
=========================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
Upgrade from Version 4.0 to MACRO.................................$27.50
Source code of Version 4.0 on disk................................$95.00
    Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// (coming soon!)...........................$???.00

Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.

ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).................. $79.00
Source Code for FLASH! Runtime Package............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware)......................$45.00
Apple White Line Trace (Lone Star Industrial Computing)...........$50.00
    (A unique learning tool)

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $4.50
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                                100 or more:  25 cents each

Ashby Shift-Key Mod...............................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
     Only Revision level 7 or later Apples.

Books, Books, Books..........................compare our discount prices!
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
```

```
"What's Where Guide" (updates first edition)..........($9.95)   $9.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

 Add $1 per book for US postage.  Foreign orders add postage needed.

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

================================================================================
DOCUMENT :AAL-8301:Articles:New.Hardware.txt
================================================================================

A Plug for some Neat New Products..............Richard Fabbri
Ridgefield, CT

Take a peek at BYTE Magazine, August 1982, Steve Ciarcia's article on the TMS9918-based Color graphics for Apple II. It has proved to be fantastic! You get 15 colors plus transparent, 32 sprite planes to overlay a 15-color hi-res of 256 dots by 192 lines. It works as advertised! Digital Dimensions (see BYTE, Nov 1982, page 352) advertises this as "E-Z Color" board, for $230. I have had one now for a month.

Two other neat new board for the Apple from Number Nine Computer Engineering Inc:

* a graphics board with 1024x1024 resolution; 256 colors from a palette of 4096; HARDWARE drawing of circles, arcs, rectangles and vectors; characters; area fill; light pen interface; $750 to $1090, depending on options.

* a processor card with 3.6 MHz 6502, 64K on-board high-speed RAM, transparent execution of all Apple II software, software-controlled speed for timed I/O operations; $745.

If you are interested: contact Number Nine at (203) 233-8134, or P.O.Box 1802, Hartford, CT 06144.

=============================================================================
DOCUMENT :AAL-8301:Articles:QD9.COVER.txt
=============================================================================

QUARTERLY DISK #9 contains all the source code from Volume 3, Issues 1-3 of the Apple Assembly Line newsletter. The files are formatted for either the S-C Assembler II Version 4.0 or the S-C Macro Assembler, on a 16-sector DOS 3.3 disk.

S.CATALOG ARRANGER -- This program allows you to arrange a disk catalog in whatever order you like.

S.USR WEEK FUNCTION, TEST USR -- Applesoft USR function for a two-byte PEEK.

S.TOOLKIT CONVERTER, TOOLKIT CONVERTER -- EXEC file to help convert DOS TOOLKIT source files to S-C format.

S.NEW APPLE TALKER -- Your Apple speaks! A program to record and play back sounds with no additional hardware.

TALK -- THIS IS A TEST, SOUND#1 through SOUND#5 -- Speech and sound/sight samples for the above program.

S.TONY'S SUPER-FAST PRIMES, TONY'S DRIVER -- The fastest prime number generator yet.

S.LOCATOR -- Program to display addresses of an Applesoft program and its data areas.

S.REPEAT/UNTIL, TEST REPEAT/UNTIL -- Ampersand routines to add a Pascal-like REPEAT/UNTIL structure to Applesoft.

S.MEYER'S RELOCATABILITY -- Routines to use internal JMPs, JSRs, and data areas in relocatable programs.

S.BITS -- A Monitor CTRL-Y routine to set or clear selected bits in a range of memory.

S.STRING ARRAY CLEAR, TEST STRING.ARRAY.CLEAR -- Routine to clear an Applesoft string array. Combine this with a fast garbage-collector and watch your program fly.

S.SPLIT, TEST SPLIT -- Splits an Applesoft string to a specified display width, without generating garbage.

=====================================================================
DOCUMENT :AAL-8301:Articles:Quickies.txt
=====================================================================

Quickie No. 5..................................Horst Schneider

To print a dashed line on the screen:

```
      JSR $FD9C     Print one dash
      JSR $FCA3     same character across screen
```

To print any character across screen:

```
      LDY #0
      LDA #$xx      xx = ASCII screen code for char
      JSR $FCA3
```

To print any character across most of screen:

```
      LDY #xx          xx = starting column
      LDA #$yy         yy = ASCII screen code for char
      JSR $FCA3
```

A Legible Phone Number for Computer Micro Works

Their ad last month was a little fuzzy around the area where the phone number was. The correct number is (305) 777-0268. George Beasley or his wife will take your order. This number is in Florida, where George is stationed with the Air Force.

I ordered one of their "Promette's". It is different than I thought, and better. Most such adapters will not work when a language card is in slot 0, because EPROM's are missing one of the enable lines the Apple uses. But the Promette has an active device inside which adds the extra enable line, so it works like you want it to. Another nice difference is that George's price is about 1/4 the normal price for these items.

==============================================================================
DOCUMENT :AAL-8301:Articles:RAM.Cards.txt
==============================================================================

Programming a Language Card..........................Bill Morgan

Recently we've received a couple of questions about the exact meaning of all those $C08x addresses used to access a language (or RAM) card in slot 0. Here's a rundown of what memory cards are and how to use them.

A RAM card is a plug-in board containing an extra 16K (or more) of memory, which can be used instead of the language ROMs on your Apple motherboard. The $C08x addresses are switches that determine which memory will be used whenever you read or write an address from $D000-$FFFF. With the proper use of the switches on a 16K card, your Apple becomes a machine with 76K of memory! (That includes motherboard RAM, motherboard ROM, and the full RAM card.)

Here's a summary of the addresses and their functions:

| Address | Read | Write | Bank |
|---|---|---|---|
| $C080 | Card | Mother | 2 |
| $C081* | Mother | Card | 2 |
| $C082 | Mother | Mother | 2 |
| $C083* | Card | Card | 2 |
| | | | |
| $C088 | Card | Mother | 1 |
| $C089* | Mother | Card | 1 |
| $C08A | Mother | Mother | 1 |
| $C08B* | Card | Card | 1 |

The stars indicate addresses which must be accessed twice to have effect (these are the ones that write-enable the card.)

These addresses are "soft switches", much like those for switching the screen display modes. To throw a switch, just use a LDA or any instruction that reads the location. From BASIC you can use a PEEK. STA or POKE also work with most RAM cards, but not all of them. Experiment with yours to see how it behaves. If you're writing a program for use on other people's Apples it's safest to stay with instructions that read the location.

The Bank column refers to the fact that a language card actually has 16K of memory, but the range from $D000 to $FFFF is only 12K. The other 4K ought to be $C000-$CFFF, but that's the area that Apple uses for special Input/Output functions. Therefore, there is an extra 4K "bank" which can be addressed at $D000-DFFF. Normally, only Bank 2 is used. If a program gets bigger than 12K it becomes necessary to use Bank 1, but that starts getting complicated. The best approach is to put routines or data in bank 1 that don't have to refer to anything in bank 2. You can then have the main code above $E000 decide which bank to use.

Some programs seem to use the motherboard and RAM card memories at the same time. Examples of this are ES-CAPE.LC and the programs that relocate DOS into the RAM card. Generally, these have a short "bridge" or "switcher" routine somewhere in the motherboard RAM. When the program in the RAM card needs to call a routine in the motherboard ROM, it actually calls the bridge. The bridge routine then throws the appropriate $C08x switches and calls the necessary ROM routine. When that routine finishes, the bridge then switches back to the RAM card and continues the program there.

Another thing to consider is whether the program in the RAM card needs the system monitor. If so, you need to make sure there is a copy of the monitor on the RAM card. Here's how to use the monitor to copy itself into a RAM card:

```
]CALL-151
*C081 C081
*F800<F800.FFFFM
```

That monitor move instruction looks like nonsense, but remember that the $C081 switch sets the computer to read from the motherboard and write to the RAM card.

==============================================================================
DOCUMENT :AAL-8301:Articles:S.C.DOCU.MENTOR.txt
==============================================================================

S-C Docu-Mentor:  Applesoft

The S-C Docu-Mentor for Applesoft provides the most complete documentation of Applesoft internals available anywhere.  You will find the information educational, entertaining, and extremely helpful.

The completely commented and ready-to-assemble source code is created with the aid of the Applesoft image in ROM (or on a RAM card).  The source creation process modifies existing data on two diskettes to produce over two dozen linked source files ready to assemble with the S-C Macro Assembler.

The information contained in the comments and labels has been gleaned from many sources over the years since 1978.  (There has been no direct involvement of Apple Computer, Inc., in this project.)  I have tried my best to provide meaningful, helpful comments and labels throughout.  Nevertheless, there may be some mis-interpretations.  If you find any errors or have suggestions for improvements, please send them to me.

Many of the label names are deliberately made the same as those published in "Applesoft Internals", by John Crossley.  This article first appeared in Volume 1, No. 1, of the Apple Orchard (magazine of the International Apple Corps).  It has been re-printed by various clubs, including Call A.P.P.L.E. in their book "All About Applesoft".

I have flagged about a half dozen bugs in the listing, and several areas of very "improve-able" code.  These are marked with "<<<" and ">>>" at each end of the comment lines.

Apple II, Apple II Plus, and Applesoft are registered trademarks of Apple Computer, Inc.

Procedure for creating the documented source files:

1.  You will need an Apple II or Apple II Plus, with Applesoft either in ROM on the mother-board, in ROM on a firmware card, or in RAM on a RAM card.

2.  Use any standard disk copier to copy the two original diskettes. Label the copies, and be sure to mark the labels "Drive 1" and "Drive 2".  Store the originals, and proceed with the following steps using your copies.

3.  Insert either disk and type "RUN HELLO".  If Applesoft is available in ROM or RAM, the source-creation process will begin.  You will see the phrase "S-C DOCU-MENTOR:  APPLESOFT" on the screen, and it will slowly be changed to inverse display.  You will hear various speaker noises, which are just there to let you know things are

working. When the process is finished, the disk CATALOG will be displayed.

4. Insert the other disk and type "RUN HELLO". Once again, you will see the same display and hear the same speaker noises. Upon completion, the CATALOG will be displayed.

Procedure for assembling the source files:

1. You will need two disk drives. The source file "S.ACF" assumes these are D1 and D2 on the same controller, but you may change these according to your configuration. Of course, you will also need a printer if you wish a permanent copy of the assembly listing.

2. Load the S-C Macro Assembler, either regular or language card version.

3. Insert disk # 1 in drive 1, and disk # 2 in drive 2.

4. Load file "S.ACF" from drive 1.

5. Make any adjustment to the title line you wish. I have set it up for printing 76 lines per page, because I set my printer to print 8 lines per inch. If you are using 6 lines per inch, change the title line to ".TI 55,et cetera".

6. Turn on your printer, and type "PR#slot" to start printing. You will probably want to set "elite" or "condensed" printing mode, because some of the lines in the assembly listing will be more than 80 columns long. I set my printer to "elite" mode (12 chars/inch) and set a left margin of 10 spaces.

7. Type the "ASM" command, and stand back! The listing is 114 pages long (including symbol table) when printed at 76 lines per page.

==============================================================
DOCUMENT :AAL-8301:Articles:Seed.Thought.txt
==============================================================

Seed Thoughts on Extensions................Sanford Greenfarb

I am currently between computers. My 4 1/2 year old Apple died and I have ordered a Basis 108 to replace it. While waiting, I have been doing some thinking; I came to the conclusion that I can extend, by appropriate coding, either the monitor or Applesoft (or both) into the unused 4K bank of my 16K RAM card. That second 4K bank at $D000-DFFF is just sitting there, with nothing to do. In all the Apple mags I have seen no one approaches thi idea. Maybe they know something I don't, but as soon as my computer comes I am going to try it.

I suspect that I could insert code at $FF7A in the monitor to switch 4K banks and jump to $D000 for a modified character search subroutine. This way I could add more control characters and routines to the monitor. This would add features while keeping all the standard entry point address unchanged.

I don't know why no one has used this concept, or at least not publicly.  I am offering this idea to you readers of Apple Assembly Line.  I can't work on it until my new computer comes anyway, and you will probably think of a lot of good uses.

==============================================================================
DOCUMENT :AAL-8301:Articles:String.Addition.txt
==============================================================================

Adding Decimal Values from ASCII Strings...Bob Sander-Cederlof

The program below shows a nifty way to add two decimal values together and get the result as an ASCII string, without ever converting decimal to binary or binary to decimal.

The example shows two six-character values being added, but any length would work the same. For simplicity's sake I used a leading zero format, and allow no signs or decimal points. Fancier features can wait for more cerebral times.

The beautiful part is the way the 6502's carry flag works. On entering the add loop, I clear carry. Then I add a pair of digits, preserving the ASCII code. If the sum is more than "9" ($39), the CMP will leave carry set, prepared for subtracting 10 at line 1160. After subtracting 10, carry will be set (because the SBC caused no borrow). This carry then propagates to the next digit.

Strictly speaking, I should allow the sum to be one digit longer than the addend and augend strings, and store the final carry value there. Any reasonably useful version would also allow leading blanks and decimal points, be callable as an &-routine with string parameters, automatically handle non-aligned decimal points, and allow negative numbers. I'll try all these for next month.

==============================================================================
DOCUMENT :AAL-8301:Articles:Super.Scroller.txt
==============================================================================

Super Scroller.................................Jeffrey Scott
East Norwalk, CT

I am a manager of a software department in a company that makes computerized money counting equipment (6502 based). We have two programming departments: one which is called "applications" (Pascal and BASIC only) and another called "software engineering" where we use assembly language.

We use the S-C Macro Assembler after having sampled all others. And in fact, with my Apple II, 5 Mbyte hard drive, and 3.6 MHz "Number Nine 6502" plug-in board, I can assemble a 300-page source program in about 2.5 minutes!

I love the Apple II, but I don't like being tied to an operating system that I didn't write myself. I use RWTS, but for the rest I use my own code.

I remember one day trying to output to the screen while receiving at 2400 baud. The Apple monitor's scroll was so slow that I lost the first few characters from the front of every line. While writing my own substitute scroll routine, the idea was born that the absolute fastest scroll would be straight in-line code: one "LDA $xxxx...STA $xxxx" pair for each byte on the screen.

Just for fun, I wrote the following program, which generates the 960 LDA-STA pairs to scroll the whole screen! The generator program is only 145 bytes long, but it "writes" a program 5521 bytes long!

This "Super Scroller" is not for everyone...it requires a spare 5521 bytes ($1591) of memory somewhere. If you do, you need only equate "PGM.START.IN.RAM" to your available area, call "PGM.TO.WRITE.SCROLLING.PGM", and then you can call the Super Scroller at "PGM.START.IN.RAM whenever you need it.

Since the scroller can be generated whenever it is needed, it can be part of an overlay environment. You only need a 5.5K buffer available at the right times. At other times the same memory can be used other ways.

To illustrate the speediness of Super Scroller, I wrote a memory dump whose output is the same as the Apple monitor memory dump. It is set up to display from $0000 through $BFFF. With Super Scroller, it takes only about 51 seconds; without, it takes 2 minutes 57 seconds (over three times longer!).

Someone might object that I did not clear the bottom line after scrolling up. I elected to just write a fresh bottom line, and clear to the end of line after the last new character is written.

================================================================================
DOCUMENT :AAL-8301:Articles:The.Book.txt
================================================================================

The Book of Apple Software 1983

It's huge!  Nearly 500 pages of insightful reviews and comparison charts, covering business, education, utilities, and games.  The review of seven assemblers includes Merlin, Lisa 2.5, Tool Kit, LJK Edit 6502, MAE, S-C Assembler II (4.0) and S-C Macro Assembler.  S-C Macro tied for first place with Merlin in the overall ratings, but surged ahead in the detail.  Consider:  not copy protected, typeset programmer reference card, cassette support, monitor and DOS commands without leaving assembler, FANTASTIC upgrade policy, RAM card optional, compressed source code, 32 character labels, and more.

Anyway, back to The Book....you owe it to yourself to consult therein before buying software.  Even if the one you want to buy isn't in the book, you will get a broader perspective.  I recommend it.

==================================================================
DOCUMENT :AAL-8301:Articles:V3N4.6801.txt
==================================================================

Funny Opcode Names in the 6801 Manual......Bob Sander-Cederlof

Paul Lundgren (of Microcomp, Inc. in Newtown, CT) brought some interesting facts to my attention today.  When I implemented my 6801 Cross Assemblers, I used what was at the time the latest documentatin available.  Paul had some printed two years later, and there were some differences.

For some reason, the Motorola 6801 Reference Manual changes the name of the ASL and ASLD opcodes to LSL and LSLD.  There is no difference in operation, just a difference in spelling.  The S-C Cross Assembler only recognizes the ASL and ASLD spellings.  The opcode tables are near the end of the assembler, so you can easily find these entries to change them if you feel strongly about it.

The Motorola book also lists alias names for the BCC and BCS opcodes. In the 6801 (or other 68xx chips), carry clear means the last test was greater or equal, so the alias name is BHS (Branch if High or Same). Carry set means the test was smaller, so the alias is BLO.  Note that the meaning of carry after a comparison in the 68xx chips is exactly the opposite of carry in the 6502!

Here are some macros to use for BHS and BLO:

```
       .MA BHS
       BCC ]1
       .EM

       .MA BLO
       BCS ]1
       .EM
```

Some assemblers for the 6502 have two alias opcodes for BCC and BCS. For example, LISA has BLT for BCC (Branch if Less Than), and BGE for BCS (Branch if Greater than or Equal).  [ I didn't do this in the S-C Assemblers because the meaning depends on whether the values tested are considered to be signed or unsigned. ]

Here are two macros to implement BLT and BGE in the 6502 version of the S-C Macro Assembler:

```
       .MA BLT
       BCC ]1
       .EM

       .MA BGE
       BCS ]1
       .EM
```

===============================================================================
DOCUMENT :AAL-8301:Articles:Whats.Where.txt
===============================================================================

The New "What's Where".....................Bob Sander-Cederlof

Micro has doubled the size and tripled the value of their "What's Where in the Apple" book. There is now a 152-page double-column type-set 20-chapter text together with the previously published atlas and gazetteer. The new edition retails at $24.95 (our price $23).

If you already have the older edition, you only need the update, called "The Guide to What's Where", for $9.95 retail (our price ($9 even).

If you order books from us, remember to include enough for shipping.

```
==============================================================================
DOCUMENT :AAL-8301:DOS3.3:S.Fname.Editor.txt
==============================================================================

 1000  *SAVE S.FILENAME.EDITOR
 1010  *---------------------------------
 1020  MON.YSAVE    .EQ $34
 1030  CONTROL.LINE .EQ $6D7
 1040  EDIT.BUFFER  .EQ $757
 1050  CURSOR.LINE  .EQ $7D7
 1060  MON.BELL     .EQ $FF3A
 1070  *---------------------------------
 1080  RENAME.FILE
 1090         JSR MOVE.FILE.INTO.BUFFER
 1100         LDY #$FF
 1110         STY MASK.ONE      INITIALIZE
 1120         INY
 1130         STY INPUT.MODE    VARIABLES
 1140         STY MASK.TWO
 1150         STY MASK.MODE
 1160         LDA #$DE          ^
 1170         STA CURSOR
 1180  *---------------------------------
 1190  E.START
 1200         JSR DISPLAY.EDIT.BUFFER  UPDATE DISPLAY
 1210  *                               AND GET KEYSTROKE
 1220
 1230  REENTRY
 1240         CMP #$A0                 CONTROL?
 1250         BCS E.INPUT              NO, INPUT IT
 1260         LDA #ZERO                YES,
 1270         STA INPUT.MODE           TURN OFF INSERT
 1280         LDA #$DE                 ^
 1290         STA CURSOR
 1300         LDX #ZERO
 1310         JSR SEARCH.AND.PERFORM   GO DO SOMETHING
 1320         JMP E.START
 1330  *---------------------------------
 1340  E.INPUT
 1350         AND MASK.ONE         CONDITION
 1360         ORA MASK.TWO         CHARACTER
 1370         STA CURRENT.CHAR
 1380         BIT INPUT.MODE       INSERT OR OVERTYPE?
 1390         BPL PLACE.CHARACTER
 1400
 1410  INSERT.CHARACTER
 1420         STY MON.YSAVE        SAVE CURSOR
 1430         LDX #29              START AT END OF BUFFER
 1440  .1     CPX MON.YSAVE        TO CURSOR YET?
 1450         BEQ PLACE.CHARACTER  YES
 1460         LDA EDIT.BUFFER-1,X  NO, MOVE CHAR UP
 1470         STA EDIT.BUFFER,X    TO MAKE HOLE
 1480         DEX                  NEXT CHAR
```

```
1490        BPL .1           ...ALWAYS
1500
1510 PLACE.CHARACTER
1520        LDA CURRENT.CHAR
1530        STA EDIT.BUFFER,Y
1540        CPY #29             END OF BUFFER?
1550        BCS .1              YES, RETURN
1560        INY                 NO, MOVE CURSOR
1570 .1     JMP E.START
1580 *--------------------------------
1590 E.OVERRIDE
1600        PLA
1610        PLA
1620        LDA #$A2            SET CURSOR
1630        STA CURSOR.LINE,Y   TO "
1640        JSR GETKEY
1650        JMP INSERT.CHARACTER
1660 *--------------------------------
1670 E.LEFT.ARROW
1680        DEY          MOVE CURSOR LEFT
1690        BPL .1       IF IT WENT NEGATIVE
1700        INY          RESTORE IT TO 0
1710 .1     RTS
1720 *--------------------------------
1730 E.RIGHT.ARROW
1740        CPY #29      AT END YET?
1750        BCS .1       YES, IGNORE
1760        INY          NO, MOVE CURSOR RIGHT
1770 .1     RTS
1780 *--------------------------------
1790 E.INSERT
1800        LDA #$FF        TURN INSERT ON
1810        STA INPUT.MODE
1820        LDA #$DC        \
1830        STA CURSOR
1840        RTS
1850 *--------------------------------
1860 E.DELETE
1870        TYA               SET X TO
1880        TAX               CURSOR
1890 .1     CPX #29           AT END?
1900        BEQ .2            BRANCH IF SO
1910        LDA EDIT.BUFFER+1,X
1920        STA EDIT.BUFFER,X MOVE ONE CHAR
1930        INX               NEXT
1940        BCC .1         ...ALWAYS
1950 .2     LDA #SPACE        PUT SPACE
1960        STA EDIT.BUFFER,X ON END
1970        RTS
1980 *--------------------------------
1990 E.BEGINNING
2000        LDY #ZERO         ZERO CURSOR
2010        RTS
2020 *--------------------------------
```

```
2030  E.END
2040         LDY #29            START AT END OF BUFFER
2050  .1     LDA EDIT.BUFFER,Y
2060         CMP #SPACE         SPACE?
2070         BNE .2             NO, WE'RE AT END OF NAME
2080         DEY                YES, MOVE LEFT
2090         BPL .1             AND TRY AGAIN
2100  .2     CPY #29            STILL AT END OF BUFFER?
2110         BEQ .3             YES, STAY THERE
2120         INY                NO, RIGHT ONE SPACE
2130  .3     RTS
2140  *--------------------------------
2150  E.RESTORE
2160         PLA                POP A RETURN
2170         PLA                ADDRESS AND
2180         JMP RENAME.FILE    START OVER
2190  *--------------------------------
2200  E.SET.MODE
2210         INC MASK.MODE         NEXT MODE
2220         LDA MASK.MODE         IF MODE = 4
2230         AND #3                MAKE IT ZERO
2240         STA MASK.MODE
2250         TAX                   USE MODE FOR INDEX
2260         LDA MASK.ONE.TABLE,X AND SET
2270         STA MASK.ONE          MASKS
2280         LDA MASK.TWO.TABLE,X
2290         STA MASK.TWO
2300         RTS
2310  *--------------------------------
2320  E.ZAP
2330         TYA                START AT
2340         TAX                CURSOR
2350         LDA #SPACE
2360  .1     STA EDIT.BUFFER,X
2370         INX
2380         CPX #30            DONE?
2390         BCC .1
2400         RTS
2410  *--------------------------------
2420  E.FIND
2430         JSR GETKEY         GET SEARCH KEY
2440         STA SEARCH.KEY
2450  .1     TYA
2460         TAX
2470  .2     INX                START AT CURSOR+1
2480         CPX #30            END?
2490         BCS .3             YES, NOT FOUND
2500         LDA EDIT.BUFFER,X
2510         CMP SEARCH.KEY     MATCH?
2520         BNE .2             NO, NEXT X
2530         TXA                YES, MOVE CURSOR
2540         TAY
2550         JSR DISPLAY.EDIT.BUFFER
2560  *                         NEXT KEYPRESS
```

```
2570        CMP SEARCH.KEY    SAME CHARACTER?
2580        BEQ .1            YES, FIND IT AGAIN
2590        PLA               NO, PULL A RETURN
2600        PLA               ADDRESS AND GO
2610        LDA CURRENT.CHAR
2620        JMP REENTRY       PROCESS THIS KEY
2630
2640 .3     JMP MON.BELL      RETURN THROUGH BELL
2650 *--------------------------------
2660 E.RETURN
2670        JSR MOVE.BUFFER.INTO.ARRAY
2680
2690 E.ESCAPE
2700        PLA               POP ONE RETURN
2710        PLA               ADDRESS AND RETURN
2720        RTS               TO ARRANGING
2730 *--------------------------------
2740 MOVE.FILE.INTO.BUFFER
2750        LDA ACTIVE.ELEMENT  SET
2760        JSR POINT.TO.A      POINTER
2770        LDY #3
2780 .1     LDA (POINTER),Y     MOVE
2790        STA EDIT.BUFFER-3,Y NAME
2800        INY
2810        CPY #$21
2820        BCC .1
2830        RTS
2840 *--------------------------------
2850 MOVE.BUFFER.INTO.ARRAY
2860        LDA ACTIVE.ELEMENT  MAKE
2870        JSR POINT.TO.A      POINTER
2880        LDY #3
2890 .1     LDA EDIT.BUFFER-3,Y MOVE
2900        STA (POINTER),Y     NAME
2910        INY
2920        CPY #$21
2930        BCC .1
2940        RTS
2950 *--------------------------------
2960 DISPLAY.EDIT.BUFFER
2970        LDA #$DD           ]
2980        STA EDIT.BUFFER-1  LEFT END
2990        LDA #$DB           [
3000        STA EDIT.BUFFER+30 RIGHT END
3010        LDX #29
3020 .1     LDA #SPACE
3030        STA CONTROL.LINE,X REMOVE OLD CONTROL
3040        STA CURSOR.LINE,X  BAR AND CURSOR
3050        LDA EDIT.BUFFER,X
3060        CMP #$A0
3070        BCS .2             CONTROL?
3080        CMP #$80
3090        BCC .2
3100        LDA #$DF    _      YES, PUT BAR
```

```
3110        STA CONTROL.LINE,X
3120 .2     DEX
3130        BPL .1
3140        LDA CURSOR        GET CURSOR,
3150        AND MASK.ONE      CONDITION IT,
3160        ORA MASK.TWO
3170        STA CURSOR.LINE,Y  AND SHOW IT
3180 *--------------------------------
3190 GETKEY LDA KEYBOARD
3200        BPL GETKEY
3210        STA KEYSTROBE
3220        STA CURRENT.CHAR
3230        RTS
3240 *--------------------------------
3250 SEARCH.AND.PERFORM.NEXT
3260        INX                NEXT ENTRY
3270        INX
3280        INX
3290
3300 SEARCH.AND.PERFORM
3310        LDA EDIT.TABLE,X   GET VALUE FROM TABLE
3320        BEQ .1             NOT IN TABLE
3330        CMP CURRENT.CHAR
3340        BNE SEARCH.AND.PERFORM.NEXT
3350 .1     LDA EDIT.TABLE+2,X  LOW BYTE OF ADDRESS
3360        PHA
3370        LDA EDIT.TABLE+1,X  HIGH BYTE
3380        PHA
3390        RTS                GO DO IT!
3400 *--------------------------------
3410 EDIT.TABLE
3420        .DA #$82,E.BEGINNING-1     ^B
3430        .DA #$84,E.DELETE-1        ^D
3440        .DA #$85,E.END-1           ^E
3450        .DA #$86,E.FIND-1          ^F
3460        .DA #$88,E.LEFT.ARROW-1   <--
3470        .DA #$89,E.INSERT-1        ^I
3480        .DA #$8D,E.RETURN-1    RETURN
3490        .DA #$8F,E.OVERRIDE-1      ^O
3500        .DA #$92,E.RESTORE-1       ^R
3510        .DA #$93,E.SET.MODE-1      ^S
3520        .DA #$95,E.RIGHT.ARROW-1  -->
3530        .DA #$9A,E.ZAP-1           ^Z
3540        .DA #$9B,E.ESCAPE-1       ESC
3550        .DA #$00,MON.BELL-1    OTHERS
3560 *--------------------------------
3570 MASK.ONE.TABLE
3580        .DA #$FF,#$3F,#$7F,#$FF
3590
3600 MASK.TWO.TABLE
3610        .DA #$00,#$00,#$40,#$20
3620 *--------------------------------
3630 CURRENT.CHAR .BS 1
3640 SEARCH.KEY   .BS 1
```

```
3650  INPUT.MODE   .BS 1 (0 OR $FF)
3660  MASK.MODE    .BS 1 (0 TO 3)
3670  MASK.ONE     .BS 1 (FROM TABLE ABOVE
3680  MASK.TWO     .BS 1 (  "    "     "  )
3690  CURSOR       .BS 1 ($DE, $DC, OR $A2)
3700  *                  ( ^ ,  \ , OR  " )
3710  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8301:DOS3.3:S.STRING.ADD.txt
==============================================================================

 1000  *SAVE S.STRING.ADD
 1010  *-------------------------------
 1020  *      STRING ADDITION
 1030  *-------------------------------
 1040  S1     .AS /000189/
 1050  S2     .AS /007030/
 1060  *-------------------------------
 1070  S3     .AS /      /
 1080  *-------------------------------
 1090  ADD    LDX #5       6 DIGITS
 1100         CLC          START WITH NO CARRY
 1110  .1     LDA S1,X     NEXT DIGIT PAIR
 1120         AND #$0F     CHANGE ASCII TO BINARY CODE
 1130         ADC S2,X     RESULT IS IN ASCII AGAIN
 1140         CMP #$3A     UNLESS MORE THAN 9
 1150         BCC .2       OKAY
 1160         SBC #10      NEED TO PROPAGATE CARRY
 1170  .2     STA S3,X     SUM DIGIT IN ASCII
 1180         DEX          MORE DIGITS?
 1190         BPL .1       YES
 1200         RTS          NO, RETURN
```

==============================================================================
DOCUMENT :AAL-8301:DOS3.3:S.SuperScroll.txt
==============================================================================

```
1000  *SAVE SUPER SCROLL GENERATOR
1010  *--------------------------------
1020  *
1030  *    APPLE SUPER SCROLLER
1040  *
1050  *--------------------------------
1060  *      PROGRAM TO CREATE A FAST SCROLLER
1070  *
1080  *      CREATES AN ALL "IN-LINE" SCROLL ROUTINE
1090  *      WHICH MAY BE CALLED AS A SUBROUTINE.
1100  *
1110  *      WILL SCROLL LINES 2-24 UP TO LINES 1-23
1120  *      IN ONLY 7.6 MILLISECONDS.
1130  *
1140  *      BOTTOM LINE IS LEFT UNCHANGED; YOU MAY
1150  *      WISH TO ADD MORE CODE TO BLANK BOTTOM LINE.
1160  *--------------------------------
1170
1180  PGM.START.IN.RAM      .EQ $4000
1190  PROGRAM               .EQ $02 - $03
1200  UPPER.LINE            .EQ $04 - $05
1210  LOWER.LINE            .EQ $06 - $07
1220  *--------------------------------
1230         .MA SCRN
1240         .DA ]1,]1+$80,]1+$100,]1+$180
1250         .DA ]1+$200,]1+$280,]1+$300,]1+$380
1260         .EM
1270  *--------------------------------
1280  APPLE.SCREEN.ADDRESSES
1290         >SCRN $400     LINES 1-8
1300         >SCRN $428     LINES 9-16
1310         >SCRN $450     LINES 17-24
1320  *--------------------------------
1330  PGM.TO.WRITE.SCROLLING.PGM
1340
1350         LDA #PGM.START.IN.RAM
1360         STA PROGRAM
1370         LDA /PGM.START.IN.RAM
1380         STA PROGRAM+1
1390  *--------------------------------
1400         LDX #0       FOR LINE = 1 TO 23
1410  .1     LDA APPLE.SCREEN.ADDRESSES,X
1420         STA UPPER.LINE
1430         LDA APPLE.SCREEN.ADDRESSES+1,X
1440         STA UPPER.LINE+1
1450
1460         LDA APPLE.SCREEN.ADDRESSES+2,X
1470         STA LOWER.LINE
1480         LDA APPLE.SCREEN.ADDRESSES+3,X
```

```
       STA LOWER.LINE+1

       TXA          SAVE LINE #
       PHA
*--------------------------------
       LDX #40      FOR CHAR = 1 TO 40
.2     LDY #0
       LDA #$AD       "LDA ABSOLUTE"
       STA (PROGRAM),Y
       INY
       LDA LOWER.LINE
       STA (PROGRAM),Y
       INY
       LDA LOWER.LINE+1
       STA (PROGRAM),Y
       INY
       LDA #$8D       "STA ABSOLUTE"
       STA (PROGRAM),Y
       INY
       LDA UPPER.LINE
       STA (PROGRAM),Y
       INY
       LDA UPPER.LINE+1
       STA (PROGRAM),Y
*--------------------------------
       TYA          UPDATE PROGRAM POINTER
       SEC
       ADC PROGRAM
       STA PROGRAM
       BCC .3
       INC PROGRAM+1
.3     INC UPPER.LINE    NEXT CHAR POSITION
       INC LOWER.LINE
       DEX
       BNE .2
*--------------------------------
       PLA
       TAX
       INX          NEXT LINE
       INX
       CPX #2*23
       BNE .1
*--------------------------------
       LDY #0
       LDA #$60       "RTS"
       STA (PROGRAM),Y
       RTS
*--------------------------------
* A FAST MEMORY DUMP!!
*--------------------------------
MEML                 .EQ $8
MEMH                 .EQ $9
SCREEN.WRITE.LINE    .EQ $7D0
*--------------------------------
```

```
2030  START.DEMO
2040         JSR PGM.TO.WRITE.SCROLLING.PGM
2050  MEMDUMP
2060         LDA #0       DISPLAY FROM $0000 THRU $BFFF
2070         STA MEML
2080         STA MEMH
2090  .1     LDX #0       X = CHAR PNTR IN OUTPUT LINE
2100         LDA MEMH     DISPLAY ADDRESS
2110         JSR DISPLAY.BYTE
2120         LDA MEML
2130         JSR DISPLAY.BYTE
2140         LDA #$AD     "- "
2150         STA SCREEN.WRITE.LINE,X
2160         INX
2170         LDA #$A0
2180         STA SCREEN.WRITE.LINE,X
2190         INX
2200         LDY #0
2210  .2     LDA (MEML),Y      DISPLAY 8 BYTES
2220         JSR DISPLAY.BYTE
2230         LDA #$A0
2240         STA SCREEN.WRITE.LINE,X
2250         INX
2260         INY
2270         CPY #8
2280         BNE .2
2290  .3     STA SCREEN.WRITE.LINE,X
2300         INX
2310         CPX #40      CLEAR TO END OF LINE
2320         BCC .3
2330  *--------------------------------
2340         JSR PGM.START.IN.RAM
2350  *--------------------------------
2360         LDA #8
2370         CLC
2380         ADC MEML
2390         STA MEML
2400         LDA MEMH
2410         ADC #0
2420         STA MEMH
2430  .4     CMP #$C0     STOP AT $BFFF
2440         BNE .1
2450         RTS
2460  *--------------------------------
2470  DISPLAY.BYTE
2480         PHA
2490         LSR
2500         LSR
2510         LSR
2520         LSR
2530         JSR DISPLAY.NYBBLE
2540         PLA
2550         AND #$0F
2560  DISPLAY.NYBBLE
```

```
2570          ORA #$B0     MAKE HEX DIGIT
2580          CMP #$BA
2590          BCC .1
2600          ADC #6
2610  .1      STA SCREEN.WRITE.LINE,X
2620          INX
2630          RTS
```

==============================================================
DOCUMENT :AAL-8302:Articles:Front.Page.txt
==============================================================

$1.50

Volume 3 -- Issue 5 February, 1983

In This Issue...



S-C Macro Assembler ///

The Apple /// version of the S-C Macro Assembler is coming right along! I am now selling a preliminary "as is" version for $100. That buys you the assembler, a few pages of documentation about the differences from the Apple ][ version, and free updates until the finished product appears. This is a working assembler for producing free-running programs; it assembles itself just fine. The biggest gap is the ability to produce relocatable modules for Pascal or BASIC. That will be added next. Call or write if you are interested in being among the first to have this new enhancement to the Apple ///.

Zero-Insertion-Force Game Socket Extender

One of the first things I did to my Apple back in 1977 was to plug a ZIF socket into the game connector. Not too easy, because it first has to be soldered to a header, but I did it.

Now I have discovered a source for a ready-made device that does the same thing, plus brings the socket outside the Apple (if you so desire). There's a picture of the device on page 14. For only $20 I'll send you one!

65C02

Many of you have expressed an interest in the new Rockwell R65C02 microprocessor. Well, I still haven't heard any more than I mentioned a couple of months ago. We're as eager as you are to get a sample. We'll have a detailed report as soon as we know more.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050.

Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).



=======================================================================
DOCUMENT :AAL-8302:Articles:Gilder.Note.txt
=======================================================================

Promising New Book

I just received an advance copy of a forthcoming book by Jules Gilder (a long-time AAL subscriber), titled "Now That You Know Apple Assembly Language, What Can You Do With It?" As the title implies, this will be an intermediate level look at really using assembly language in your Apple. It looks good. As soon as I have details about price and publication date, I'll let you know.

==============================================================================
DOCUMENT :AAL-8302:Articles:IIe.txt
==============================================================================

Apple //e Notes.............................Bob Sander-Cederlof

We don't have one yet, but we did play with one for about an hour last week.  All our software works fine, as long as you stay in the 40-column caps-lock mode.  We will be making new versions available in the near future which take full advantage of the extended memory, lower-case, and 80-column display.

The best write-up I have seen yet on the //e is in the February 1983 Apple Orchard (published by the International Apple Core, 908 George St., Santa Clara, CA 95050).

Here are some of the things that caught my attention:

*  Real shift key, and a caps-lock key.

*  Open-Apple and Closed-Apple keys, which duplicate the first two paddle buttons.

*  Recessed RESET key.  CTRL-RESET required (no longer a switchable option).  CTRL-Closed-Apple-RESET starts a memory test program.

*  Two 8K ROMs, instead of six 2K ROMs.  The extra 2K of ROM space is used by the modified Monitor program.  Fancy soft-switches map the extra 2K into the $C000-C7FF space.  These sockets are supposedly compatible with 2764 EPROMs.

*  Apparently the Monitor now uses (clobbers) zero-page locations $08 and $1F.

*  Up- and down-arrows on the keyboard.  Down is CTRL-J, or linefeed. Up is CTRL-K.

*  The keyboard includes all the ASCII set, even $7F (DELETE, or RUBOUT).

*  64K RAM on the motherboard.  This simulates an Apple II Plus with a 16K RAM card in slot 0.

*  New slot instead of slot 0, with 60-pin connector (other slots still have 50-pin connectors).  Apple's 80-column card plugs in here. The extra pins carry other signals not normally available at the slots.  Look for some amazing new combined function cards from the peripheral-card makers for this slot!  I wouldn't be surprised to find ads real soon for 256K RAM cards including 80-column support, clock-calendar, serial/parallel interfaces, and all on one card.

*  80-column card with or without extra 64K RAM.  But this 64K RAM is soft-switched in a totally different manner.  It maps over the same

space as the motherboard 64K, with switches to map portions such as page-zero, text screen, hi-res screen, and so on.

* Now you can READ the state of most of the soft-switches. Bit 7 (high bit) tells the state, as follows:

```
$C013 -- RAMREAD
$C014 -- RAMWRT
$C015 -- SLOTCXROM/CX00ROM
$C016 -- ALTZP/MAIN
$C017 -- SLOTC3ROM/SLOTROM
$C018 -- 80 COL STORE
$C019 -- VERTICAL BLANKING
$C01A -- TEXT
$C01B -- MIXED MODE
$C01C -- PAGE2
$C01D -- HIRES
$C01E -- ALTCHAR
$C01F -- 80 COL DISP
```

* Yes, you saw right...the vertical blanking signal is now readable! So lovers of Lancaster's Enhancements can continue to tinker!

* Inverse lower-case display is selectable, at the expense of the flashing mode.

* The cursor display is different. A small checkerboard alternates with the character under the cursor in 40-column mode. In 80-column mode an inverse blank is the normal cursor, and an inverse "+" is used when in escape-mode.

Whether we view the changes as improvements or not, the //e will very soon be the standard we all have to deal with. The same situation arose when Apple switched from II to II Plus. A year from now, when 300,000 have been sold, we will wonder how we ever lived without it!

=====================================================================
DOCUMENT :AAL-8302:Articles:MoreVidexPatchs.txt
=====================================================================

More on the Macro-Videx Connection.................Bill Linn

Don Taylor's original article in the August (1982) issue of AAL and Mike Laumer's follow-up the next month gave us the patches for running the S-C Macro Assembler in conjunction with the Videx 80-column board. I recently purchased a Videx card in order to implement the 80-column version of ES-CAPE, so I installed the patches.

I have really enjoyed using the Macro assembler in 80-column mode. Naturally, though, I couldn't resist adding a few enhancements to Don's and Mike's work.

Mike added the right arrow code, which copies characters off the Videx screen, but he stopped short of implementing the Escape-L LOAD sequence.  To install the following code, you will need to change line 3080 in Don's article to point to my routine.  Change it to "3080 .DA MY.ESC.L-1".  Also, the STX instruction at line 4235 in Mike's article must be labelled GETCH.

```
*---------------------------------------
SCM.INSTALL .EQ SCM.BASE+$52A
*
MY.ESC.L
       CPX #0          CURSOR AT BEGINNING?
       BEQ .1          YES, CONTINUE
       JMP SCM.ESC.L   NO, LET S-C HANDLE IT
.1     LDA #0          CONNECT DOS
       STA $AA52       BY SETTING INTERCEPT STATE = 0
       LDA #$84        SEND A CTRL-D
       JSR MON.COUT
.2     LDA LOADCMD,X
       JSR SCM.INSTALL
       JSR FAKE.COUT
       CPX #6
       BCC .2
.3     STX $406        SAVE CHAR POS'N
       JSR GETCH       GET SCREEN CHAR
       LDX $406        RESTORE POS'N
       JSR SCM.INSTALL
       JSR FAKE.COUT
       CPX #40         40 CHARS SENT YET?
       BNE .3          NO, LOOP BACK
       JMP CLREOP      CLEAR TO END OF PAGE
*                      AND EXIT
*
LOADCMD    .AS -/LOAD  /
*---------------------------------------
```

Secondly, I wanted a longer "*---" line on my screen, so I changed it to 68 characters instead of 38. This uses more of the 80-column screen, without wrapping around during assembly. To make this modification insert the following two lines after the label "INSTALL.PATCHES" in Don's original listing:

```
     LDA #68
     STA SCM.BASE+$494
```

Finally, I changed the dimensions of the Videx cursor so that it looks like a blinking underline instead of a blinking block. (Users of my ES-CAPE are already familiar with my love for the blinking underline!) Insert the following lines immediately after the "INSTALL.VECTORS" label:

```
LDA #$0A       VIDEX REGISTER 10
STA V.DEV0
LDA #$68
STA V.DEV0+1
LDA #$0B       VIDEX REGISTER 11
STA V.DEV0
LDA #$08
STA V.DEV0+1
```

Speaking of ES-CAPE, I am making progress on Version 2 and have included suggestions from many of you. If you have others, please drop me a line soon at 3199 Hammock Creek, Lithonia, GA 30058, or call evenings at (404) 483-7637.

```
==================================================================
DOCUMENT :AAL-8302:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler (the best there is!)...........................$80.00
Upgrade from Version 4.0 to MACRO..................................$27.50
Source code of Version 4.0 on disk.................................$95.00
    Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// ..........................................$100.00
    Preliminary version.  Call or write for details.

Applesoft Source Code on Disk......................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.

ES-CAPE:  Extended S-C Applesoft Program Editor....................$60.00

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)................... $79.00
Source Code for FLASH! Runtime Package.............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware).......................$45.00
The Incredible JACK!...............................................$79.00

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each

Ashby Shift-Key Mod................................................$15.00
Lower-Case Display Encoder ROM.....................................$25.00
     Only Revision level 7 or later Apples.

Books, Books, Books..........................compare our discount prices!
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
```

```
"What's Where Guide" (updates first edition)..........($9.95)   $9.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

 Add $1 per book for US postage.  Foreign orders add postage needed.

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

```
===============================================================================
DOCUMENT :AAL-8302:Articles:Patch.TF.txt
===============================================================================
```

Macro Assembler Patch

Peter Bartlett, of Chicago, has reported an unpublished limit on the number of Target Files that can be generated by one assembly.  Right now there can only be 31; above that number the load address and length bytes go astray.  If you need more than 31 files from one assembly, you can make the following patches:

Regular version

```
:$29EA:3F
```

Language Card version

```
:$C083 C083 EB36:3F N C080
```

These patches will allow you to have up to 63 target files.  That should be plenty!

==============================================================================
DOCUMENT :AAL-8302:Articles:Patch.TI.txt
==============================================================================

Patch to Fix .TI Problem.........................Mike Laumer

You may have noticed the annoying problem with the .TI directive, in which there is sometimes a blank line after the title line and sometimes not. The blank line is there when the page break is forced with a .PG directive, but not when it is caused by merely filling a page.

The following little patch will fix it. I haven't put a definite address on the patch, because I don't know what other patches you may already have appended to the assembler. Just find an empty place and plop it in!

```
Motherboard version:   :$21F0:4C xx yy   (was 20 CF 2C)
                       :$yyxx:20 CF 2C 4C E3 21

RAM Card version:      :$E33C:4C xx yy   (was 20 1B EE)
                       :$yyxx:20 1B EE 4C 2F E3
```

Another .TI problem of which I am aware is that the line count is messed up on the first page of the symbol table listing. This is caused by the fact that the extra carriage returns in the "SYMBOL TABLE" message are not counted. You can clean up the appearance by making the last line of your source program be ".PG"; this forces the symbol table to start on a fresh page.

```
==============================================================================
DOCUMENT :AAL-8302:Articles:PtchMacroHex.txt
==============================================================================
```

A Sometimes Useful Patch...................Bob Sander-Cederlof

Sometimes you would like to see all the hex bytes a macro produces, but not the expanded lines of source code.  The >LIST MOFF directive turns off both, but with the following three byte patch you can see the hex bytes for each macro call.

```
Motherboard version:   :$218B:0   (was 03)
                       :$21B3:0   (was 05)
                       :$21E2:0   (was 10)

RAM Card version:      :$C083 C083   (enable writing)
                       :$E2D7:0   (was 03)
                       :$E2FF:0   (was 05)
                       :$E32E:0   (was 10)
```

Don't make these into permanent patches, because there will be times when you want to use the .LIST directives normally.  If you feel like making the changes often, you might make two separate versions of the assembler, or make some EXEC files to do the patching on demand.

```
==============================================================================
DOCUMENT :AAL-8302:Articles:Quickie.6.txt
==============================================================================
```

Quickie No. 6.....An Eleven-Byte Alarm.....Bob Sander-Cederlof

Here is a little run-anywhere program sure to wake up the neighborhood dogs.  Put it in your program as a last resort to get attention, because the only escape is by RESET or power-off.

```
1000 ALARM  INY        INCREMENT DELAY TIME
1010        TYA
1020        TAX        DELAY COUNT TO X
1030        LDA $C030  TOGGLE SPEAKER
1040 .1     DEX        DELAY LOOP
1050        BNE .1
1060        BEQ ALARM  MORE NOISE, FOREVER....
```

That's it, only eleven bytes!  For a slightly different effect, change the "DEX" instruction in line 1030 to "INX".

==============================================================================
DOCUMENT :AAL-8302:Articles:SC.WP.txt
==============================================================================

Source Code for a Word Processor...........Bob Sander-Cederlof

I finally have had to face it. I am never going to have time to finish the S-C word processor. It is certainly usable, because we have been using it here for months now. And we use it a lot, writing the newsletter, manuals, letters, etc. My father-in-law uses it, and so does my best friend, Fred. Fred's 11-year-old daughter is also using it, and loves it. She is currently typing a research paper using it.

I know it is easy to use, because I didn't even give Fred a list of commands, let alone a reference manual. Of course, I did sit down with them for a few hours at the first, because they had never even seen a word processor before.

In power, it is somewhere between Applewriter 1.1 and Applewriter II. It is similar in operation to Applewriter 1.1, and works in 40-column mode only. It requires a lower-case display and shift-key mod.

It can read Applewriter 1.1 files, and instantly convert them to standard ASCII form. Normally it uses standard Apple text files (type T in the catalog). Of course, with Bobby Deen's help, I built in FAST read and write of those text files. Faster than binary files, actually. Something like 100 sectors in 7 seconds, if I remember correctly.

I want to make a deal with you. I'll send you the complete commented source code on disk, together with a few sample text files. The text files will describe the command repertoire. If you are already familiar with Applewriter 1.1, you won't have any trouble at all. The assembled word processor will also be there, in case you don't have the S-C Macro Assembler.

But if you do have my assembler, you can proceed to modify, improve, augment, enhance, and so on, to your heart's content.

I'll send you the disk, if you'll send me $50. Or your charge card numbers, of course. I also want your commitment to keep this in the family. You know, don't go out and write a manual and wrap it in a fancy cover and call it YOUR product!

If you do enhance it, send in your additions and we'll make this a joint effort. With all of us working on it, we may soon have the world's best word machine!

==============================================================================
DOCUMENT :AAL-8302:Articles:Scooter.txt
==============================================================================

Many of you have expressed an interest in the new Rockwell R65C02 microprocessor. Well, we still haven't heard any more than I mentioned a couple of months ago. We're as eager as you are to get a sample. We'll have a detailed report as soon as we know more.

Zero-Insertion-Force Game Socket Extender

One of the first things I did to my Apple back in 1977 was to plug a ZIF socket into the game connector. Not too easy, because it first has to be soldered to a header, but I did it.

Now I have discovered a source for a ready-made device that does the same thing, plus brings the socket outside the Apple (if you so desire). For only $20 I'll send you one!

```
=============================================================================
DOCUMENT :AAL-8302:Articles:Skinny.Page.txt
=============================================================================
```

On CATALOG ARRANGER and RAM Card DOS

Chuck Welman just called to report some errors in the January piece on using CATALOG ARRANGER with a relocated DOS.  He says that the sentence about where to put the BIT MONREAD statements had problems. Here's his corrected version:

"Then add BIT MONREAD at these positions:  Lines 1675, 3775, 3895, 3955, 4015 (".5" moved to this line), 4205 (".3" moved to this line, 4315, 4425, 4455 (".7" moved to this line), and 4895."

Chuck also passed along instructions for using FILENAME EDITOR with a RAM Card DOS.  Here are his additions:

```
2635 .3     BIT MONREAD
2640        JSR MON.BELL
2642        BIT DOSREAD
2644        BIT DOSREAD
2646        RTS
```

Thanks to all of you for showing your appreciation for these programs.

Quickie No. 6.....Bob Sander-Cederlof

Here is a little run-anywhere program sure to wake up the neighborhood dogs.  Put it in your program as a last resort to get attention, because the only escape is by RESET or power-off.

```
1000 ALARM  INY       INCREMENT DELAY TIME
1010        TYA
1020        TAX       DELAY COUNT TO X
1030        LDA $C030 TOGGLE SPEAKER
1040 .1     DEX       DELAY LOOP
1050        BNE .1
1060        BEQ ALARM ....FOREVER....
```

That's it, only eleven bytes!  For a slightly different effect, change the "DEX" in line 1030 to "INX".

```
=====================================================================
DOCUMENT :AAL-8302:Articles:Stars.txt
=====================================================================
```

Star-tling Stunts.....................Bill Morgan & Mike Laumer

In most assemblers, including the S-C Macro Assembler, you can use the character "*" in the operand of an instruction to mean the current value of the location counter. (The location counter is a variable used by the assembler to keep track of where the next byte of object code goes.) Here are a couple of simple examples of using the *, from page 6-2 of the Macro Assembler manual:

```
0800- 03       1000 QT     .DA #QTSZ
0801- 41 42 43 1010        .AS /ABC/
0003-          1020 QTSZ   .EQ *-QT-1
               1030
0804- 00 00    1040 VAR    .DA *-*
               1050
0806-          1060 FILLER .BS $900-*
0900-          1070 END    .EQ *
```

The QT, QTSZ example uses the * to help calculate the length of a string of characters. The VAR line uses "*-*" to define a variable as having a value of zero.

The expression labelled FILLER causes the assembler to skip ahead to $900. This has much the same effect as .OR $900, but it won't cause the assembler to close a target file, the way .OR would.

One thing Bill wanted was an expression to have the assembly skip up to the beginning of the next page, no matter what that page might be. Here's what we came up with:

```
0800- 34 12    1000 START  .DA $1234
0802-          1010 FILL   .BS *+255/256*256-*
0900- 45 23    1020 END    .DA $2345
```

If you change the origin to $C00, END will move to $D00. With this coding, END will always be $100 above START. Note that there is no precedence when the assembler is evaluating an expression. Terms are taken strictly left-to-right. But notice how smart the expression cracker in the assembler is! It knows that a "*" between numbers or labels means "multiply", and a "*" between arithmetic operators means "location counter".

In the American Heart Association CPR project Mike uses lots of overlays, and has to make sure that modules don't grow above a certain address. He does it by putting lines like these at the end of a module:

```
1000        .DO *>LIMIT
1010   !!! PROGRAM TOO BIG !!!
```

```
1020        .FIN
```

Here's an example, to keep a program below the Hi-res pages:

```
1000        .OR $1FFE
1010        .DA $4321
1020        .DO *>$2000
1030   !!! PROGRAM TOO BIG !!!
1040        .FIN
```

That will assemble just fine:

```
               1000        .OR $1FFE
1FFE- 21 43    1010        .DA $4321
               1020        .DO *>$2000
               1040        .FIN

0000 ERRORS IN ASSEMBLY
```

But, try inserting another line:

```
1015        .DA $1234
```

Here's what happens:

```
*** BAD OPCODE ERROR
 1030   !!! PROGRAM TOO BIG !!!

0001 ERRORS IN ASSEMBLY
```

The key to this technique is putting a couple of blanks at the beginning of line 1030. That way, the assembler tries to parse "!!!" as an opcode, and reports an error during pass one, before any code has been generated.

You should be very careful about using "*", and experiment on a test disk when trying something new. For example, take another look at line 1060 in the first listing. If you put "*-$900" for the operand, that would be negative. The result would be $FF07, which would try to write 65,287 zero bytes onto your target file. The next thing you see is probably DISK FULL!

That's about all the tricky things we have room for right now. We hope these hints will help you to navigate "by the stars" in your programming. Just remember to experiment carefully with the * operand before using it in vital programs. There are also many pitfalls on this road!

==============================================================================
DOCUMENT :AAL-8302:Articles:String.Adder.txt
==============================================================================

Really Adding ASCII Strings................Bob Sander-Cederlof

Last month I promised a "reasonably useful" program to add two numbers together from ASCII strings.  I promised:

*  Callable from Applesoft, using &.

*  Automatic passing of string parameters.

*  Allow operands of unequal length.

*  Automatic alignment of decimal points.

*  Allow negative numbers.

*  Handle sums longer than operands.

*  Allow leading blanks on operands.

*  Allow operands and results up to 253 bytes long!

Okay!  It took me three days, but I did it!  Of course, the program has grown from 12 lines and 26 bytes of code to over 290 lines and over 450 bytes, too.

The program is now assembled to load at $9000, but you can choose other positions by changing line 1130.  I set HIMEM:36864 before doing anything else in the Applesoft program, and then BRUN B.STRING ADDER.

When B.STRING ADDER is BRUN, only the setup code in lines 1160-1220 is executed.  What this does is link in the ampersand (&) to the body of my program.  Once the "&" is linked, my program responds to a call like "& +$,A$,B$,C$" by adding the numeric values represented in ASCII in A$ and B$ and storing the sum as a string in C$.

When an &-line occurs, Applesoft branches to my line 1520.  Lines 1520-1600 check for the characters "+$," after the ampersand.  If you don't like those characters, change them to something else.  Anyway, if the characters do not match, you get SYNTAX ERROR.  If they do match, it is time to collect the three strings variables.

Lines 1620-1690 collect the three string variables.  The first two are the operands, the third is the result string.  I save the address and length of the actual data of the operand strings.  All I save at this point for the result string is the address of the variable descriptor. I call the subroutine PARSE.STRING.NAME to check for a leading comma, search for the variable name, and store the length and address of the referenced string data.

Lines 1730-1860 scan each operand string in turn to find the decimal point position. The routine SCAN divides a string at the decimal point (or where the decimal point would be if there was one), and returns in Y the number of characters to the left of the decimal point. SCAN returns in X the count of the

number of characters on the right end, including the decimal point. I save the "digits.after" parts of both strings, and also the maxima of the two parts. The maxima describe the result string (almost).

Lines 1900-2000 finish the description of the result string, by lengthening the integral (left) side by two characters. These two characters allow for extension of the result by carry, and for representation of the sign of the result using ten's complement notation. At this point I also clear the necessary bytes of the result to zero, so the buffer can be used as an accumulator.

Now comes the EASY part. Lines 2040-2100 add each operand in turn to the buffer contents. EASY. Just call the subroutine ADD.TO.BUFFER, and it's done! Don't worry, I'll amplify later.

In ten's complement notation, if the first digit is 0-4 the number is positive; if the first digit is 5-9, the number is negative. For example, 1234 looks like 001234; -1234 becomes 998766. Ten's complement means in decimal the same thing two's complement means in binary. I can form the ten's complement by subtracting the number from a power of ten equal to the number of digits in the result. In that example, 1000000-1234=998766. Note that the ten's complement is equal to the nine's complement plus one. (Since 10=9+1.)

Lines 2140-2410 convert the buffer contents from the ten's complement numeric notation back to ASCII. Lines 2140-2180 set or clear the CARRY and TENS.FLAG sign bits according to the first digit in the buffer. A negative number, with a first digit of 5-9, causes both of these variables to get a value of the form 1xxxxxxx.

Lines 2190-2360 scan through the number from right to left, making the ten's complement if the number was negative, and converting each digit to ASCII. Lines 2370-2400 store a minus sign in the first digit position if the result is negative.

Line 2410 calls a subroutine to chop off leading zeros, and move the minus sign if there is one. You may justifiably ask, "Why did you call a subroutine rather than use in-line code?" Because when I wrote it in-line, the local labels stretched out too far from the major label STRADD and caused an assembly error. Also, sometimes I use subroutines for clarity, even when the subroutine is only called once.

The final step is to pack the resulting string up and ship it to the result string variable. Lines 2450-2590 do just that. AS.GETSPA makes room at the bottom of string pool space, and AS.MOVSTR copies the string data. C'est finis!

Lines 2640-3100 do the actual addition. On entry, X is either 0 or 4, selecting either the first or second operand. SETUP.OPERAND copies the string address into VARPNT, and retrieves the length of the string. Lines 2690-2760 set or clear the TENS.FLAG and CARRY variables according to the sign of the operand.

Lines 2780-2810 compute the position in the buffer at which the operand will be aligned properly. We saved the size of the integral (left) side of the buffer in MAX.DIGITS.BEFORE. That plus the lenght of the fractional side of the operand tells us where this operand aligns. Since we are using ten's complement for negative numbers, rather than nine's complement, we don't have to worry about extending the fractional parts to the same length. We can just start adding at the end of the current operand. (In ten's complement form fractional extensions are zeros; in nine's complement form, the extension digits would all be nines.)

Lines 2830-3100 do the addition. X points into the buffer, and Y points into the operand string. To start with, both X and Y point just past the end; therefore the loop BEGINS with a test-and-decrement sequence. I first t-a-d the buffer pointer; if it is zero, all is finished. If not, on to t-a-d the string pointer. If it is zero, there are still digits left in the buffer, so I use an assumed leading zero digit for the operand. We still may have carries to propagate across the rest of the sum.

Assuming neither pointer is zero, line 2900 gets the next digit from the operand string. If it is a decimal point, I just store the decimal point ASCII value into the buffer. If you want to be able to ignore leading blanks, insert the following two lines between line 2920 and 2930:

```
2924       CMP #'          BLANK?
2925       BEQ .3          YES, USE ZERO.
```

I left them out in my version, because I forgot I promised it to you.

If the character is not a decimal point (or blank), it may be a minus sign or digit. I did not put any error checking in my program for other extraneous characters; if you try them, you will get extraneous results! I treat a sign as a leading zero in the arithmetic loop.

If the character is a digit, or an assumed leading zero, we can add it to the buffer's value. Lines 2960-3010 will complement the digit if the operand had a minus sign. Lines 3020-3070 add the current operand digit (or its complement) to the current buffer digit, plus any carry hung over from the preceding digit, and save the resulting carry in CARRY.

That's it! Now here is a short little Applesoft program to test the code.

```
100  REM TEST&+$,A$,B$
110  HIMEM: 36864: PRINT  CHR$ (4)"BLOAD B.STRING ADDER":
```

```
     CALL 36864
120  INPUT A$: INPUT B$
130  &  + $,A$,B$,C$
140  PRINT C$: GOTO 120
```

```
===================================================================
DOCUMENT :AAL-8302:Articles:Trapper.txt
===================================================================
```

TRAPPER:  An Applesoft Input Tuner...............Allen Marsalis

How would you like a radio which played every available station at one time?  Well that's how I sometimes feel about using Applesoft's INPUT statement.  I want to be able to "tune in" on the character(s) of the input stream, in much the same way as a radio tunes into a station. Applesoft's INPUT statement, however, accepts all characters typed into the keyboard and allows up to 255 of them.  This means that I have to do a lot of checking and monitoring of string lengths and characters to avoid input errors.

For example, when answering a Y or N question, what happens when the user inputs "WXYZ"?  Provisions are needed within the program to guard against such errors.  This can be very inconvenient and space-consuming, yet it is essential for good programming.

A better example occurs when you are creating a disk file.  Field lengths and data types are often restricted, such as in a name, address, or social security number.  A SSN, for instance, has a fixed length and must be constructed of numbers only.  Checking a field such as this can be very time consuming and lengthy.  In fact, it seems that a quarter of the contents of my Applesoft programs does nothing but check on field lengths, option boundaries, and other input checks.

So, I set out to create an input routine which would allow Applesoft to "tune" into the characters specified and also monitor the field length.  I've seen several input routines such as this on larger systems, but all had one disadvantage:  Only a fixed number of options were available, such as alpha only, numeric only, and (Y or N) input. More options available meant more parameters were necessary, making the systems more cumbersome to work with.  After much thought I decided on a totally new approach which would allow almost limitless control of input.  I christened this routine TRAPPER for "Tuning and Regulating APPlesoft Entries by Restriction."

TRAPPER employs a coded restriction string (not unlike Applesoft's IF expression) to tune out the characters I don't want to accept. TRAPPER is then, in essence, a tiny interactive interpreter that provides a short, convenient method of filtering out any unwanted characters in the input.  Here's how it works.

TRAPPER uses three parameters as follows:

```
Syntax:  & INPUT (A, B$, C$)
     A:  Input field length (real expression)
    B$:  Coded restriction string (string expression)
         includes:  > < = ' AND OR NOT <sp> <single char>
    C$:  Input string (string variable)
         variable to receive input
```

As I have said, the restriction string is a simple relational expression as is used by Applesoft's IF statement. It is constructed of the following special characters and rules:

1) < > = are its relational operators
2) AND OR NOT are its logical operators
3) Blanks are allowed anywhere within the expression, but lengthy expressions increase the delay between keystrokes.
4) One and only one character is allowed within single quotes.
5) <cr> and <-- have special functions and cannot be trapped.
6) Parentheses are not yet implemented.

EXAMPLES:

```
YN$ = " ='Y' OR ='N' "            :REM (Y OR N) ONLY
NOSP$ = " NOT =' ' "              :REM NO SPACES ALLOWED
MENU$ = " NOT <'1' AND NOT >'4' " :REM ALLOWS 1 THRU 4
WAITCR$ = ""                      :REM WAIT FOR A <CR>
```

After using Trapper awhile, I noticed a significant reduction in the size of my Applesoft programs, with even better error trapping than ever before possible. And it doesn't print that leading question mark which I never did like (not all input prompts are questions.)

For a 48K Apple, DOS sets HIMEM at $9600. Trapper resides just below this at $9300 and moves HIMEM down to that point.

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:Divide.16.16.txt
==============================================================================

 1000  *SAVE S.DIV.16/16
 1010  *--------------------------------
 1020  *      DIVIDE 16 BY 16
 1030  *--------------------------------
 1040  ACL    .EQ $50
 1050  ACH    .EQ $51
 1060  XTNDL  .EQ $52
 1070  XTNDH  .EQ $53
 1080  AUXL   .EQ $54
 1090  AUXH   .EQ $55
 1100  *--------------------------------
 1110  DIVMON LDY #16      INDEX FOR 16 BITS
 1120  .1     ASL ACL      DIVIDEND/2, CLEAR QUOTIENT BIT
 1130         ROL ACH
 1140         ROL XTNDL
 1150         ROL XTNDH
 1160         SEC
 1170         LDA XTNDL    TRY SUBTRACTING DIVISOR
 1180         SBC AUXL
 1190         TAX
 1200         LDA XTNDH
 1210         SBC AUXH
 1220         BCC .2       TOO SMALL, QBIT=0
 1230         STX XTNDL    OKAY, STORE REMAINDER
 1240         STA XTNDH
 1250         INC ACL      SET QUOTIENT BIT = 1
 1260  .2     DEY          NEXT STEP
 1270         BNE .1
 1280         RTS
 1290  *--------------------------------
 1300  *      SIGNED DIVISION 32/16
 1310  *--------------------------------
 1320  SIGN   .EQ $2F
 1330  *--------------------------------
 1340  SIGNED.DIV.MON
 1350         LDY #0
 1360         STY XTNDL    CLEAR ACC EXTENSION
 1370         STY XTNDH
 1380         STY SIGN
 1390         LDX #ACL
 1400         JSR ABS
 1410         LDX #AUXL
 1420         JSR ABS
 1430         JSR DIVMON
 1440         LDA SIGN
 1450         BPL .1       RESULT POSITIVE
 1455         LDX #ACL
 1460         JSR COMPLEMENT
 1470  .1     RTS
```

```
1480  *--------------------------------
1490  ABS    LDA 1,X      LOOK AT SIGN
1500         BPL ABSRET   POSITIVE
1510         EOR SIGN     COMPLEMENT RESULT SIGN
1520         STA SIGN
1530  COMPLEMENT
1540         SEC
1550         TYA          =0
1560         SBC 0,X
1570         STA 0,X
1580         TYA          =0
1590         SBC 1,X
1600         STA 1,X
1610  ABSRET RTS
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.ARRAYS.txt
==============================================================================

 1000  *  S.ARRAYS
 1010  *--------------------------------
 1020  CHRGET   .EQ $B1
 1030  CHKCOM   .EQ $DEBE
 1040  SYNCHR   .EQ $DEC0
 1050  PTRGET   .EQ $DFE3
 1060  GETARYPT .EQ $F7D9
 1070  PRNTAX   .EQ $F941
 1080  CROUT    .EQ $FD8E
 1090  PRHEX    .EQ $FDDA
 1100  COUT     .EQ $FDED
 1110  *--------------------------------
 1120  LENGTH       .EQ 0
 1130  STRING.ADDR  .EQ 1,2
 1140  ELEMENT.PNTR .EQ 3,4
 1150  ARRAY.END    .EQ 5,6
 1160  *--------------------------------
 1170         .OR $300
 1180
 1190  START  LDA #X
 1200         STA $3F6
 1210         LDA /X
 1220         STA $3F7
 1230         RTS
 1240  *--------------------------------
 1250  *  GET ONE ARRAY ELEMENT
 1260  *--------------------------------
 1270  X      CMP #'X
 1280         BNE Y
 1290         JSR CHRGET
 1300         JSR CHKCOM      BE SURE COMMA IS NEXT
 1310         JSR PTRGET
 1320  *--------------------------------
 1330  *   NOW $83,84 POINTS AT A$(3,5)
 1340  *--------------------------------
 1350         LDY #0          FIRST BYTE IS STRING LENGTH
 1360         LDA ($83),Y     GET LENGTH
 1370         STA LENGTH
 1380         INY             NEXT TWO BYTES POINT
 1390         LDA ($83),Y     AT STRING VALUE
 1400         STA STRING.ADDR
 1410         INY
 1420         LDA ($83),Y
 1430         STA STRING.ADDR+1
 1440  *--------------------------------
 1450  *   NOW LET'S PRINT THE STRING, JUST FOR FUN
 1460  *--------------------------------
 1470         LDY #0
 1480  .1     CPY LENGTH
```

```
1490        BCS .2         FINISHED
1500        LDA (STRING.ADDR),Y
1510        ORA #$80
1520        JSR COUT
1530        INY
1540        BNE .1         ...ALWAYS
1550 .2     JMP CROUT
1560 *--------------------------------
1570 *  GET ENTIRE ARRAY
1580 *--------------------------------
1590 Y      LDA #'Y
1600        JSR SYNCHR
1610        JSR CHKCOM
1620        JSR GETARYPT
1630 *--------------------------------
1640 *   NOW $9B,9C HAVE ADDRESS OF START OF ARRAY
1650 *   NEED TO MOVE POINTER UP TO FIRST ELEMENT
1660 *--------------------------------
1670        LDY #4      POINT AT LSB OF # DIMENSIONS
1680        LDA ($9B),Y
1690        ASL         DOUBLE IT (IGNORE MSB, #<120)
1700        ADC #5      POINT AT FIRST ELEMENT
1710        STA $9D
1720        LDY #2      POINT AT LSB OF OFFSET
1730        CLC         COMPUTE ADDRESS JUST PAST END OF ARRAY
1740        LDA $9B
1750        ADC ($9B),Y
1760        STA ARRAY.END
1770        LDA $9C     MSB
1780        INY
1790        ADC ($9B),Y
1800        STA ARRAY.END+1
1810 *--------------------------------
1820 *      NOW COMPUTE FULL ADDRESS OF FIRST ELEMENT
1830 *--------------------------------
1840        CLC
1850        LDA $9D
1860        ADC $9B
1870        STA ELEMENT.PNTR
1880        LDA $9C
1890        ADC #0
1900        STA ELEMENT.PNTR+1
1910 *--------------------------------
1920 *      NOW WALK THROUGH STRINGS
1930 *--------------------------------
1940 .1     LDY #0               POINT AT FIRST ELEMENT
1950        LDA (ELEMENT.PNTR),Y  GET LENGTH
1960        STA LENGTH
1970        INY
1980        LDA (ELEMENT.PNTR),Y  GET ADDRESS
1990        TAX
2000        INY
2010        LDA (ELEMENT.PNTR),Y
2020        JSR PRNTAX
```

```
2030        LDA #':+$80
2040        JSR $FDED
2050        LDA #' +$80
2060        JSR $FDED
2070        JSR $FDED
2080        LDA LENGTH
2090        JSR PRHEX
2100        JSR CROUT
2110 *--------------------------------
2120        CLC
2130        LDA #3
2140        ADC ELEMENT.PNTR
2150        STA ELEMENT.PNTR
2160        LDA ELEMENT.PNTR+1
2170        ADC #0
2180        STA ELEMENT.PNTR+1
2190 *--------------------------------
2200        LDA ELEMENT.PNTR
2210        CMP ARRAY.END
2220        LDA ELEMENT.PNTR+1
2230        SBC ARRAY.END+1
2240        BCC .1
2250        RTS
```

```
==============================================================
DOCUMENT :AAL-8302:DOS3.3:S.Div.32.16.Trc.txt
==============================================================

 1000  *SAVE S.DIVIDE 32/16 WITH TRACE
 1010  *--------------------------------
 1020  OVERFLOW   .EQ $00
 1030  DIVIDEND   .EQ $01 THRU $04
 1040  REMAINDER  .EQ DIVIDEND
 1050  QUOTIENT   .EQ DIVIDEND+2
 1060  DIVISOR    .EQ $05 AND $06
 1070  *--------------------------------
 1080  MON.CROUT  .EQ $FD8E
 1090  MON.PRHEX  .EQ $FDDA
 1100  MON.COUT   .EQ $FDED
 1110  *--------------------------------
 1120  DIVIDE LDX #17           16-BIT DIVISOR
 1130         CLC               START WITH NO OVERFLOW
 1140  .1     ROR OVERFLOW
 1150         JSR TRACE
 1160         SEC
 1170         LDA DIVIDEND+1    NEXT-TO-HIGHEST BYTE
 1180         SBC DIVISOR+1     LEAST SIGNIFICANT BYTE
 1190         TAY               SAVE RESULT
 1200         LDA DIVIDEND      HIGHEST BYTE
 1210         SBC DIVISOR
 1220         BCS .2            QUOTIENT BIT = 1
 1230         ASL OVERFLOW      TRUE QUOTIENT BIT
 1240         BCC .3
 1250  .2     STY DIVIDEND+1    QUOTIENT BIT = 1
 1260         STA DIVIDEND
 1270  .3     ROL DIVIDEND+3    SHIFT QUOTIENT BIT INTO END
 1280         ROL DIVIDEND+2    AND MOVE TO NEXT POSITION
 1290         ROL DIVIDEND+1
 1300         ROL DIVIDEND
 1310         DEX
 1320         BNE .1
 1330         ROR DIVIDEND      SHIFT REMAINDER BACK IN PLACE
 1340         ROR DIVIDEND+1
 1350         ROR OVERFLOW      SET SIGN BIT IF OVERFLOW
 1360  *--------------------------------
 1370  TRACE  LDA #$B0
 1380         BIT OVERFLOW
 1390         BPL .1
 1400         LDA #$B1
 1410  .1     JSR MON.COUT
 1420         LDY #0
 1430  .2     LDA #$A0
 1440         JSR MON.COUT
 1450         LDA DIVIDEND,Y
 1460         JSR MON.PRHEX
 1470         INY
 1480         CPY #4
```

```
1490        BCC .2
1500        JSR MON.CROUT
1510        RTS
1520  *--------------------------------
1530        .LIF
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.Div.8.4.txt
==============================================================================

 1000  *SAVE S.DIV.8.BY.4
 1010  *--------------------------------
 1020  *      DIVIDE 8-BIT VALUE
 1030  *          BY 4-BIT VALUE
 1040  *--------------------------------
 1050  DIVIDEND   .EQ 0
 1060  DIVISOR    .EQ 1
 1070  QUOTIENT   .EQ 2
 1080  *--------------------------------
 1090  S.DIV.8.BY.4
 1100         LDY #5       COUNT OFF 5 STEPS
 1110         LDA #0
 1120         STA QUOTIENT
 1130         LDA DIVISOR       SEE IF DIVISOR IN RANGE
 1140         BEQ .3            DIVIDE BY ZERO IS ILLEGAL
 1150         ASL          SHIFT DIVISOR TO LEFT NYBBLE
 1160         ASL
 1170         ASL
 1180         ASL
 1190         STA DIVISOR
 1200  .1     LDA DIVIDEND      COMPARE DIVIDEND TO DIVISOR
 1210         SEC
 1220         SBC DIVISOR
 1230         BCC .2            DIVIDEND IS SMALLER
 1240         CMP DIVISOR       SEE IF STILL LARGER
 1250         BCS .3            YES, OVERFLOW
 1260         SEC               SET QUOTIENT BIT = 1
 1270         STA DIVIDEND
 1280  .2     ROL QUOTIENT      SHIFT QUOTIENT BIT IN
 1290         LSR DIVISOR       SHIFT DIVISOR OVER
 1300         DEY
 1310         BNE .1            DO NEXT STEP
 1320         ROL DIVISOR  RESTORE DIVISOR
 1330         RTS
 1340  .3     BRK          DIVIDE FAULT
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.Divide.32.16.txt
==============================================================================

 1000  *SAVE S.DIVIDE 32/16
 1010  *--------------------------------
 1020  DIVIDE LDX #17          16-BIT DIVISOR
 1040         CLC              START WITH NO OVERFLOW
 1050  .1     ROR OVERFLOW
 1060         SEC
 1070         LDA DIVIDEND+1   NEXT-TO-HIGHEST BYTE
 1080         SBC DIVISOR+1    LEAST SIGNIFICANT BYTE
 1090         TAY              SAVE RESULT
 1100         LDA DIVIDEND     HIGHEST BYTE
 1110         SBC DIVISOR
 1120         BCS .2           QUOTIENT BIT = 1
 1130         ASL OVERFLOW     TRUE QUOTIENT BIT
 1140         BCC .3
 1150  .2     STY DIVIDEND+1   QUOTIENT BIT = 1
 1160         STA DIVIDEND
 1170  .3     ROL DIVIDEND+3   SHIFT QUOTIENT BIT INTO END
 1180         ROL DIVIDEND+2   AND MOVE TO NEXT POSITION
 1190         ROL DIVIDEND+1
 1200         ROL DIVIDEND
 1210         DEX
 1220         BNE .1
 1230         ROR DIVIDEND     SHIFT REMAINDER BACK IN PLACE
 1240         ROR DIVIDEND+1
 1250         ROR OVERFLOW     SET SIGN BIT IF OVERFLOW
 1260         RTS
 1270  *--------------------------------
 1280  DIVIDEND   .BS 4
 1290  REMAINDER  .EQ DIVIDEND
 1300  QUOTIENT   .EQ DIVIDEND+2
 1310  DIVISOR    .BS 2
 1320  OVERFLOW   .BS 1
 1330  *--------------------------------
 1340         .LIF
```

```
==============================================================
DOCUMENT :AAL-8302:DOS3.3:S.LinnsVidex.txt
==============================================================

1000  *---------------------------------------
1010  SCM.INSTALL .EQ SCM.BASE+$52A
1020  *
1030  MY.ESC.L
1040         CPX #0          CURSOR AT BEGINNING?
1050         BEQ .1          YES, CONTINUE
1060         JMP SCM.ESC.L   NO, LET S-C HANDLE IT
1070  .1     LDA #0          CONNECT DOS
1080         STA $AA52       BY SETTING INTERCEPT STATE = 0
1090         LDA #$84        SEND A CTRL-D
1100         JSR MON.COUT
1110  .2     LDA LOADCMD,X
1120         JSR SCM.INSTALL
1130         JSR FAKE.COUT
1140         CPX #6
1150         BCC .2
1160  .3     STX $406        SAVE CHAR POS'N
1170         JSR GETCH       GET SCREEN CHAR
1180         LDX $406        RESTORE POS'N
1190         JSR SCM.INSTALL
1200         JSR FAKE.COUT
1210         CPX #40         40 CHARS SENT YET?
1220         BNE .3          NO, LOOP BACK
1230         JMP CLREOP      CLEAR TO END OF PAGE
1240  *                      AND EXIT
1250  *
1260  LOADCMD    .AS -/LOAD  /
1270  *---------------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.MACRO.MACROS.txt
==============================================================================

 1000  *SAVE S.MACRO.MACROS
 1010         .MA BLD
 1020         ]1
 1030         ]2
 1040         ]3
 1050         ]4
 1060         .EM
 1070         >BLD ".MA ATOB","LDA A","STA B",".EM"
 1080         >BLD ".MA BTOA","LDA B","STA A",".EM"
 1090  *--------------------------------
 1100  A      .BS 1
 1110  B      .BS 1
 1120  *--------------------------------
 1130         >ATOB
 1140         >BTOA
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.ScreenPrinter.txt
==============================================================================

 1000  *SAVE S.SCREEN PRINTER
 1010  *--------------------------------
 1020  *      INSTANT HARDCOPY PROGRAM
 1030  *      BY ULF SCHLICHTMANN
 1040  *--------------------------------
 1050  SLOT   .EQ 1
 1060  BASL   .EQ $28
 1070  BASH   .EQ $29
 1080  *--------------------------------
 1090  COLUMNS    .EQ $678
 1100  DOS.REHOOK .EQ $03EA
 1110  AS.VTAB    .EQ $F25A
 1120  MON.PR     .EQ $FE95
 1130  MON.CROUT  .EQ $FD8E
 1140  MON.COUT   .EQ $FDED
 1150  MON.SETVID .EQ $FE93
 1160  *--------------------------------
 1170         .OR $300
 1180  HCOPY  LDA #SLOT    SET UP OUTPUT VECTOR
 1190         JSR MON.PR   TO POINT AT PRINTER
 1200         JSR MON.CROUT     START A NEW LINE
 1210         STA COLUMNS+SLOT  DISABLE SCREEN
 1220         LDX #0       START AT TOP OF SCREEN
 1230  .1     JSR AS.VTAB  COMPUTE BASE ADDRESS
 1240         LDY #0       START IN COLUMN 1
 1250  .2     LDA (BASL),Y NEXT CHARACTER FROM THIS LINE
 1260         JSR MON.COUT
 1270         INY
 1280         CPY #40      END OF LINE YET?
 1290         BNE .2       NO
 1300         JSR MON.CROUT
 1310         INX          NEXT LINE
 1320         CPX #24      END OF SCREEN YET?
 1330         BNE .1       NO
 1340         JSR MON.SETVID
 1350         JMP DOS.REHOOK
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.ScrnPrntrPlus.txt
==============================================================================
```

```
 1000  *SAVE S.SCREEN PRINTER.PLUS
 1010  *---------------------------------
 1020  *      INSTANT HARDCOPY PROGRAM
 1030  *      BY ULF SCHLICHTMANN
 1040  *---------------------------------
 1050  SLOT   .EQ 1
 1060  BASL   .EQ $28
 1070  VLINE  .EQ $FC
 1080  *---------------------------------
 1090  FLAGS      .EQ $7F8
 1100  DOS.REHOOK .EQ $03EA
 1110  MON.VTAB   .EQ $FC22
 1120  MON.VTABZ  .EQ $FC24
 1130  MON.PR     .EQ $FE95
 1140  MON.CROUT  .EQ $FD8E
 1150  MON.COUT   .EQ $FDED
 1160  MON.SETVID .EQ $FE93
 1170  MON.DASH   .EQ $FD9E
 1180  *---------------------------------
 1190         .OR $300
 1200  HCOPY  LDA #SLOT    SET UP OUTPUT VECTOR
 1210         JSR MON.PR   TO POINT AT PRINTER
 1220         JSR MON.CROUT     START A NEW LINE
 1230         LDA FLAGS+SLOT
 1240         AND #$BF
 1250         STA FLAGS+SLOT
 1260         JSR DASH.LINE
 1270         LDX #0       START AT TOP OF SCREEN
 1280  .1     TXA
 1290         JSR MON.VTABZ   COMPUTE BASE ADDRESS
 1300         LDA #VLINE
 1310         JSR MON.COUT
 1320         LDY #0       START IN COLUMN 1
 1330  .2     LDA (BASL),Y NEXT CHARACTER FROM THIS LINE
 1340         ORA #$80     BE SURE IN RANGE FOR PRINTING
 1350         CMP #$A0
 1360         BCS .3
 1370         LDA #$A0     PRINT SPACE IN PLACE OF ILLEGALS
 1380  .3     JSR MON.COUT
 1390         INY
 1400         CPY #40      END OF LINE YET?
 1410         BNE .2       NO
 1420         LDA #VLINE
 1430         JSR MON.COUT
 1440         JSR MON.CROUT
 1450         INX          NEXT LINE
 1460         CPX #24      END OF SCREEN YET?
 1470         BNE .1       NO
 1480         JSR DASH.LINE
```

```
1490        JSR MON.VTAB RE-ESTABLISH CURSOR POSITION
1500        JSR MON.SETVID
1510        JMP DOS.REHOOK
1520 *--------------------------------
1530 DASH.LINE
1540        LDY #42
1550 .1     JSR MON.DASH
1560        DEY
1570        BNE .1
1580        JMP MON.CROUT
1590 *--------------------------------
```

```
=======================================================================
DOCUMENT :AAL-8302:DOS3.3:S.SuperStrAddr.txt
=======================================================================

 1000  *SAVE S.SUPER STRING ADDER
 1010  *--------------------------------
 1020  *       STRING ADDITION:  & +$,A$,B$,C$
 1030  *--------------------------------
 1040  BUFFER              .EQ $200 - $2FF
 1050  AMPERSAND.VECTOR    .EQ $3F5 - $3F7
 1060  AS.CHRGET           .EQ $00B1
 1070  AS.SYNERR           .EQ $DEC9
 1080  AS.PTRGET           .EQ $DFE3
 1090  AS.CHKCOM           .EQ $DEBE
 1100  AS.GETSPA           .EQ $E452
 1110  AS.MOVSTR           .EQ $E5E2
 1120  *--------------------------------
 1130         .OR $9000
 1140         .TF B.STRING ADDER
 1150  *--------------------------------
 1160  SETUP  LDA #$4C     JMP OPCODE
 1170         STA AMPERSAND.VECTOR
 1180         LDA #STRADD
 1190         STA AMPERSAND.VECTOR+1
 1200         LDA /STRADD
 1210         STA AMPERSAND.VECTOR+2
 1220         RTS
 1230  *--------------------------------
 1240  FRESPC              .EQ $71,72
 1250  VARPNT              .EQ $83,84
 1260  *--------------------------------
 1270  *       TWO SIMILAR BLOCKS, FOR A$ AND B$
 1280  *       REFERENCED WITH X=0 OR X=4
 1290  *--------------------------------
 1300  A.LENGTH            .BS 1
 1310  A.ADDR              .BS 2
 1320  A.DIGITS.AFTER      .BS 1
 1330  *
 1340  B.LENGTH            .BS 1
 1350  B.ADDR              .BS 2
 1360  B.DIGITS.AFTER      .BS 1
 1370  *--------------------------------
 1380  *       A THIRD BLOCK, NEARLY THE SAME AS ABOVE,
 1390  *       FOR C$:  REFERENCED WITH X=8
 1400  *--------------------------------
 1410  C.LENGTH            .BS 1
 1420  C.STRING            .BS 2
 1430  *--------------------------------
 1440  CARRY               .BS 1
 1450  TENS.FLAG           .BS 1
 1460  C.ADDR              .BS 2
 1470  MAX.DIGITS.BEFORE   .BS 1
 1480  MAX.DIGITS.AFTER    .BS 1
```

```
1490  *-------------------------------
1500  *      & BRANCHES HERE
1510  *-------------------------------
1520  STRADD CMP #$C8     CHECK FOR "+$,"
1530         BNE .1
1540         JSR AS.CHRGET
1550         CMP #'$
1560         BNE .1
1570         JSR AS.CHRGET
1580         CMP #',
1590         BEQ .2
1600  .1     JMP AS.SYNERR
1610  *-------------------------------
1620  .2     LDX #0       POINT AT A$ DATA
1630         JSR PARSE.STRING.NAME   FIRST OPERAND
1640         LDX #4       POINT AT B$ DATA
1650         JSR PARSE.STRING.NAME   SECOND OPERAND
1660         JSR AS.CHKCOM           RESULT STRING
1670         JSR AS.PTRGET
1680         STY C.STRING+1          ADDRESS OF VARIABLE
1690         STA C.STRING
1700  *-------------------------------
1710  *      SCAN BOTH STRINGS TO DETERMINE BUFFER PARAMETERS
1720  *-------------------------------
1730         LDX #0       POINT AT A$ DATA
1740         JSR SCAN     GET Y=LEFT LENGTH, X=RIGHT LENGTH
1750         STX A.DIGITS.AFTER
1760         STX MAX.DIGITS.AFTER
1770         STY MAX.DIGITS.BEFORE
1780         LDX #4       POINT AT B$ DATA
1790         JSR SCAN     GET Y=LEFT LENGTH, X=RIGHT LENGTH
1800         STX B.DIGITS.AFTER
1810         CPX MAX.DIGITS.AFTER
1820         BCC .3
1830         STX MAX.DIGITS.AFTER
1840  .3     CPY MAX.DIGITS.BEFORE
1850         BCC .4
1860         STY MAX.DIGITS.BEFORE
1870  *-------------------------------
1880  *      CLEAR THAT MUCH OF THE BUFFER
1890  *-------------------------------
1900  .4     INC MAX.DIGITS.BEFORE  TWO MORE CHARS FOR
1910         INC MAX.DIGITS.BEFORE    SIGN AND CARRY
1920         CLC
1930         LDA MAX.DIGITS.BEFORE  TOTAL LENGTH OF RESULT
1940         ADC MAX.DIGITS.AFTER
1950         STA C.LENGTH
1960         TAY
1970         LDA #0       ZERO THE BUFFER FOR USE AS AN
1980  .5     STA BUFFER-1,Y    ACCUMULATOR
1990         DEY
2000         BNE .5
2010  *-------------------------------
2020  *      ADD A$ TO BUFFER
```

```
2030  *--------------------------------
2040         LDX #0       POINT AT A$ DATA
2050         JSR ADD.TO.BUFFER
2060  *--------------------------------
2070  *      ADD B$ TO BUFFER
2080  *--------------------------------
2090         LDX #4       POINT AT B$ DATA
2100         JSR ADD.TO.BUFFER
2110  *--------------------------------
2120  *      CONVERT BUFFER TO ASCII AGAIN
2130  *--------------------------------
2140         LDA BUFFER   SEE IF NUMBER IS NEGATIVE
2150         CMP #5       SET CARRY IF NEGATIVE, ELSE CLEAR
2160         ROR          MAKE A=0XXXXXXX OR 1XXXXXXX
2170         STA CARRY        TO SET OR CLEAR THESE FLAGS
2180         STA TENS.FLAG    APPROPRIATELY
2190         LDX C.LENGTH
2200         BEQ .10      FINISHED
2210  .6     LDA BUFFER-1,X
2220         CMP #'.
2230         BEQ .9
2240         BIT TENS.FLAG
2250         BPL .8
2260         ASL CARRY
2270         LDA #10
2280         SBC BUFFER-1,X
2290         CMP #10
2300         BCC .7
2310         SBC #10
2320  .7     ROR CARRY
2330  .8     ORA #'0
2340  .9     STA BUFFER-1,X
2350         DEX
2360         BNE .6
2370  .10    BIT TENS.FLAG   SEE ABOUT FINAL SIGN
2380         BPL .11         VALUE IS POSITIVE
2390         LDA #'-         NEGATIVE, SO STUFF "-"
2400         STA BUFFER        IN FRONT OF BUFFER
2410  .11    JSR CHOP.OFF.LEADING.ZEROES
2420  *--------------------------------
2430  *      PUT (BUFFER) IN OUTPUT STRING
2440  *--------------------------------
2450         LDX #8       POINT AT C$ DATA
2460         JSR SETUP.OPERAND
2470         JSR AS.GETSPA
2480         LDY #0
2490         STA (VARPNT),Y
2500         INY
2510         LDA FRESPC
2520         STA (VARPNT),Y
2530         INY
2540         LDA FRESPC+1
2550         STA (VARPNT),Y
2560         LDY C.ADDR+1
```

```
2570        LDX C.ADDR
2580        LDA C.LENGTH
2590        JMP AS.MOVSTR
2600  *--------------------------------
2610  *      ADD STRING TO BUFFER
2620  *      ENTER WITH X=0 FOR A$, X=4 FOR B$
2630  *--------------------------------
2640  ADD.TO.BUFFER
2650        JSR SETUP.OPERAND
2660        TAY           STRING LENGTH
2670        LDA A.DIGITS.AFTER,X
2680        PHA
2690        LDX #0
2700        LDA (VARPNT,X)  CHECK FOR MINUS SIGN
2710        CMP #'-
2720        BEQ .1        YES, CARRY SET
2730        CLC           ELSE CLEAR CARRY
2740  .1    ROR           MAKE A=0XXXXXXX OR 1XXXXXXX
2750        STA TENS.FLAG  MAKE FLAGS<0 IF MINUS
2760        STA CARRY
2770  *--------------------------------
2780        CLC           POINT INTO BUFFER WHERE OPERAND
2790        PLA                ALIGNS
2800        ADC MAX.DIGITS.BEFORE
2810        TAX
2820  *--------------------------------
2830  .2    TXA           TEST X FOR BEGINNING OF BUFFER
2840        BEQ .8        YES, FINISHED!
2850        DEX           NO, BACK ANOTHER ONE
2860        TYA           CHECK OPERAND POINTER
2870        BEQ .3        END OF OPERAND, BUT WE
2880  *                     STILL NEED TO FINISH CARRIES
2890        DEY           BACK UP IN OPERAND
2900        LDA (VARPNT),Y  NEXT CHAR FROM OPERAND
2910        CMP #'.       DECIMAL POINT?
2920        BEQ .7        YES, SKIP OVER IT
2930        CMP #'-       MINUS SIGN?
2940        BNE .4        NO, MUST BE DIGIT
2950  .3    LDA #'0       ASCII ZERO THEN
2960  .4    AND #$0F      CONVERT ASCII TO BINARY
2970        BIT TENS.FLAG
2980        BPL .5        NOT 9'S COMPLEMENTING
2990        EOR #$FF
3000        CLC
3010        ADC #10       FORM 9'S COMPLEMENT
3020  .5    ASL CARRY     GET PREVIOUS CARRY INTO C-BIT
3030        ADC BUFFER,X
3040        CMP #10       SEE IF CARRY
3050        BCC .6        NO
3060        SBC #10       YES, BACK THIS DIGIT DOWN
3070  .6    ROR CARRY     SAVE CARRY FOR NEXT LOOP
3080  .7    STA BUFFER,X
3090        JMP .2
3100  .8    RTS
```

```
3110  *-------------------------------
3120  *      SCAN STRING
3130  *      ENTER WITH X=0 FOR A$, X=4 FOR B$
3140  *      RETURN WITH X = # DIGITS AFTER DECIMAL POINT
3150  *                        (COUNTING THE DECIMAL POINT)
3160  *                  Y = # DIGITS BEFORE DECIMAL POINT
3170  *                        (COUNTING SIGN IF ANY)
3180  *-------------------------------
3190  SCAN
3200         JSR SETUP.OPERAND
3210         LDY #0
3220         TAX
3230         BEQ .2       NULL STRING
3240  .1     LDA (VARPNT),Y
3250         CMP #'.      LOOKING FOR DECIMAL POINT
3260         BEQ .2
3270         INY
3280         DEX
3290         BNE .1
3300  .2     RTS
3310  *-------------------------------
3320  *      CHOP OFF LEADING ZEROES
3330  *-------------------------------
3340  CHOP.OFF.LEADING.ZEROES
3350         LDY #1       FIND FIRST NON-ZERO POSITION
3360  .1     LDA BUFFER,Y
3370         CMP #'0
3380         BNE .2
3390         INY
3400         CPY MAX.DIGITS.BEFORE
3410         BCC .1
3420         DEY
3430  .2     LDA BUFFER   SIGN, MAYBE
3440         CMP #'-
3450         BNE .3
3460         DEY
3470         STA BUFFER,Y
3480  .3     CLC
3490         TYA
3500         ADC #BUFFER
3510         STA C.ADDR
3520         LDA #0
3530         ADC /BUFFER
3540         STA C.ADDR+1
3550         SEC
3560         TYA
3570         EOR #$FF
3580         ADC C.LENGTH
3590         STA C.LENGTH
3600         RTS
3610  *-------------------------------
3620  *      PARSE STRING NAME, SET UP POINTER
3630  *-------------------------------
3640  PARSE.STRING.NAME
```

```
       TXA
       PHA
       JSR AS.CHKCOM
       JSR AS.PTRGET     GET SECOND STRING PNTR
       PLA
       TAX
       LDY #0
       LDA (VARPNT),Y    GET LENGTH
       STA A.LENGTH,X
       INY
       LDA (VARPNT),Y    GET ADDRESS OF DATA
       STA A.ADDR,X
       INY
       LDA (VARPNT),Y
       STA A.ADDR+1,X
       RTS
*---------------------------------
*      LOAD ADDRESS INTO VARPNT
*      X=0 FOR A$, X=4 FOR B$
*---------------------------------
SETUP.OPERAND
       LDA A.ADDR,X
       STA VARPNT
       LDA A.ADDR+1,X
       STA VARPNT+1
       LDA A.LENGTH,X
       RTS
*---------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8302:DOS3.3:S.TRAPPER.txt
=============================================================================

 1000  *SAVE S.TRAPPER
 1010  *-----------------------------------
 1020  *      TRAPPER, BY ALLEN MARSALIS
 1030  *-----------------------------------
 1040         .OR $9300
 1050         .TF B.TRAPPER
 1060  *-----------------------------------
 1070  RLEN   .EQ $1A      RESTRICTION STRING
 1080  RSTR   .EQ $1B      DESCRIPTOR
 1090  TEMPPT .EQ $52
 1100  LASTPT .EQ $53
 1110  FRESPC .EQ $71,72
 1120  HIMEM  .EQ $73,74
 1130  VARPNT .EQ $83,84
 1140  FACMO  .EQ $A0
 1150  *-----------------------------------
 1160  BUF    .EQ $200     INPUT BUFFER
 1170  AMPVEC .EQ $3F5     AMPERSAND VECTOR
 1180  STROBE .EQ $C010    KEYBOARD STROBE
 1190  *-----------------------------------
 1200  AS.FRMNUM  .EQ $DD67     EVALUATE NUMERIC FORMULA
 1210  AS.CHKSTR  .EQ $DD6C     REQUIRE STRING
 1220  AS.FRMEVL  .EQ $DD7B     EVALUATE GENERAL FORMULA
 1230  AS.CHKCLS  .EQ $DEB8     REQUIRE ")"
 1240  AS.CHKCOM  .EQ $DEBE     REQUIRE ","
 1250  AS.CHKOPN  .EQ $DEBB     REQUIRE "("
 1260  AS.SYNCHR  .EQ $DEC0     REQUIRE (A-REG)
 1270  AS.SYNERR  .EQ $DEC9     SYNTAX ERROR
 1280  AS.PTRGET  .EQ $DFE3     GET VARIABLE PNTR
 1290  AS.GETSPA  .EQ $E452     GET SPACE IN STRING AREA
 1300  AS.MOVSTR  .EQ $E5E2     COPY STRING DATA
 1310  AS.FRETMP  .EQ $E604     FREE TEMPORARY STRING
 1320  AS.CONINT  .EQ $E6FB     CONVERT FAC TO 8-BITS
 1330  *-----------------------------------
 1340  MON.CLREOL .EQ $FC9C     CLEAR TO END-OF-LINE
 1350  MON.RDKEY  .EQ $FD0C     READ A KEY
 1360  MON.COUT   .EQ $FDED     DISPLAY A CHARACTER
 1370  *-----------------------------------
 1380  SETUP  LDA #$4C     "JMP" OPCODE
 1390         STA AMPVEC
 1400         LDA #TRAPPER
 1410         STA AMPVEC+1
 1420         LDA /TRAPPER
 1430         STA AMPVEC+2
 1440         LDA #SETUP   SET HIMEM UNDER TRAPPER
 1450         STA HIMEM
 1460         LDA /SETUP
 1470         STA HIMEM+1
 1480         RTS
```

```
*-------------------------------
*       AMPERSAND COMES HERE
*-------------------------------
TRAPPER
        LDA #$84          "INPUT" TOKEN
        JSR AS.SYNCHR
        JSR AS.CHKOPN     "& INPUT ("
        JSR AS.FRMNUM     READ FIELD LENGTH PARAMETER
        JSR AS.CONINT     CONVERT TO 8-BIT VALUE
        STX FL            SAVE FIELD LENGTH
        JSR AS.CHKCOM     ","
        JSR AS.FRMEVL     GET RESTRICTION STRING
        JSR AS.CHKSTR
        JSR AS.CHKCOM     ANOTHER ","
        LDY #2            SAVE DESCRIPTOR
.1      LDA (FACMO),Y
        STA RLEN,Y
        DEY
        BPL .1
        LDA TEMPPT        DID FRMEVL MAKE A TEMP STRING?
        CMP #$56
        BCC .2            NO
        LDA LASTPT        YES, SO FREE THE TEMP
        LDY #0
        JSR AS.FRETMP
.2      LDA #0            INIT BUFFER INDEX
        STA BINDEX
*---UNDERSCORE INPUT FIELD-------
        LDA #$DF          UNDERLINE CHAR
        JSR PRINT.FIELD
        LDA #$88          BACKSPACE  TO BEGINNING AGAIN
        JSR PRINT.FIELD
*---READ A KEY-------------------
        BIT STROBE        DON'T ALLOW TYPE AHEAD
.3      JSR MON.RDKEY     READ NEXT KEY
        AND #$7F          INTERNAL FORM
        STA KEY           SAVE IT
*---BACKSPACE--------------------
        CMP #$08          BACKSPACE?
        BNE .22           NO
        LDA BINDEX        IGNORE AT BEGINNING OF LINE
        BEQ .21
        LDA #$88          YES, ECHO IT
        JSR MON.COUT
        LDA #$DF          REPLACE UNDERLINE
        JSR MON.COUT
        LDA #$88          BACKSPACE AGAIN
        JSR MON.COUT
        DEC BINDEX        BACK UP BUFFER TOO
.21     JMP .3
*---CARRIAGE RETURN--------------
.22     CMP #$0D          RETURN?
        BNE .23           NO
        JSR MON.CLREOL
```

```
2030        JSR AS.PTRGET     GET DESTINATION STRING
2040        JSR AS.CHKCLS     MUST HAVE ")" AT END
2050        LDA BINDEX        LENGTH OF INPUT LINE
2060        JSR AS.GETSPA     FIND ROOM FOR IT
2070        LDY #0            MOVE IN DESCRIPTOR
2080        STA (VARPNT),Y
2090        INY
2100        LDA FRESPC
2110        STA (VARPNT),Y
2120        INY
2130        LDA FRESPC+1
2140        STA (VARPNT),Y
2150        LDY /BUF          COPY DATA INTO STRING
2160        LDX #BUF
2170        LDA BINDEX
2180        JMP AS.MOVSTR     ...AND RETURN
2190 *---CHECK IF VALID KEY-----------
2200 .23    JSR CHECK.RESTRICTIONS
2210 *---CHECK VALIDITY AND ECHO------
2220        LDA KEY           GET KEY AGAIN
2230        LDA BINDEX
2240        CMP FL
2250        BCS .27           TOO FAR, ABORT KEY
2260        LDA NEW           IF NEW = FAIL, ABORT KEY
2270        BEQ .27           YES, ABORT KEY
2280        LDA KEY
2290        LDY BINDEX
2300        STA BUF,Y         PUT KEY INTO BUFFER
2310        INC BINDEX
2320        CMP #$20          IF KEY WAS CONTROL-KEY,
2330        BCS .26              THEN PRINT SPACE
2340        LDA #$20
2350 .26    ORA #$80
2360        JSR MON.COUT      ECHO
2370        JMP .3            NEXT KEY
2380 .27    LDA #$07          RING BELL
2390        BNE .26
2400 *--------------------------------
2410 CHECK.RESTRICTIONS
2420        LDA #0
2430        STA RINDEX        RINDEX = 0
2440        STA NEW           NEW = FAIL
2450        STA ANDOR         ANDOR = OR
2460        STA NOT           NOT = FALSE
2470 *---FETCH OPERATOR---------------
2480 .4     LDY RINDEX        IF RINDEX >= RLEN,
2490        CPY RLEN              THEN QUIT SCAN
2500        BCC .5            NOT YET
2510        RTS
2520 .5     LDA (RSTR),Y      FETCH OPERATOR
2530        INC RINDEX
2540 *---DETERMINE OPERATION----------
2550        CMP #'            IGNORE BLANKS
2560        BEQ .4
```

```
2570        CMP #'<          < = >, THEN FETCH OPERAND
2580        BEQ .10
2590        CMP #'>
2600        BEQ .10
2610        CMP #'=
2620        BEQ .10
2630        CMP #'A          "AND"
2640        BEQ .7
2650        CMP #'O
2660        BEQ .8
2670        CMP #'N          "NOT"
2680        BEQ .9
2690        JMP AS.SYNERR
2700 *---AND OPERATOR-----------------
2710 .7     LDA #'N
2720        JSR SYNSTR
2730        LDA #'D
2740        JSR SYNSTR
2750        LDA #1           SET AND OPERATOR
2760        STA ANDOR
2770        BNE .4           ...ALWAYS
2780 *---OR OPERATOR------------------
2790 .8     LDA #'R
2800        JSR SYNSTR
2810        LDA #0           SET OR OPERATOR
2820        STA ANDOR
2830        BEQ .4           ...ALWAYS
2840 *---NOT OPERATOR-----------------
2850 .9     LDA #'O
2860        JSR SYNSTR
2870        LDA #'T
2880        JSR SYNSTR
2890        LDA #1           SET NOT OPERATOR "TRUE"
2900        STA NOT
2910        BNE .4           ...ALWAYS
2920 *---FETCH OPERAND----------------
2930 .10    STA ROPR
2940        LDA #$27         CHECK FOR APOSTROPHE
2950        JSR SYNSTR
2960        LDY RINDEX
2970        LDA (RSTR),Y     GET OPERAND
2980        STA ROPD
2990        INC RINDEX
3000        LDA #$27         ANOTHER APOSTROPHE
3010        JSR SYNSTR
3020 *---EVALUATE RELATIONAL OPERATION
3030        LDA NEW
3040        STA LAST         LAST = NEW
3050        LDA #0           NEW = FAIL
3060        STA NEW
3070        LDY ROPR         OPERATOR
3080        LDA KEY          LATEST KEY
3090        CMP ROPD         COMPARE TO OPERAND
3100        BEQ .11          THEY ARE EQUAL
```

```
3110         BCC .12          KEY < OPERAND
3120         CPY #'>          KEY > OPERAND
3130         BEQ .13          SUCCESS!
3140         BNE .14          FAIL.
3150  .11    CPY #'=
3160         BEQ .13          SUCCESS
3170         BNE .14          FAIL
3180  .12    CPY #'<
3190         BNE .14          FAIL
3200  .13    LDA #1           FLAG SUCCESS
3210         STA NEW
3220  *---PERFORM NOT OPERATION--------
3230  .14    LDA NOT          IF NOT, TOGGLE NEW
3240         BEQ .17          NOT NOT
3250         LDA NEW
3260         EOR #1
3270         STA NEW
3280         LDA #0           CLEAR NOT
3290         STA NOT
3300  *---PERFORM AND/OR OPERATION-----
3310  .17    LDA LAST
3320         LDY ANDOR
3330         BEQ .18          OR
3340         AND NEW          AND
3350         STA NEW
3360         JMP .4
3370  .18    ORA NEW
3380         STA NEW
3390         JMP .4
3400  *--------------------------------
3410  SYNSTR STA HOLD     SAVE CHAR
3420  .1     LDY RINDEX
3430         LDA (RSTR),Y
3440         INC RINDEX
3450         CMP #'           IGNORE BLANKS
3460         BEQ .1
3470         CMP HOLD
3480         BEQ .2
3490         JMP AS.SYNERR
3500  .2     RTS
3510  *--------------------------------
3520  PRINT.FIELD
3530         LDY FL
3540  .1     JSR MON.COUT
3550         DEY
3560         BNE .1
3570         RTS
3580  *--------------------------------
3590  HOLD   .BS 1
3600  NOT    .BS 1
3610  ANDOR  .BS 1
3620  FL     .BS 1
3630  NEW    .BS 1
3640  LAST   .BS 1
```

```
3650  KEY    .BS 1
3660  BINDEX .BS 1
3670  RINDEX .BS 1
3680  ROPR   .BS 1
3690  ROPD   .BS 1
3700  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8302:DOS3.3:TEST.ARRAYS.txt
==============================================================================

```
dÜA$(7,9)7nA$(3,5)–"ABCDEFG":A$(2,3)–
"MNOPQRST"FxØX,A$(3,5)UåØX,A$(2,3)[»≤m“ÅJ–0¡7:ÅK–0¡9w◊A$–""â‹ÅI –
1¡¤(1) 5»5°ÊA$–A$»Á(¤(1) 26»65)®ÇI∫ı∫J"  "K"  "A$œ˙A$(J,K) –
A$:ÇK:ÇJŸØY,A$d
```

==============================================================================
DOCUMENT :AAL-8302:DOS3.3:Test.Str.Adder.txt
==============================================================================

```
d≤TEST &+$, A$, B$En£36864:∫Á(4)"BLOAD B.STRING
ADDER":å36864QxÑA$:ÑB$bÇØ»$,A$,B$,C$oå∫C$:´120d
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:TEST.TRAPPER.2.txt
==============================================================================

#d∫Á(4)"BLOAD B.TRAPPER":å37632)nóIxÑ"NUMBER OF CHARACTERS:
";NhÇÑ"RESTRICTION STRING: ";A$nå∫îÊØÑ(N,A$,B$):∫:∫"STRING:
'";B$;"'"úÎ∫:∫•´120
```

```
==============================================================================
DOCUMENT :AAL-8302:DOS3.3:TEST.TRAPPER.txt
==============================================================================
```

```
#d∫Á(4)"BLOAD B.TRAPPER":å37632=nó:∫"TRAPPER EXAMPLES"hxÉ "NOT<'A' AND
NOT>'Z'" , "ALPHA ONLY"èÇÉ " ='Y' OR ='N' " , "Y OR N ONLY"ºåÉ "
NOT<'0' AND NOT>'9'" , "DIGITS ONLY"ˆñÉ  " NOT<'0' AND NOT>'9' OR ='-'
" , "DIGITS OR DASH"      †É "."    »áA$:≠A$–"."ƒÄ       “áP$(
     ‹∫P$":  ";@     ÊØÑ(8,A$,B$):ñ30:∫B$I     ´200
```

=============================================================================
DOCUMENT :AAL-8303:Articles:AAL.INDEX.txt
=============================================================================

E

Index to Apple Assembly Line

================================================================================
DOCUMENT :AAL-8303:Articles:CROSS.AD.txt
================================================================================

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

| | | | |
|---|---|---|---|
| Motorola: | 6800/6801/6802 | now | $32.50 |
| | 6805 | now | $32.50 |
| | 6809 | now | $32.50 |
| | 68000 | now | $50 |
| Intel: | 8048 | now | $32.50 |
| | 8051 | now | $32.50 |
| | 8085 | soon | $32.50 |
| Zilog: | Z-80 | now | $32.50 |
| RCA: | 1802/1805 | soon | $32.50 |
| Rockwell: | 65C02 | now | $20 |

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

======================================================================
DOCUMENT :AAL-8303:Articles:Division.txt
======================================================================

Division...................................Bob Sander-Cederlof

Remembering long division in decimal can be hard enough, but visualizing it in binary and implementing it in 6502 assembly language is awesome! Study the following example, in which I divide an 8-bit value by a 4-bit value:

```
                 00110            6
             ----------         ---
       1101 ) 01010101      13 ) 85
   step A:   -0000              -78
              ----               --
              1010                7
   step B:    -0000
               ----
              10101
   step C:    - 1101
               -----
               10000
   step D:     - 1101
                -----
                 0111
   step E:      -0000
                 ----
                 0111   Remainder
```

In the binary version, I have not made any leaps ahead like we do in decimal. That is, I wrote out the steps even when the quotient digit = 0. Now let's see a program which divides an 8-bit value by a 4-bit value, just like the example above.

If you think this is a clumsy program, you may be right. Note that the loop runs five times, not four. This is because there are five steps, as you can see in the sample division above.

The first thing the program does is to clear the quotient value. In a 4-bit machine performing 8-bit by 4-bit division would yield a 4-bit quotient, so the top bits must be cleared. The rest of the bits will be shifted in as the division progresses.

Next the divisor is shifted up to the high nybble position, to align with the left nybble of the dividend. This is equivalent to step A in the example above. The loop running from line 1200 through line 1310 performs the five partial divisions.

If the divisor is zero, or if the first partial division proves that the quotient will not fit in four bits, the program branches to ".3". I put a BRK opcode there, but you would put an error message printer, or whatever.

To run the program above, I typed:

```
     :$0:55 0D N 800G 0.2
```

and Apple responded with:   0000- 07 0D 06

which means the remainder is 7, and the quotient is 6.

Dividing Bigger Values:

The following program will divide one two-byte value by another. The program assumes that both the dividend and the divisor are positive values between 0 and 65535. This program was in the original Apple II monitor ROM at $FB84, but is not present in the Apple II Plus and Apple //e ROMs.

As written, this program expects the XTNDL and XTNDH bytes to be zero initially. If they are not, a 32-bit by 16-bit division is performed; however, there is no error checking for overflow or divide fault conditions.

This program builds the quotient in the same memory locations used for the dividend. As the dividend is shifted left to align with the divisor (opposite but equivalent to the shifting done in the previous program), empty bits appear on the right end of the dividend register. These bit positions can be filled with the quotient as it develops.

Signed Division

With a few steps of preparation, we can divide signed values using an unsigned division subroutine. All we need to remember is the rule learned in high school: If numerator and denominator have the same sign, the quotient is positive; if not, the quotient is negative.

Double Precision, Almost:

What if I want to divide a full 32-bit value by a full 16-bit value? Both values are unsigned. The 32-bit dividend may have a value from 0 to 4294967295, and the divisor from 0 to 65535. All of the published programs I could find assume the leading bit of the dividend is zero, limiting the range to half of the above.

If the leading bit of the dividend is significant, a one bit extension is needed in the division loop. The following program implements a full 32/16 division.

Line 1020 sets up a 17-step loop, because the 16-bit divisor can be shifted to 17 different positions under the 32-bit dividend. To make it easier to understand the layout of bytes in memory, I departed from the usual low-byte-first-format in this program. I assume this time that the most significant bytes are first:

```
     Dividend:   $83A $83B $83C $83D
                 msb . . . . . . . lsb

     Divisor:    $83E $83F
                 msb...lsb
```

I also have written this program to feed the quotient bits into the least significant end of the dividend register, as the dividend shifts left. The remainder will be found in the left two bytes of the dividend register, and the quotient in the right two bytes.

Watching It All Work:

Not being quite clairvoyant, I wanted to see what was really happening inside the 32/16 division program. So I added some trace printouts by inserting "JSR TRACE" right after lines 1050 and 1250. I also moved the variables into page zero, to show how much memory that can save. (All memory references are changed from 3-byte instructions to 2-byte instructions.)

The trace program prints first the overflow extension bit. If this is "1" on the last line, the quotient is too large to fit in 16-bits. TRACE next prints the four hex-digits of the quotient, and lastly the remainder. A line is printed before each step, and at the end to show the final results.

Now here are the printouts for a few values of dividend and divisor.

==============================================================
DOCUMENT :AAL-8303:Articles:Front.Page.txt
==============================================================

$1.50

Volume 3 -- Issue 6 March, 1983

In This Issue...



S-C Macro Assembler Version 1.1

That's right, Version 1.1! I've added all the most-requested new features, corrected those few lingering problems, and it's almost ready. Look inside for more details.

A New Screen-Oriented Editor

Several people have asked about a screen-oriented editor for the S-C Macro Assembler. Well, Mike Laumer has come up with one for you. It runs with the Language Card version of the Macro Assembler, in the unused bank. I still prefer a line editor, but Bill is rapidly falling in love with the new screen editor. Now everyone has a choice! See Mike's ad inside.

65C02

Many of you have expressed an interest in the new Rockwell R65C02 microprocessor. Well, I still haven't heard any more than I mentioned a couple of months ago. We're as eager as you are to get a sample. We'll have a detailed report as soon as we know more.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8303:Articles:Garbage.Indic.txt
==============================================================================

Patching Applesoft for Garbage-Collection Indicator
...........Lee Meador

I wanted to know when (how often and how long) Applesoft was doing garbage collection.  The following patch will cause an inverse "!" to placed in the lower right hand corner of the screen whenever garbage collection takes place.

It is a little tricky to patch Applesoft, since it is in ROM!  The first step is to copy the ROMs into the language card RAM space (any slot 0 RAM card will do).  If you have an old Apple II with Integer BASIC on the mother board, you can do this by booting the DOS 3.3 Master.  Otherwise, here are the steps:

```
]CALL-151
*C081 C081
*D000<D000.FFFFM
```

Next you need to place some code inside the Applesoft image in the RAM card.  I chose to place the new code on top of the HFIND subroutine at $F5CB.  (The code from $F5CB through $F5FF is never used by Applesoft.)  Here is the routine I put there:

```
PATCH  PHA
       LDA #$21         INVERSE "!"
       STA $7F7         BOTTOM RIGHT CORNER
       PLA
       JSR GARBAG
       PHA
       LDA #$A0         BLANK BACK ON SCREEN CORNER
       STA $7F7
       PLA
       RTS
```

You also need to patch the existing "JSR GARBAG" inside Applesoft to jump to this new code.  Here are the patches in hex:

```
*C083 C083          write enable RAM card
*E47B:CB F5
*F5CB:48 A9 21 8D F7
*F5D0:07 68 20 84 E4 48 A9 A0
*F5E0:8D F7 07 68 60
*C080               write protect RAM card
*control-C
]run your program
```

Here is a little Applesoft program which generates a lot of garbage strings so you can see the patch in action:

```
100 DIM A$(100)
110 FOR I = 1 TO 100
120 FOR J = 1 TO 200 : A$(I) = A$(I) + "B" : NEXT
130 PRINT I, : NEXT
```

Try running the program with different HIMEM values, to see the different effects.

==============================================================
DOCUMENT :AAL-8303:Articles:IIe.Stuff.txt
==============================================================

More on the //e............................Bob Sander-Cederlof

1. Page Zero Usage:

Last month I erroneously reported that the new //e monitor used location $08 in page zero. It does not.

However, I was correct when I said the monitor now uses location $1F. It is possible that your programs conflict with this, and it is possible that some commercial programs conflict with this. For example, standard SWEET-16 uses $1F for half of its register 15, which is its PC-register.

If you disassemble the //e monitor at $FC9C (CLREOL, Clear to end of line), you will find a STY $1F a few lines down. This is the only visible place where $1F is used. However, there are some invisible ones lurking in the shadows of ROM.

2. The Shadow ROM:

By shadows, I mean the alternate ROM space which overlays the I/O slot ROMs. By switching the SLOTCX soft switch, the monitor turns on this shadow ROM; the rest of the code necessary in the new monitor is then accessible starting at $C100. At $FBB4 the new monitor saves the current status, disables interrupts and saves the status of the SLOTCX softswitch, and switches to the shadow ROM. Then it JMP's to $C100 with the Y-register indexing one of 9 or 10 functions.

The "shadow ROM" (my terminology, not Apple's) covers the address space from $C100-C2FF and $C400-C7FF. The space from $C300-$C3FF is also there, but it is always turned on in my //e. It holds the startup code for the 80-column card, and some memory management subroutines.

The space from $C100-C2FF contains the extra code for handling monitor functions in the //e. $C400-C7FF holds the self-test program that you initiate by pressing control-solid-apple-reset or control-both-apples-reset. (With both Apples, you get sound with the self-test.)

There is more ROM you switch in and out with another soft switch at $C800-CFFE. This holds the 80-column firmware.

3. Version ID Byte:

Location $FBB3 in the monitor identifies which type of Apple you have:

FBB3- 38 ... old Apple II
FBB3- EA ... Apple II Plus (Autostart Monitor)
FBB3- 06 ... Apple //e

This byte is now a permanent feature; Apple will continue to use it as an ID byte in the future. Art Schumer and Clif Howard published an extensive Version ID program in the February 1983 issue of Call APPLE. They listed two versions, one for use from DOS and one for use from Pascal.

4.

```
===============================================================================
DOCUMENT :AAL-8303:Articles:Macro.Macros.txt
===============================================================================
```

Macro Can Build Macros............................Mike Laumer

The S-C Macro Assembler can do a lot of things even its designer never dreamed of. The macro capability may be limited compared to mainframe systems, but it still has a lot of power.

A few days ago I got a bright idea that maybe you could even define macros inside macros, or write a macro that builds new macros. Lo and behold, it works! Here is what I tried:

```
1000        .MA BLD
1010        ]1
1020        ]2
1030        ]3
1040        ]4
1050        .EM
```

Notice that every line from the opcode field on is defined by a macro parameter. I called it with lines like this:

```
1060        >BLD ".MA ATOB","LDA A","STA B",".EM"
1070        >BLD ".MA BTOA","LDA B","STA A",".EM"
```

Here is how it all looks when you type ASM:

```
               1010        .MA BLD
               1020        ]1
               1030        ]2
               1040        ]3
               1050        ]4
               1060        .EM
0800-          1070        >BLD ".MA ATOB","LDA A","STA B",".EM"
               0000>        .MA ATOB
               0000>        LDA A
               0000>        STA B
               0000>        .EM
0800-          1080        >BLD ".MA BTOA","LDA B","STA A",".EM"
               0000>        .MA BTOA
               0000>        LDA B
               0000>        STA A
               0000>        .EM
               1090 *-------------------------------
0800-          1100 A      .BS 1
0801-          1110 B      .BS 1
               1120 *-------------------------------
0802-          1130        >ATOB
0802- AD 00 08 0000>        LDA A
0805- 8D 01 08 0000>        STA B
0808-          1140        >BTOA
```

```
0808- AD 01 08 0000>       LDA B
080B- 8D 00 08 0000>       STA A
```

I don't know whether this is really useful or not.... If you think of a way to use it that is significant, I'd like to hear from you!

```
==============================================================================
DOCUMENT :AAL-8303:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
Upgrade from Version 4.0 to MACRO.................................$27.50
Source code of Version 4.0 on disk................................$95.00
    Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// .........................................$100.00
    Preliminary version.  Call or write for details.

S-C Word Processor................................................$50.00
  As is, with fully commented source code.  Needs S-C Macro Assembler.

Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.

ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982
    QD#10: Jan-Mar 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).................. $79.00
Source Code for FLASH! Runtime Package............................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware)......................$45.00
The Incredible JACK!...............................(reg. $129.00)  $99.00

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                                100 or more:  25 cents each

ZIF Game Socket Extender..........................................$20.00
Ashby Shift-Key Mod...............................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
     Only Revision level 7 or later Apples.

Books, Books, Books..........................compare our discount prices!
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster.................($15.95)  $15.00
```

```
"Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
"Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
"Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
"Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
"What's Where in the Apple", Second Edition..........($24.95)  $23.00
"What's Where Guide" (updates first edition)..........($9.95)   $9.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

 Add $1 per book for US postage.  Foreign orders add postage needed.

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8303:Articles:Patch.4.68K.Asm.txt
==============================================================================

Required patch in 68000 Cross Assembler

John Wallace, of Two Rivers, Wisconsin, has reported a bug in the 68000 Macro Cross Assembler. If you have serial number 1-46, or 52 or 53, you need this patch. I recommend that you do the patching on a COPY of the original disk, just in case....

In a MOVEM.L instruction the last pair of bytes is garbled.

Patches for motherboard version:

1. Boot the Cross Assembler disk, and select option 1 from the menu to load the motherboard version.

2. Type in the following patch:

```
:$3B31:98   (it was 99)
```

3. Save the patched version:

```
:BSAVE S-C.ASM.MACRO.68000,A$1000,L$2D7F
```

Patches for RAM Card version:

1. Boot the Cross Assembler disk, and select option 2 from the menu to load the RAM card version.

2. Patch the correct byte as follows:

```
:MNTR
*C08B C08B D7AF:98     (it was 99)
```

3. Save the patched result:

```
*BSAVE S-C.ASM.MACRO.68000.LC2,A$D01C,L$0838
*C080
*3D0G
```

(The second patch is trickier, because the patch goes into the second bank.)

Sincerely,

Bob Sander-Cederlof

```
=============================================================================
DOCUMENT :AAL-8303:Articles:PtrGet.GetAryPt.txt
=============================================================================
```

All About PTRGET & GETARYPT................Bob Sander-Cederlof

Both Leo Reich and E. Melioli have asked for some clarification on how to pass array variables between Applesoft programs and assembly language programs. I hope this little article will be of some help to them.

The Variable Tables:

We need to start with a look at the structure of the Applesoft variable tables. There are two variable tables: one for simple variables, and the other for arrays. (You might turn to page 137 of the Applesoft Reference Manual now.) Entries in these tables include the variable names; some codes to distinguish real, integer, and string variables; and the value if numeric. String variables include the length of the string and the address of the string, but not the string itself.

The address of the start of the simple variable table is kept in $69,$6A. The next pair, $6B and $6C, hold the address of the end of the simple variable table plus one. This happens to also be the address of the beginning of the array variable table. The address of the end of the arrays plus one is kept in $6D,$6E. The actual string values may be inside the program itself, in the case of "string" values; or in the space between the top of the array variable table and HIMEM.

Here is a picture, with a few more pointers thrown in for good measure:

```
(73.74) -->    HIMEM

               <string values>

(6F.70) -->    String Bottom

               <free space>

(6D.6E) -->    Free Memory Bottom

               <arrays>

(6B.6C) -->    Array Variable Bottom

               <variables>

(69.6A) -->    Simple Variable Bottom

               <program>
```

```
(67.68) -->    Program Bottom
```

Inside an Array:

Let's look a little closer at the array variable space. Each array in there consists of a header part and a data part. The header part contains the name, flags to indicate real-integer- string, the offset to the next array, the number of dimensions, and each dimension. The data part contains all the numeric values for real or integer arrays, and all the string length-address pairs for string arrays.

Here is a picture of the header part:

```
Bytes  Contents
----------------------
0,1    Name of Array
2,3    Offset
4      # of dimensions
5,6    last dimension
...
x,y    first dimension
```

The sign bits in each byte of the name combine to tell what type of array variable this is. If both bytes are positive, it is a real array; if both are negative, it is integer. Contrary to what it says on page 137 of the Applesoft manual, if the 1st byte is positive and the 2nd byte negative it is a string array. The manual has it backwards.

The value in the offset can be added to the address of the first byte of the header to give the address of the first byte of the header of the next array (or the end of arrays if there are no more).

The number of dimensions is one byte, which obviously means no more than 255 dimensions per array. Oh well! In my sample below I assume that no more than 120 dimensions have been declared. If you try to declare more than that, you will see how hard it is.

The dimensions are stored in backward order, last dimension first. Why? Why not? It has to do with the order they are used in calculating position for an individual element. Each dimension is also one larger than you declare in the DIM statement, because subscripts start at 0.

The data part of an array consists of the elements ordered so that the first subscript changes fastest. That is, element X(2,10) directly follows element X(1,10) in memory. Integer array elements are two bytes each, with the high byte first. Note: this is just about the only place in all the 6502 kingdom where you will find highbytes first on 16-bit values!

Real array elements take five bytes each: one byte for exponent, and four for mantissa. String array elements take three bytes each: one

for length of the string, and two for the address of the string.
Note: the string array elements DO NOT hold the string data, but only the address and length of that data!

Getting to the Point:

There is a powerful and much-used subroutine in the Applesoft ROMs which will find a particular variable in the tables. It is called PTRGET, and starts at $DFE3. It is too complicated to fully explain here, but here is what it does:

1. Reads the variable name from the program text.
2. Determines whether the variable is a simple one or an array.
3. Searches the appropriate table for the name.
4. If the name is not found, create a variable of the approriate type (simple or array; integer, real, or string).
5. Return with the address of the variable in Y,A (high-byte in the Y-register, low-byte in the A-register) and also in $83,84.

That is usually what happens. Actually there are several different entry points and two control bytes which modify PTRGET's behavior depending on the caller's whims. DIMFLG ($XX) is set non-zero when called by the DIM-statement processor, and is otherwise cleared to zero. SUBFLG ($YY) has four different states:

$00 -- normal value
$40 -- when called by GTARYPT
$80 -- when called to process "DEF FN"
$C1-$DA -- when called to process "FN"

We are concerned with the two cases SUBFLG = 0 and SUBFLG = $40, with DIMFLG = 0. Since the point of this whole article is to clarify access to array variables, I will concentrate on the main entry at $DFE3 and the GETARYPT subroutine at $F7D9. $DFE3 sets SUBFLG = 0, while GETARYPT sets SUBFLG = $40.

When we want to find an individual element inside an array, we call PTRGET at $DFE3. When we want to find the whole array, we call GETARYPT at $F7D9. GETARYPT is used by the STORE and RECALL Applesoft statements (which you might not realize even exist, since their function is only of interest to cassette tape users!)

The "& X" calls in the following program use PTRGET to find an array element.

On the other hand, if we want to sort the array, or if we want to save it all on disk, or some other feat which requires seeing the whole thing at once, we need to call GETARYPT. Then we can even find out how many subscripts were used in the DIM statement, and what the value of each dimension is. GETARYPT returns with the starting address of the whole array in $9B and $9C (called LOWTR).

The "& Y" call in the program prints out the starting address and length of each string of a string array.

I hope that as you work through the descriptions and examples above they are of some help.

==============================================================================
DOCUMENT :AAL-8303:Articles:QD10.COVER.txt
==============================================================================

QUARTERLY DISK #10 contains all the source code from Volume 3, Issues 4-6 of the Apple Assembly Line newsletter. The files are formatted for either the S-C Assembler II Version 4.0 or the S-C Macro Assembler, on a 16-sector DOS 3.3 disk.

S.SUPER SCROLL GENERATOR -- Program to generate the fastest possible screen-scroll program. This program occupies 145 bytes and creates 5521 bytes of code.

S.FILENAME EDITOR -- Add this unique line editor to the popular CATALOG ARRANGER, or adapt it to your own programs.

S.STRING ADD -- A simple demonstration of adding numeric values as ASCII strings.

S.SUPER STRING ADDER, TEST STRING ADDER -- A complete string adder for Applesoft string variables. Here are the basics of 240-digit-precision arithmetic.

S.LINNS VIDEX PATCH -- Another new feature for the Videx 80-column version of the S-C Macro Assembler.

S.TRAPPER, TEST TRAPPER, & TEST TRAPPER 2 -- An Applesoft input tuner. Allows you to easily specify permissible input values, using relational and logical operators similar to IF ... THEN.

S.ARRAYS, TEST ARRAYS -- Here's how you can use the Applesoft routines PTRGET and GETARYPT to process Applesoft arrays from your assembly language program.

S.MACRO.MACROS -- A macro which generates macros! Can you think of a use for it?

S.SCREEN PRINTER & S.SCREEN PRINTER PLUS -- Routines to dump a text screen to an Epson MX-80 printer.

S.DIVIDE 8/4, S.DIVIDE 16/16, S.DIVIDE 32/16, & S.DIVIDE 32/16 WITH TRACE -- Demonstration routines to show binary division. There are different routines to handle several sizes of signed and unsigned values.

xxx 1983

==============================================================================
DOCUMENT :AAL-8303:Articles:Screen.Printer.txt
==============================================================================

Epson MX-80 Text Screen Dump.................Ulf Schlichtmann
West Germany

Here is a short machine language program I wrote some time ago when I was working on a data-base program. It permits you to make a hard copy of the Apple text screen. It was written for an Epson MX-80 with Epson's Apple II Interface kit type 2, but with just one slight modification it should work with any other printer or interface as well.

I thought readers of AAL might have a use for this, especially after seeing a similar program in NIBBLE (Vol. 3 No. 3 pages 147-148) that was over three times longer to produce exactly the same result! The authors of that program required 149 bytes, and even used self-modifying code. My routine is only 40 bytes long.

There is one difference: in the NIBBLE program KSWL,H is changed so that the routine will be invoked every time control-P is pressed; also the ampersand vector is set up to re-install the KSWL,H vector whenever needed. I don't need these features, but even when they are added my program is still only about 78 bytes long (and WITHOUT any self-modifying code!).

Lines 1180-1200 direct all following output to the printer, and is equivalent to the Applesoft statements:

```
     PR#1 : PRINT
```

Next I store $8D (left over from MON.CROUT) as the number of columns for the printer, since any number greater than 40 will disable output to the screen. If you have a different printer interface card, you may need to use a different location than $678+SLOT. It should be stated somewhere in the printer interface manual. This is the slight modification I mentioned earlier.

Then I use the Applesoft VTAB routine to calculate the base address for each line. The entry point I chose requires the X-register to be loaded with the number of the desired line (starting with zero for the top-most line). The base address will then be stored in BASL,H. [ Note that using AS.VTAB means that this program will only work if Applesoft is switched on. If you call this when the other memory bank is on, no telling what might happen! ]

Next I let Y run from 0 to 39 to pick up all the characters in that particular line via indirect addressing. Each character is immediately fed to the printer. Upon completing a line, I call MON.CROUT to cause the printer to print the line. When I have sent all 24 lines, I then redirect output to the CRT and rehook DOS (lines 1340-1350).

Of course, there are a lot of possibilities for adding features to my basic screen dumper. The next version below does not rely on the Applesoft version of VTAB, so it can be called even when the Applesoft image is switched out. I also draw a border around the screen image: a line of dashes above and below, and vertical lines up down both sides.

Instead of using $8D as a line length to turn off the screen output, I masked out the flag bit in $7F8+SLOT. This works in the Grappler and Grappler Plus interfaces, whereas the former method did not. (It is equivalent to printing control-I and letter-N.)

Further, I now restore the value of BASL,H at line 1490. Otherwise the value in CV ($25) and the address in BASL,H do not agree after printing the screen.

The last enhancement is at lines 1340-1370. Here I now convert characters from flashing and inverse modes to normal mode, or to blanks in some cases. You might want to arrange for a different mapping here, according to your own taste.

Even with all these enhancements, the program is still only 86 bytes long. The first version could be loaded anywhere without reassembly, because there are no internal references. The second version does have an internal JSR, so it would have to be reassembled to run at other locations, or modified to be made run-anywhere.

==============================================================================
DOCUMENT :AAL-8303:Articles:Short.Item.txt
==============================================================================

Optional Patch for TEXT/ Command...........Bob Sander-Cederlof

Several have asked how to patch the character output at the beginning of each line by the TEXT/ command. TEXT/ normally writes your source code as a text file with control-I in place of each line number.

At $1AAD in the mother-board version, or $DAAD in the language card version, you will find $88. This is control-I minus one. Put what every character you wish there, less one. For example, if you want a leading space on each line, put $1F in $1AAD and/or $DAAD.

Short Note About Prime Benchmarks.................Frank Hirai
West Lebanon, NH

About your faster primes articles (Vol 2 #1, Vol 2 #5, and Vol 3 #2).... If you go back to Jim Gilbreath's original BYTE article you will find that the times he lists are for TEN iterations. As such they are not unreasonable for Integer BASIC and Applesoft. When comparing times for your 6502 assembly language versions, remember to multiply by ten!

Even so, 1.83 seconds for 10 iterations using Anthony Brightwell's program in the Apple compares quite well against 1.12 seconds for 10 iterations in an 8 MHz Motorola 68000.

[ ...and wait till we try it on a Number Nine 6502 card at 3.6 MHz! Or with a 65C02! ]

==============================================================================
DOCUMENT :AAL-8303:Articles:ShortPrimeNotes.txt
==============================================================================

Short Note About Prime Benchmarks.................Frank Hirai
West Lebanon, NH

About your faster primes articles (Vol 2 #1, Vol 2 #5, and Vol 3 #2).... If you go back to Jim Gilbreath's original BYTE article you will find that the times he lists are for TEN iterations. As such they are not unreasonable for Integer BASIC and Applesoft. When comparing times for your 6502 assembly language versions, remember to multiply by ten!

Even so, 1.83 seconds for 10 iterations using Anthony Brightwell's program in the Apple compares quite well against 1.12 seconds for 10 iterations in an 8 MHz Motorola 68000.

[ ...and wait till we try it on a Number Nine 6502 card at 3.6 MHz! Or with a 65C02! ]

```
==========================================================================
DOCUMENT :AAL-8303:Articles:T.MACRO.MACROS.txt
==========================================================================

               1010        .MA BLD
               1020        ]1
               1030        ]2
               1040        ]3
               1050        ]4
               1060        .EM
0800-          1070        >BLD ".MA ATOB","LDA A","STA B",".EM"
               0000>        .MA ATOB
               0000>        LDA A
               0000>        STA B
               0000>        .EM
0800-          1080        >BLD ".MA BTOA","LDA B","STA A",".EM"
               0000>        .MA BTOA
               0000>        LDA B
               0000>        STA A
               0000>        .EM
               1090 *--------------------------------
0800-          1100 A      .BS 1
0801-          1110 B      .BS 1
               1120 *--------------------------------
0802-          1130        >ATOB
0802- AD 00 08 0000>        LDA A
0805- 8D 01 08 0000>        STA B
0808-          1140        >BTOA
0808- AD 01 08 0000>        LDA B
080B- 8D 00 08 0000>        STA A

SYMBOL TABLE

0800- A
0801- B

0000 ERRORS IN ASSEMBLY
```

=============================================================================
DOCUMENT :AAL-8303:Articles:Version1.1.txt
=============================================================================

S-C Macro Assembler Version 1.1

A new version of the S-C Macro Assembler is just about ready, and it's going to be great!

I have added many new features, corrected a few problems, and created a special version to take advantage of the extra features of the new Apple //e computer.  Here's a summary of the new items, so far:

New or Extended Features:

1.  The .HS directive now allows optional "." characters before and after each pair of hex digits.  (e.g., .HS ..12..34..AB)  This makes for easier counting of bytes, and allows you to put meaningful comments above or below the .HS lines.

2.  .DO--.FIN can now be nested to 63 levels, rather than just 8 levels.

3.  In EDIT command, the insert mode is now invoked by ^A (ADD), rather than ^I.  The TAB or ^I keys now perform a clear-to-tab function.  Skip-to-tab is still invoked by ^T.

4.  Comment lines may now begin with either "*" or ";".

5.  Added .SE directive, which allows re-definable symbols.

6.  Binary constants are now supported.  The syntax is "%11000011101" (up to 16 bits).

7.  ASCII literals with the high-bit set are now allowed, and are signified with the quotation mark:  LDA #"X generates A9 D8.  Note that a trailing "-mark is optional, just as is a trailing apostrophe with previous ASCII literals.

8.  Blanks are now compressed inside macro skeletons when they are added to the symbol table.  This saves about 30% of the space used by the skeletons.

9.  The TEXT/ <filename> command now outputs the current TAB character (default ctrl-I).  It used to put out control-I no matter what the current TAB character was.

10. During assembly, the assembler now protect $001F-$02FF and $03D0-$07FF, as well as MACLBL thru EOT and MACSTK thru $FFFF.

11. Now allow USER parameters to override memory protection.  $101C-101D contains lower bound, and $101E-101F contains the upper bound of an area the user wants to UN-PROTECT.  (The parameter for the starting

page of the symbol table has moved from $101D to $1021, or $D01D to $D021.)

12. Added .PH and .EP directives, to start and end a phase. With these directives you can assemble a section of code that is intended to be moved and run somewhere else, without having to create a separate Target File.

13. Added .DUMMY and .ED to start and end a dummy section.

14. The TAB character may now be set to any character, including non-control characters, if you so desire.

Fixes to Known Problems:

1. Eliminated endless loop which occurred when a character > "Z" was typed in column 1 as a command.

2. .TI now properly spaces at top of each page, and at beginning of symbol table.

3. .AS and .AT now assemble lower case properly.

4. Changed the way the relative branches are assembled, so that "*" is equal to the location of the opcode byte. It used to be the location offset byte, which was non-standard.

5. Now pass two errors emit the proper number of object bytes, so that false range errors are not indicated.

6. HIDE now performs MERGE prior to HIDE, in case you forgot to do so.

Features added in support of Apple //e:

1. The Apple //e version allows you to change between 80- and 40-column screens at will, using PR#3 to go to 80-columns, or ESC-^Q to go to 40-columns.

2. In both normal input and edit modes, the DELETE key acts like a backspace key. It is interpreted the same as a left arrow (^H).

And there's more! The release disk will now include 80-column versions of the assembler for the Videx, STB, and ... 80-column cards.

I haven't made up my mind yet about a new price, how we'll handle the upgrades, or how much the charge will be. We'll have the final details in AAL next month.

==============================================================================
DOCUMENT :AAL-8303:Articles:Version11Short.txt
==============================================================================

S-C Macro Assembler Version 1.1

A new version of the S-C Macro Assembler is just about ready, and it's going to be great!

I have added many new features, corrected a few problems, and created a special version to take advantage of the extra features of the new Apple //e computer.  Here's a summary of the new items, so far:

New or Extended Features:

1.  80-column support!  The release disk will now include versions for the Videx, STB, and maybe other 80-column cards.

2.  The .HS directive now allows optional "." characters before and after each pair of hex digits.  (e.g., .HS ..12..34..AB)  This makes for easier counting of bytes, and allows you to put meaningful comments above or below the .HS lines.

3.  .DO--.FIN can now be nested to 63 levels, rather than just 8 levels.

4.  Comment lines may now begin with either "*" or ";".

5.  Added .SE directive, which allows re-definable symbols.  Now a macro can tell how many times it has been called.

6.  Binary constants are now supported.  The syntax is "%11000011101" (up to 16 bits).

7.  ASCII literals with the high-bit set are now allowed, and are signified with the quotation mark:  LDA #"X generates A9 D8.  Note that a trailing "-mark is optional, just as is a trailing apostrophe with previous ASCII literals.

8.  The TEXT/ <filename> command now outputs the current TAB character (default ctrl-I).  It used to put out control-I no matter what the current TAB character was.

9.  Now allow USER parameters to override memory protection.  $101C-101D contains lower bound, and $101E-101F contains the upper bound of an area the user wants to UN-PROTECT.  (The parameter for the starting page of the symbol table has moved from $101D to $1021, or $D01D to $D021.)

10. Added .PH and .EP directives, to start and end a phase.  With these directives you can assemble a section of code that is intended to be moved and run somewhere else, without having to create a separate Target File.

11. Added .DUMMY and .ED to start and end a dummy section.

12. The TAB character may now be set to any character, including non-control characters, if you so desire.

Fixes to Known Problems:

1. .TI now properly spaces at top of each page, and at beginning of symbol table.

2. .AS and .AT now assemble lower case properly.

3. Changed the way the relative branches are assembled, so that "*" is equal to the location of the opcode byte. It used to be the location offset byte, which was non-standard.

4. Now pass two errors emit the proper number of object bytes, so that false range errors are not indicated.

Features added in support of Apple //e:

1. The Apple //e version allows you to change between 80- and 40-column screens at will, using PR#3 to go to 80-columns, or ESC-^Q to go to 40-columns.

2. In both normal input and edit modes, the DELETE key acts like a backspace key. It is interpreted the same as a left arrow (^H).

I haven't made up my mind yet about a new price, how we'll handle the upgrades, or how much the charge will be. We'll have the final details in next month's AAL.

==============================================================================
DOCUMENT :AAL-8303:Articles:VisibleCPU.txt
==============================================================================

Review: "The Visible Computer: 6502".....Bob Sander-Cederlof

For five years I have talked about it. "Someone should write a program that illustrates 6502 code being executed, using hi-res animation."

Software Masters never heard me, but they did it anyway! "The Visible Computer: 6502" is an animated simulation of our favorite microprocessor. You see inside the chip and watch the registers change, micro-step by micro-step. You even see the "hidden" registers: DL (data latch), DB (data buffer), IR (instruction register), and AD (address). You see HOW the instructions are executed.

I was amazed at the quality of the documentation. You get 140 pages of easy-to-follow, fun-to-read tutorial and reference text. The manual assumes only that you have an Apple, and are moderately familiar with Applesoft. It doesn't try to teach everything there is to know about machine language, but it does deliver the fundamental concepts.

Thirty demonstration programs are included on the disk, which progressively lead you through the instruction set. You begin with a two-byte register load, and work up to hi-res graphics and tone generation. All of the example programs are explained in detail in the manual. Of course, you can also trace your own programs or programs inside the Apple ROMs.

You can also use the simulator as a debugging tool, if your program will fit in the user memory area. The simulator provides a 1024-byte user memory, plus a simulated page zero and page one. You can also use $300-$3CF, if you wish. One unusual tool for debugging purposes is a full 4-function calculator mode, which works in binary, decimal, or hexadecimal.

Here is a list of the commands available at the normal level:

```
BASE           select binary, decimal, or hexadecimal
BLOAD          load a program to be simulated
BOOT           boot disk in slot 6, drive 1
CALC           turn on 4-function calculator
EDIT           short-cut entry of hex code into memory
ERASE          clear screen (so graphics can be seen)
L              disassemble five lines of code
LC             select memory for displayed in left column
PRINTER        turn on/off printer in slot one
RC             select memory for display in right column
RESTORE        restore normal screen display
STEP           select one of four simulation modes:
```

```
                 0 -- fastest, no display update until BRK
                 1 -- Full display, simulate until BRK
                 2 -- Full display, simulate one instruction
                      with no pause between steps
                 3 -- Full display, simulate one instruction,
                      pausing before each step

WINDOW           select one of three display options:
                 MEM:  window shows 16 memory cells
                 OPEN:  window is blank
                 CLOSE:  window shows "hidden" 6502 registers
<addr><value>    store value at memory address
<reg><value>     store value in register
```

A "MASTER" mode can be turned on, which enables more features and commands for experienced users. In the master mode you can use the REAL zero page, you can modify any location in memory (even the ones that are dangerous!), you can BLOAD and BSAVE on standard DOS 3.3 disks, and run previously checked subroutines at full 6502 speed.

I know that a lot of you are looking for some help in understanding assembly language; "The Visible Computer" may be just the help you need. Let your own Apple teach you! Some of you are teaching 6502 classes; "The Visible Computer" is the most helpful teaching tools I have ever seen.

I was gratified to learn that the author is an old customer! He used an older version of the S-C Assembler for coding the longer examples, and the assembly language portions of the simulator. We even got a free plug on page 108!

The normal retail price of "The Visible Computer" is $49.95, our price will be an even $45 to readers of Apple Assembly Line.

==============================================================================
DOCUMENT :AAL-8304:Articles:Circut.Desc.txt
==============================================================================

The Apple ][ Circuit Description:  A Review.........Bill Morgan

"Have you ever wanted to know the detailed circuit operation of your Apple ][ computer?  Perhaps you were designing a peripheral or making a modification.  Maybe you were repairing an Apple.  You may have just been curious about how it works."

That's the first paragraph of a new book called The Apple ][ Circuit Description, by Winston D. Gayler.  If the answer to that question is "yes", you need to look at this book.  Circuit Description contains about 160 pages of text describing the operation of every component on the Apple's motherboard and keyboard.  There are also 44 large fold-out pages of easy-to- read block diagrams, schematics, timing diagrams, and waveform drawings.  The enlarged, readable schematics alone will be worth the price of the book to some users!

One of the first things Mr. Gayler handles is identifying the various revisions of the Apple ][, from the original Rev. 0 through last year's RFI treated motherboard, Rev. D.  The body of the book covers that last version, while an appendix goes into the differences in all earlier revisions, and the diagrams show all revisions.  The very latest thing, the Apple //e, is not mentioned, since that's a radical departure from all others.

The book is intended for engineers, technicians, students, and serious hobbyists.  The descriptions, schematics, timing diagrams, and waveform drawings can be an invaluable help in designing peripherals and modifications, troubleshooting, studying practical circuit design, and just understanding how your Apple works.

Each chapter has two sections, Overview, and Detailed Circuit Description.  You can cruise the Overview sections to get an idea of what's going on in each piece of your Apple, or you can sit down with the Detailed Circuit Description, the schematics, your Apple, and your TTL Data Book, and figure out each and every signal in the computer.

Here is a chapter-by-chapter summary:

1. Introduction and overview of the book.

2. Block-diagram discussion of the whole computer's structure, introducing concepts like "address multiplexer" and "video address generator".  Apple's unique patented power supply is also covered here.

3. Clocks: the master oscillator, clock generator, and the horizontal portion of the video address generator.  Clocks are especially important in the Apple due to their interplay with the video circuitry.

4. The vertical portion of the video address generator and the sync, blanking, and color burst signals.

5. RAM memory, the 4116's and their addressing, as well as the shared access scheme for the video memory.

6. The 6502 processor and its internal cycles, including read cycles, write cycles, RAM and ROM cycles, I/O and keyboard cycles, interrupts, and DMA (direct memory access).

7. On-board I/O devices, including cassette I/O, the game port, the speaker, and the current two-piece keyboard.

8. Video generator hardware, how it creates TEXT, LORES, and HIRES displays under software control.

Appendices:

A. Introduction to standard video signal techniques, for those of us who know even less about video than about digital.

B. Various revisions of the Apple motherboard.  The main text of the book describes the RFI, Rev. D board.  This appendix covers the differences in all earlier boards, as well as the old one-piece keyboard.

C. Schematics.  Pages and pages of enlarged diagrams of all versions of the motherboard and keyboards.

In the Introduction, Gayler says that the reader should be familiar with TTL (gates, flip-flops, shift registers, and multiplexers) and should have a basic knowledge of micro- processor and microcomputer architecture.  Well, I have a very basic knowledge of architectures, and almost no familiarity with TTL details.  This book looks like it will be a great tool for learning about TTL, because I will be able to relate what the data books say about a chip to a knowledge of what that chip is doing in my very own Apple.

One thing I would like to see is a sort of cross-reference by motherboard coordinate.  It would be nice to be able to ask the book "What is the function of that 74LS20 at location D2?"  As it is, I had to look through several foldouts for a chip symbol labelled "D2".  It is a NAND gate in "Fig. C-2. Clock Generator (all revisions)"  Since it's part of a clock circuit, it must be covered in chapter 3.  Several minutes of poking around in chapter 3 tells me that chip is part of one of the Apple's most unique features!  Every 65th CPU cycle is slightly stretched (1117 us vs. 978 us) to maintain sync with the color signals, and D2 is responsible for triggering that stretch.

That last paragraph started out to describe a shortcoming of the book, and turned into yet another example of the kind of great information contained in The Apple ][ Circuit Description.  If you're doing any hardware work with the Apple, or if you want to learn more about what's going on in there, you need this book.

The Apple ][ Circuit Description, by Winston D. Gayler. Published by Howard W. Sams. 8 1/2 by 11 comb binding. 172 pp. text, 44 fold-out diagrams. Shipping weight 3 lbs. List price is $22.95, our price will be $21 + shipping ($2 domestic, $12 overseas).

=============================================================================
DOCUMENT :AAL-8304:Articles:Disasm.Patches.txt
=============================================================================

DISASM and the //e..................................Bill Morgan

Yesterday afternoon I received two phone call in less than 30 minutes, both reporting that RAK-Ware's disassembler, DISASM, does not work on the Apple //e. The problem occurs when DISASM calls a non-standard entry into the monitor HOME routine. At several places in the routines to enter address information Bob Kovacs used $FC5A for a sort of combination VTAB and Clear-to- End-of-Page. Well, that won't work on a //e. The following patches change all the calls to $FC5A into $FC58, or the standard HOME routine. This will change the behavior of the program a little, making the screen clear between entries, rather than just tab down, but the program should now work.

```
84C:58     94D:58     A79:58
AD8:58     BBA:58     BFB:58
```

=====================================================================
DOCUMENT :AAL-8304:Articles:Fast.DOS.Patch.txt
=====================================================================

Patch DOS 3.3 for Fast LOAD and BLOAD......Bob Sander-Cederlof

There must be at least a dozen products on the market now to speed up DOS 3.3: Diversi-DOS, David-DOS, The DOS Enhancer, QuickDOS, FastDOS, Hyper-DOS, et cetera. Some of these are unfortunately not compatible with the everyday programs we like to use, such as the S-C Assembler, ES-CAPE, or our favorite word processor. And it can be quite difficult sometimes to determine the degree of compatibity.

For the record, S&H Software's DOS Enhancer is completely compatible with the S-C Macro Assembler. David-DOS works well until you try to use the .TF directive.

Most of the speed-up systems only improve the speed of LOAD, BLOAD, RUN, BRUN, SAVE, and BSAVE. Some also speed up booting into the language card. And two (Diversi-DOS and David-DOS) speed up READing and WRITE-ing TEXT files, as well as offering a lot of minor enhancements in pursuit of more "user- friendliness".

It seems that the more the speed-up system does, the more compatibility problems you can expect. After all, to add a feature you do have to change some code. And many programs on the market expect the DOS image to be un-modified so they can jump into DOS subroutines in strange unexpected places and make their own custom patches to the DOS image.

Paul Schlyter (a subscriber in Sweden) sent me a small patch for DOS 3.3 early in April, 1982. Paul's patch speeds up only RUN, BRUN, LOAD and BLOAD, but it such a small patch that it will almost fit into the interstices (unused bytes) inside DOS. In fact, after I removed one bug and reorganized the code a little, I was able to fit it entirely within two unused areas: $BA69-BA95 and $BCDF-BCFF. I believe the result is completely compatible with all the programs I use around here, except for the ones that use their own modified and protected DOS.

Paul's patch turns out to be functionally equivalent to the much longer patch proposed in HardCore Magazine's HyperDOS, but it leaves the INIT command intact.

I ran some timing tests:

```
     LOAD   40 sectors  standard 10 sec
                        patched   3.5 sec

     BLOAD  37 sectors  standard 11 sec
                        patched   4 sec

     LOAD  132 sectors  standard 32 seconds
```

patched   7.5 seconds

I didn't try measuring times, but I suspect that SAVE and BSAVE may be just a little faster with this patch installed (during the read-after-write phase).

Since the S-C Assemblers use the LOAD command to process .IN directives, large assemblies with large included files will assemble about three times faster when you install this speed-up patch.

The patch is really rather simple.  But before examining the patch, let's review the normal flow inside DOS for LOADing and BLOADing.

DOS is constructed in three layers:  the outer layer accepts your commands from the keyboard or from your program.  The inner layer, called RWTS, handles the intimate details of reading or writing a specified sector on a specified track.  RWTS also does the raw disk initialization when you use the INIT command.  The layer between commands and RWTS is called the File Manager (FM).

The command layer calls FM to open, close, rename, lock, unlock, verify, or delete a file; to print a catalog; to initialize a disk; or to position within a file.  There are also four kinds of calls for reading and writing files, to read or write one byte or a range of bytes.

When you use the RUN or LOAD command, the command layer calls FM to read the first two bytes.  These bytes contain the length of your program.  For Integer BASIC or S-C Assembler source files, the length is subtracted from HIMEM to get a loading address.  The loading address for Applesoft programs is found in $67,68.  Then FM is called to read a range of bytes of that length, to be stored starting at the loading address just determined.

When you use the BRUN or BLOAD command, the first four bytes are read off the front of the file.  The first two bytes are the loading address, and the next two are the length.  (Of course, you can override the loading address with the "A" parameter after the file name.)

After winding our way through the front end of FM, we finally get to this subroutine (where the range is read):

```
                1000 READ.RANGE
AC96- 20 B5 B1  1010       JSR DECR.TEST.LENGTH
AC99- 20 A8 AC  1020       JSR READ.BYTE
AC9C- 48        1030       PHA              SAVE THE BYTE
AC9D- 20 A2 B1  1040       JSR GET.ADDRESS.INC
ACA0- A0 00     1050       LDY #0
ACA2- 68        1060       PLA              GET THE BYTE
ACA3- 91 42     1070       STA ($42),Y      STORE IN BUFFER
ACA5- 4C 96 AC  1080       JMP READ.RANGE
```

The subroutine DECR.TEST.LENGTH breaks out of this loop when the range has been completely read. The READ.BYTE subroutine picks bytes out of the DOS buffer, and reads a sector into that buffer when the buffer is empty.

To understand the speed-up patch, break the reading process into three parts: the first sector, the last sector, and all the in-between sectors. We will let the loop shown above handle the first sector and possibly the last sector, and read the in-between sectors using a faster method. Short files with only one or two data sectors will not have any in-between sectors, and so there will be no improvement in speed.

First we need to read the rest of the first sector of the file. The first two or four bytes were already read to get address and length information. We can let the loop shown above do that job. But we need a way to break into the loop when it is our turn. Let's patch the JMP on the last line to jump to our patch.

Our patch will get control after the loop above has read and stored a byte of data. At that time our patch can look at the current file position in $B5E6; if $B5E6 is non-zero, then there are still bytes in the DOS buffer. As long as there are bytes in the DOS buffer, we will branch back to $AC96 and let FM handle the bytes in its normal way.

Once the first sector has been read and stored, a byte at a time, $B5E6 will have a zero value. Then our patch can look at the remaining length. If the remainging length is at least one whole sector, we can read it faster. If not, FM can read the last partial sector in its normal fashion.

To read a sector faster, we bypass the DOS buffer. We can temporarily patch the actual destination address where the sector must go into the RWTS call block. RWTS can put the entire sector directly into its final destination, rather than into the DOS buffer to be later moved by the rather slow loop above.

The extra time saved by eliminating the middle man will save an entire revolution of the drive to get the next sector (if it is in the same track, and they usually are). A 40 sector file laid out sequentially on three tracks will save 38 revolutions of the disk. The disk spins at 5 revolutions per second, so we will save a hair over 7 seconds. (If the file is not laid out sequentially, the savings will be less.)

The bigger the file, the bigger the percentage improvement. We can save 3 seconds per track. It normally takes FM about 18 revolutions to read a track; with our patch, a track can be read in about 3 revolutions. We save 15 revolutions or 3 seconds on each full track. That is, a full track can be read in .6 seconds instead of 3.6 seconds. The rest of the time required to read the file is spent moving the head from track to track, and reading the catalog and VTOC sectors.

If all 16 sectors of a track are to be read, and if the sectors were allocated the normal DOS 3.3 way, I think this is the way it happens with my patch installed:

```
F    E   D   C   B   A   9   8    7   6   5   4   3   2   1    0
F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0
```

The bottom line of numbers shows the physical sector numbers.  As you move across the page from left to right, you simulate the disk read head.  It may take up to a full revolution of the disk before sector F appears, but once it does we proceed to pick off approximately every other sector as they come by.  The top line of numbers shows the DOS 3.3 logical sector numbers.  Logical sector E is actually physical sector 2, and so on.  So it takes two full revolutions, plus two more sectors, to read all 16.

If you are trying to figure out where the rest of the time is used, keep in mind that DOS first reads the VTOC (track 17, sector 0); then the first catalog sector (track 17, sector 15); if the file specified is not in the first catalog sector, it reads another; and so on.  If the file is far down in the catalog, it might have to read all 15 catalog sectors to find the file.  Then the track/sector list is read; it is usually in sector 15 of the same track containing the first 15 sectors of data.  On the other hand, as the disk fills up the sectors get splattered all over the disk.

Here is the patch code, arranged so that it squeezes into those two interstices I mentioned earlier:

To install the patches, you need to BLOAD PATCH1 and BLOAD PATCH2. Then patch locations $ACA6-7 to 69 BA, to change the JMP READ.RANGE instruction to a JMP PATCH1.  Note that you must BLOAD the patches before changing $ACA6-7.  If you change $ACA6-7 first, the system will crash as soon as you try to execute a BLOAD.

Here is an Applesoft program (which you could append to your HELLO program) to poke the patches into DOS.

```
20000  REM INSTALL FAST DOS LOAD AND BLOAD PATCHES
20010  READ N: IF N = 0 THEN  END
20020  READ A
20030  FOR I = 1 TO N: READ P: POKE A,P:A = A + 1: NEXT
20040  GOTO 20010
20100  DATA  44,47721,173,230,181,208,36,173,194,181,
       240,31,173,203,181,72,173,204,181,72,173,195,
       181,141,203,181,173,196,181,141,204,181,32,
       182,176,176,3,76,223,188,76,111,179,76,150,172
20110  DATA  33,48351,238,228,181,208,3,238,229,181,
       238,196,181,238,204,181,206,194,181,208,11,
       104,141,204,181,104,141,203,181,76,150,172,
       76,135,186
20120  DATA  2,44198,105,186
20130  DATA  0
```

Paul mentioned he was working on an equally simple patch to speed up SAVE and BSAVE, but I haven't heard any more from him on that subject.

==============================================================
DOCUMENT :AAL-8304:Articles:Front.Page.txt
==============================================================

$1.50

Volume 3 -- Issue 7 April, 1983

In This Issue...



New Goodies

We have several new products available this month. There are descriptions inside this issue of the new Cross Reference program for the S-C Macro Assembler, and the new book "Apple ][ Circuit Description". Also, the long-awaited RCA 1802 Cross Assembler is now ready, at $32.50.

Version 1.1 of the Macro Assembler is now ready to go! The upgrade from the current Version 1.0 will only cost you $12.50. That gets you //e, Videx, and STB 80-column support, 5 new directives and all the other new features described last month.

DISASM and the //e

Yesterday afternoon I received two phone call in less than 30 minutes, both reporting that RAK-Ware's disassembler, DISASM, does not work on the Apple //e. The problem occurs when DISASM calls an odd entry into the monitor HOME routine. At several places in the routines to enter address information Bob Kovacs used $FC5A for a sort of combination VTAB and Clear-to-End-of- Page. Well, that won't work on a //e. The following patches change all the calls to $FC5A into $FC58, or the standard HOME routine. This will change the behavior of the program a little, making the screen clear between entries, rather than just tab down, but the program should now work.

```
84C:58     94D:58     A79:58
AD8:58     BBA:58     BFB:58
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage

for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8304:Articles:Mikes.Stuff.txt
==============================================================================

New S-C Cross Reference Utility.....................Mike Laumer

At last a Cross Reference Utility is available for the S-C Assembler that is fully compatible with the latest releases of the S-C Macro Assembler. It handles all the new directives, shows macro calls, and can even give an optional cross reference on the opcodes! It only takes a few seconds to cross reference even a huge file and begin the listing! It is even faster than the Macro Assembler in processing the source lines.

The Cross Reference Utility also can optionally print a paginated source file listing before printing the cross reference. That way you can be certain that you have a program listing with the same line numbers shown in the cross reference listing.

The price is reasonable: only $20.00 for the object code version and $50.00 for both source and object code. What other company sells source code to their utilities!

FLASH! Compiler note:

The FLASH! Integer Basic Compiler was recently reviewed by PEELINGS magazine and received an A+. It is currently the highest rated Integer compiler (the competition is rated only A). The price? Just $79.00 ($70 less than the competition)!

S-C Word Processor note.............................Mike Laumer

We recently had one customer give us a great compliment on the S-C Word Processor. He has given up on WORDSTAR! He found that the S-C Word Processor can read and write large text files 20 times faster than WORDSTAR and that scrolling was much quicker. He can be in and out of the S-C Word Processor before WORDSTAR even lets him type a single key. The S-C Word Processor is also much less expensive than WORDSTAR and you don't have to buy a Z-80 card!

His only desire was to have an 80 column version of the Word Processor. However, that wouldn't be nearly so fast since SCWP re-writes the screen on every keystroke. I have noticed also that the 40 column display never causes me eye strain, but all the 80 column displays do.

Full Screen Editor for S-C Macro Assembler..........Bill Morgan

Laumer Research has recently introduced a new utility for the S-C Macro Assembler. This month seems to be the time for new utilities.

The Full Screen Editor is used with a language card and a 48K Apple. It runs in the spare 4K memory bank of a language card and is entered from the S-C Macro Assembler by typing "/" optionally followed by a line number. The neat thing is that all of the assembler regular editing commands COPY, REPlace, EDIt, FINd etc. are also availiable at the same time. It is almost a Macro Assembler Upgrade by itself.

It functions similar to the EDIt command in the macro assembler except that you can move forward and backward though the lines with cursor moves or move with paging keys a whole screen at a time. One intresting new edit command is control-C which can copy characters from the line above the cursor to the next tab stop of the current line. What a handy feature! How many times have you had to comment a routine that had no comments in it? With a control-W key a new left margin can be set at the comment area so every time you type the RETURN key you are all set to type the next comment line. This makes commenting a routine is as easy as eating apple pie!

The Screen Editor really cleans up a display because long lines are not wrapped arround on the display. Instead they are shown in a "window" on the display and the window can be moved up and down though a file and left or right to view long lines. As you type over the right side of the screen the "window" tracks over to always keep the cursor in the "window" of the screen.

It is very fast! Flipping though the pages of a source file to the routine you want to look at is just a few taps of a key. I hardly ever use the LISt command any more because the full screen editor is so easy to use: "/2400" for example will enter the editor and move to line 2400 at the top of the display.

For my own use I have made a Macro Assembler diskette that I boot on when I need the assembler. It loads up the Assembler and Screen Editor at the same time and applies several of the more useful patches published in the Apple Assembly Line for the Macro Assembler. An EXEC file is provided on the program diskette which can load the screen editor in to the langauge card from the assembler.

One of the most unusual features of the Screen Editor is that it comes with a SYSGEN program to help you create different customized versions of the screen editor for STB80 or VIDEX 80 column cards or the regular 40 column Apple II display. This keeps a user from performing a complicated series of BLOADs, POKEs and BSAVEs to modify the tab tables, screen width, margin settings and scroll values.

Some of the parameter settings are settable within the editor while you are editing like tab stops and the left margin. Others however are not accessable without re-running the Applesoft SYSGEN program and thats somewhat of a problem. I can't complain too much though because the source code comes with it and I can make it do anything extra that I want it to.

The Screen Editor can be used with the S-C Macro Cross Assemblers except for the 68000 version. Only the Z-80 cross assembler requires a slight adjustment to the small 20 byte patch for the "/" command. Provided with the program diskette is a tidy 9 page manual that describes the Screen Editor features and the patches to the assembler required.

We only have one machine here at S-C Software with an 80 column board but we use the Screen Editor mostly with the regular Apple II driver module.  Bob S-C is still holding out on using it but the rest of us counted how many times we typed the LISt command and decided to screen edit instead.  The Full Screen editor does for S-C Macro Assembler programmers what ES-CAPE does for Applesoft programmers.  They both make my job a lot easier!

The price for this little jem is $49.00 for both source and object code.

```
==================================================================
DOCUMENT :AAL-8304:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
Source code of S-C Assembler II, Version 4.0, on disk.............$95.00
    Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// ..........................................$100.00
    Preliminary version.  Call or write for details.

S-C Cross Reference Utility ......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00

S-C Word Processor................................................$50.00
  As is, with fully commented source code.  Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).................. $79.00
Source Code for FLASH! Runtime Package............................$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).......$49.00

The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00
Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
DISASM Dis-Assembler (Rak-Ware)...................................$30.00

Blank Diskettes (with hub rings)................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each...............10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                               100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Ashby Shift-Key Mod...............................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
     Only Revision level 7 or later Apples.

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
```

```
"Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
"Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
"Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
"Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
"Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
"What's Where in the Apple", Second Edition..........($24.95)  $23.00
"What's Where Guide" (updates first edition)..........($9.95)   $9.00
"6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
"6502 Subroutines", Leventhal........................($12.99)  $12.00
"MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
"MICRO on the Apple--3", includes diskette...........($24.95)  $23.00

 Add $1 per book for US postage.  Foreign orders add postage needed.

    *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
    ***                  (214) 324-2050                  ***
    *** We take Master Charge, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8304:Articles:ORG.Macro.txt
==============================================================================

An "ORG" Macro for Self-Aligning Code......Bob Sander-Cederlof

Roger Johnson (Minnesota) called a week or so ago with a plea for an easy way to make program segments align themselves automatically on page boundaries. He was writing a system to be burned into EPROM and run on another computer; it would be easier to debug in the target machine if subroutines and data blocks began on even page boundaries. There was ample room, so the wasted bytes between routines didn't bother him.

Of course, the .OR directive in the S-C Macro Assembler can easily change the origin whenever you wish, but it also changes the target address (.TA directive) or closes any open target file (.TF directive). Therefore a different approach is required.

Bill Morgan and Mike Laumer described how to do this in these pages a few months back, using the .BS directive to reserve enough bytes to reach the next page boundary. But with the help of a simple macro, we can not only make it easier to make self-aligning code: we can also make it generate error messages if the origin we try to set involves backing up over a longer-than-expected predecessor.

Here is the macro definition, and a few lines demonstrating how to call the macro:

```
1000        .MA ORG
1010        .DO *>]1
1020  !!! ERROR: ORG ]1 RANGE CROSSED !!!
1030        .ELSE
1040        .BS ]1-*
1050        .FIN
1060        .EM
1070 *-------------------------------
1080        .OR $800
1090 SAMPLE LDA $1234
1100        RTS
1110        >ORG $900
1120        STA $1234
1130        RTS
1140        >ORG $980
1150 DATA   .DA #1,#2,#3
```

Line 1110 calls the ORG macro with a parameter of "$900". This means that everywhere you find "]1" in the macro definition, the assembler will see "$900". The conditional (.DO) on line 1010 will read ".DO *>$900". Since * equals $804 at this point, it is not greater than $900. Therefore the condition is false, and the lines following line 1010 will be skipped up to line 1030 where there is an ".ELSE". The lines after 1030 through the ".FIN" on line 1050 will be assembled.

Line 1040 will be assembled as ".BS $900-*", which will bump the location up to $900.

Here's how the above example assembles:

```
               1000        .MA ORG
               1010        .DO *>]1
               1020  !!! ERROR: ORG ]1 RANGE CROSSED !!!
               1030        .ELSE
               1040        .BS ]1-*
               1050        .FIN
               1060        .EM
               1070 *-------------------------------
               1080        .OR $800
0800- AD 34 12 1090 SAMPLE LDA $1234
0803- 60       1100        RTS
0804-          1110        >ORG $900
               0000>        .DO *>$900
               0000>        .ELSE
0804-          0000>        .BS $900-*
               0000>        .FIN
0900- 8D 34 12 1120        STA $1234
0903- 60       1130        RTS
0904-          1140        >ORG $980
               0000>        .DO *>$980
               0000>        .ELSE
0904-          0000>        .BS $980-*
               0000>        .FIN
0980- 01 02 03 1150 DATA   .DA #1,#2,#3
```

If we had written line 1110 as ">ORG $800" the condition on line 1010 would be true, causing line 1020 to be assembled. Line 1020 is illegal syntax for the assembler, so it will be listed after an error message. The "]1" will be filled in to make the line list like this:

```
     *** BAD OPCODE ERROR
      1110>  !!! ERROR: ORG $800 RANGE CROSSED !!!
```

That will occur during pass one of the assembly, so no code will be generated. The error message will be the only output.

==============================================================================
DOCUMENT :AAL-8304:Articles:Patcher.txt
==============================================================================

PATCHER:  A General-Purpose Patch Installer.........Bill Morgan

My favorite new feature in Version 1.1 of the S-C Macro Assembler is the .PH directive.  When Bob first described the new directive to me, I didn't quite see how to use it.  Then he showed me a program like this one, and now I don't see how I did without it!

The directive .PH <expr> in an assembly causes the origin to be reset to <expr>, but the code continues to be stored in successive bytes of the same area as before.  The result is much like the following lines all rolled into one:

```
2000 LABEL
2010        .OR SOMEWHERE.ELSE
2020        .TA LABEL
```

The difference is that the above lines would close an open Target File, whereas .PH SOMEWHERE.ELSE continues to direct code into the same file.  The end of an offset block is marked with a .EP directive, that restores the origin to match the target address.

With this feature is so easy to assemble one program to create some patches and move them into place, all in one step.  Anyway, here's the general purpose PATCHER, with some dummy code to show it off.

Notice that the object code columns show the bytes to be all over pages 3, 10, 20, and 30.  The labels in the Symbol Table show the same thing.  But, if you look around in memory, all this is in page 3. Once you type $300G, the JMP instructions will be moved to their true destinations.

Bob's DOS Fast Load patches elsewhere in this issue are an ideal example of how to use PATCHER.  Here's all it takes:

1> Make the following changes to lines 1410-1430 of PATCHER:

```
     1410 P1.ORIGIN .EQ $BA69
     1420 P2.ORIGIN .EQ $BCDF
     1430 P3.ORIGIN .EQ $ACAF
```

2> Substitute Lines 1090-1160 of Fast Load for Line 1450 of PATCHER.

3> Substitute Lines 1210-1410 of Fast Load for Lines 1530-1550 of PATCHER.

4> Substitute Lines 1460-1590 of Fast Load for Lines 1630-1650 of PATCHER.

5> And substitute the following line for Lines 1740-1750 of PATCHER:

```
            .DA PATCH1
```

Now you have a BRUNnable program which will quickly install the Fast Load patches into DOS. And if you want to add other DOS patches to the same program, just tack them in between lines 1790 & 1800.

If you want to patch something running in a RAM card, like the Macro Assembler, you just need to add the following lines:

```
1082        LDA $C083
1084        LDA $C083

1315 .4     LDA $C080
1320        RTS
```

And that's how I expect to handle patches from now on. Hope you find it useful!

==============================================================================
DOCUMENT :AAL-8304:Articles:Prawm.Board.txt
==============================================================================

More about the PRAWM Board.................Bob Sander-Cederlof

Advanced Peripheral Enterprises has introduced their PRAWM board, and is advertising elsewhere in this issue of AAL.

The PRAWM board contains from 2K to 8K of EEPROM.  Data or programs can be written into the memory on the card, just as though it were RAM.  Yet the memory is non-volatile, as in ROM, PROM, or EPROM.  If you turn of your Apple, remove the card, ship it around the world...when you plug it back in the bytes will still be there!

EEPROM stands for "Electrically Eraseable PROM"; circuitry on the card allows you to individually write any bytes you wish, without erasing the rest of the memory.  You do not need a separate EPROM programmer and ultraviolet EPROM eraser.  There are no batteries either.  The card is priced about the same as an EPROM card, but you save a lot of money on accessories.  You will also save a lot of time, since you don't have to erase for 30-60 minutes, program chips for 5-20 minutes, and plug and unplug countless times.  (You can program the entire 8K on a fully loaded PRAWM board in less than 25 seconds!)

The PRAWM card contains from 1 to 4 EEPROM chips, providing from 2K to 8K bytes.  Each chip maps into the address space from $C800-$CFFF, and is accessed by switching in one chip at a time.  On-board firmware makes it easy to move blocks of data between any chip and RAM.

By installing a jumper strap, you can even have the program stored in the first 2K chip automatically start up when you turn on your Apple, before or instead of booting a floppy.  Just think of the possibilities:  set up special commands, execute security procedures, power fail recovery, "boot" a mini-DOS of your own creation from PRAWM, eliminate the need for disk drives in turn-key monitoring applications...!  Other strap options allow you to write-protect the board and to disable the $CFFF de-select function.

If you do a lot of development work involving EPROMs now, I think this card would be a big help.  See Advanced Peripheral Enterprises' ad for price and ordering information.

==================================================================
DOCUMENT :AAL-8304:Articles:V3N7.3.3E.txt
==================================================================

New Version of DOS -- Patchers Beware

When Apple released the //e they apparently also slipped in a slightly revised version of DOS, called DOS 3.3e (or 3.3c. Reports differ.) The following information about the changes is from Tom Weishaar's DOStalk column in the April issue of Softalk.

The boot routine now throws a couple of new soft switches ($C00C and $C00E) and stores $FF in location $4FB. These steps turn off the //e's 80-column mode during boot-up.

A routine at $B331 that calculates position in a random access file is now simplified.

Now for the biggie: Another APPEND fix! (attempted) According to Weishaar, they eliminated a bug that occurred maybe once in 10,000 tries by introducing a new bug that bites once every 256 calls. Tom says that the most reliable method is to use the old DOS 3.3 and POKE -18851,0 before each APPEND.

The most significant thing about the APPEND change is where they put the patch: at $BA69! That used to be empty space and a popular place to install patches. No more! As a matter of fact, Bob's Fast Load patch in this issue goes into that area, and therefore should not be used with DOS 3.3e.

This means that //e users should be especially careful about installing published patches into DOS 3.3e, and all of us should quit using $BA69-BA95 for patches that will be distributed.

 !pr2
**

** Not true! See "New Version of DOS -- Patchers Beware".

```
=============================================================================
DOCUMENT :AAL-8304:DOS3.3:Fast.Patch.txt
=============================================================================

8d∑:®:∑"PATCHING DOS FOR FAST LOADING...":∑20000:®:∑Í'∑
44,47721,173,230,181,208,36,173,194,181,240,31,173,203,181,72,173,204,
181,72,173,195,181,141,203,181,173,196,181,141,204,181,32,182,176,176,
3,76,223,188,76,111,179,76,150,172{ '∑
33,48351,238,228,181,208,3,238,229,181,238,196,181,238,204,181,206,194
,181,208,11,104,141,204,181,104,141,203,181,76,150,172,76,135,186• $'∑
2,44198,105,186™    .'∑ 0ª     N∑INSTALL FAST DOS LOAD AND BLOAD
PATCHESÃ  *N∑N:∑N»0∏∑‘  4N∑AÙ     >N∑I»1∑N:∑P:∑A,P:A»A¿1:∑
HN∑20010
```

```
==========================================================================
DOCUMENT :AAL-8304:DOS3.3:S.DATER.txt
==========================================================================

 1000  *---------------------------------
 1010  *    DATE PROCESSING MODULES
 1020  *       BY BROOKE BOERING
 1030  *---------------------------------
 1040         .OR $800
 1050  *---------------------------------
 1060  *         JUMP TABLE             *
 1070   JMP CONV1  MM/DD/YY -> STD FMT
 1080   JMP CONV2  STD FMT -> MM/DD/YY
 1090   JMP CONV3  STD FMT -> CENTURY
 1100  *           DAY & WEEKDAY CODE
 1110   JMP CONV4  KICK STD FMT DATE UP
 1120  *           (FROM 1 TO 225 DAYS)
 1130  *---------------------------------
 1140  *       MONITOR EQUATES
 1150
 1160  COUT   .EQ $FDED
 1170  PRBYTE .EQ $FDDA
 1180  *---------------------------------
 1190  *         LOCAL EQUATES
 1200
 1210  LOC0    .EQ $40  (A3L)
 1220  LOC1    .EQ $41  (A3H)
 1230  LOC2    .EQ $42  (A5L)
 1240  LOC3    .EQ $43  (A5H)
 1250  ACL     .EQ $50
 1260  ACH     .EQ $51
 1270  XTNDL   .EQ $52
 1280  XTNDH   .EQ $53
 1290  AUXL    .EQ $54
 1300  AUXH    .EQ $55
 1310  ANSLO   .EQ $50
 1320  PLIER   .EQ $51
 1330  CAND    .EQ $52
 1340  SAVER   .EQ $53
 1350  SLASH   .EQ $AF (/)
 1360  *---------------------------------
 1370  * - - - - LOCAL WORKING - - - - *
 1380  WKG   .HS 0000000000000000
 1390  BINYY .EQ WKG+0
 1400  BINMM .EQ WKG+1
 1410  BINDD .EQ WKG+2
 1420  CENTURY.DAY.HI  .EQ WKG+4
 1430  CENTURY.DAY.LO  .EQ WKG+5
 1440  *---------------------------------
 1450  *     USER ALTERABLE CONTROLS
 1460
 1470  * LOWEST ACCEPTABLE YEAR
 1480  *   DEFAULT= 75
```

```
1490  * HIGHEST ACCEPTABLE YEAR
1500  *   DEFAULT= 84
1510  * DAY-OF-WEEK SKIP
1520  *   DEFAULT= SUNDAY & SATURDAY
1530  *-------------------------------
1540  *      CONVERT EXTERNAL FORMAT
1550  *   MM/DD/YY TO STANDARD INTERNAL
1560  *   FORMAT; BITS YYYYYYYMMMMDDDDD
1570  *
1580  * ENTRY: RA= DATA ADDRESS-LO
1590  *        RY=   "       "  -HI
1600  *  EXIT: CC= EQUAL IF OK
1610  *          RA= YYYYYYYM BYTE
1620  *          RX= MMMMDDDD BYTE
1630  *        CC= NEQ IF ERROR
1635        .PG
1640  CONV1
1650   STA LOC2      SET INDIRECT ADDR
1660   STY LOC3        :
1670   LDA #0        INIT WKG
1680   STA BINMM
1690   STA BINDD
1700   STA BINYY
1710  *-- DO 'MM'
1720   JSR GET.DOUBLE
1730   BNE BADATE
1740   TAY           ZERO?
1750   BEQ BADATE
1760   CMP #13       TOO HI?
1770   BCS BADATE
1780   STA BINMM     ITS OK
1790   INC LOC2      KICK PAST '/'
1800  *-- DO 'DD'
1810   JSR GET.DOUBLE
1820   BNE BADATE
1830   TAY           ZERO?
1840   BEQ BADATE
1850   LDX BINMM     RX= INDEX TO LIST
1860   DEX
1870   CMP DAYS.COUNT,X
1880   BCC .3        G-A IF OK
1890   BEQ .3        G-A IF OK
1900   CMP #29       29TH (OF FEB)?
1910   BNE BADATE    NO, ERR!
1920   STY BINYY     YES, SET YY-FLAG
1930  * (ACCEPT TEMPORARILY)
1940  .3
1950   STY BINDD     ITS OK (PROBABLY)
1960   INC LOC2      KICK PAST '/'
1970  *-- DO 'YY'
1980   JSR GET.DOUBLE
1990   BNE BADATE
2000   CMP OLDEST.YEAR
2010   BCC BADATE
```

```
2020    LDX BINYY   RX= FEB 29TH FLAG
2030   STA BINYY    = 0YYYYYYY
2040    BEQ .6      G-A IF NOT FEB 29
2050   AND #$03     LEAP YEAR?
2060   BNE BADATE   ERR IF NOT LEAPYEAR
2070  *-- SET EXIT CONDITIONS
2080  .6
2090   LDA BINMM
2100   ASL
2110   ASL
2120   ASL
2130   ASL
2140   ASL
2150   ORA BINDD
2160   TAX          RX= MMMDDDDD
2170   LDA BINYY
2180   ROL          RA= YYYYYYYM
2190   LDY #0       EXIT OK
2200   RTS
2210
2220  BADATE
2230   LDY #$FF     DATE ERROR EXIT
2240   RTS
2245         .PG
2250  *********************************
2260  * S/R TO GET NEXT DOUBLE DIGIT
2270  *   (MAINLY USED FOR DATE INPUT)
2280  * ENTRY: LOC2/3= DATA ADDRESS
2290  GET.DOUBLE
2300   LDY #0
2310   LDA (LOC2),Y
2320   TAX          RX= TENS DIGIT
2330   INC LOC2
2340   LDA (LOC2),Y RA= UNITS DIGIT
2350   INC LOC2
2360   JSR ASC2BIN
2370  * (CC= ERROR STATUS; PASS BACK)
2380   RTS
2390  *********************************
2400  * S/R TO CONVERT 2 ASCII DIGITS
2410  *   TO SINGLE BINARY BYTE
2420  *
2430  * ENTRY: RA= UNITS ASCII DIGIT
2440  *        RX= TENS ASCII DIGIT
2450  *
2460  *  EXIT: CC= EQUAL IF OK
2470  *          RA= BINARY EQUIV
2480  *        CC= NEQ IF NON DIGIT
2490  ASC2BIN
2500   STA LOC1     (SAVE TEMP)
2510   TXA          RA= TENS
2520   CMP #0
2530   BCC NOTNUM
2540   CMP #10
```

```
2550   BCS NOTNUM
2560   AND #$0F
2570   BEQ .4
2580   TAX
2590   LDA #0
2600   CLC
2610  .3
2620   ADC #10
2630   DEX
2640   BNE .3
2650  .4
2660   STA LOC0
2670   LDA LOC1     RA= UNITS
2680   CMP #0
2690   BCC NOTNUM
2700   CMP #10
2710   BCS NOTNUM
2720   AND #$0F
2730   CLC
2740   ADC LOC0
2750   LDX #0       SET EXIT= OK
2760   RTS
2770
2780  NOTNUM
2790   LDX #$FF
2800   RTS
2805         .PG
2810  *--------------------------------
2820  *      CONVERT STANDARD INTERNAL
2830  *   DATE FORMAT, YYYYYYYMMMMDDDDD
2840  *   TO EXTERNAL FORMAT MM/DD/YY.
2850  *
2860  * ENTRY: RA= HI BYTE (YYYYYYYM)
2870  *        RX= LO BYTE (MMMDDDDD)
2880  *        CV/CH PRESUMED PRESET
2890  CONV2
2900  *-- EXPLODE TO BINYY,BINMM,BINDD
2910   JSR EXPLODE.STANDARD.FORMAT
2920   LDA BINMM
2930   JSR DATE.MM  PRINT MM
2940   LDA #SLASH   PRINT '/'
2950   JSR COUT
2960   LDA BINDD
2970   JSR DATE.DD  PRINT DD
2980   LDA #SLASH   PRINT '/'
2990   JSR COUT
3000   LDA BINYY
3010   JSR DATE.YY  PRINT YY
3020   RTS
3030  *********************************
3040  * S/R TO CONVERT YY BYTE TO DECI-
3050  *   MAL, THEN TO ASCII & DISPLAY.
3060  DATE.YY
3070   CMP #100     OVFLO PROTECT
```

```
3080   BCC .4         :
3090   LDA #99        :
3100  .4
3110   JMP DATE.DD  GOTO COMMON
3120  *********************************
3130  * S/R TO CONVERT MM BYTE TO DECI-
3140  *   MAL, THEN TO ASCII & DISPLAY.
3150  DATE.MM
3160   CMP #12      OVFLO PROTECT
3170   BCC .4         :
3180   LDA #12        :
3190  .4
3200   JMP DATE.DD  GOTO COMMON
3210  *********************************
3220  * S/R TO CONVERT DD BYTE TO DECI-
3230  *   MAL, THEN TO ASCII & DISPLAY.
3240  DATE.DD
3250   LDX #0      RX= 10'S CTR
3260  .2
3270   CMP #$A     < 10 ?
3280   BCC .3      YES, JUMP OUT
3290   SEC
3300   SBC #$A     MINUS 10
3310   INX         KICK 10'S CTR
3320   BNE .2      LOOP BACK
3330  *JMP^^^
3340  .3
3350   STA LOC0    SAVE TEMP
3360   TXA         GET 10'S CTR
3370   ASL         POSN HI
3380   ASL           :
3390   ASL           :
3400   ASL           :
3410   ORA LOC0    'OR' TOGETHER
3420   JMP PRBYTE  PRINT IT
3430  *RTS*
3435         .PG
3440  *--------------------------------
3450  *     CONVERT STANDARD FORMAT TO
3460  *     CENTURY DAY & WEEKDAY CODE
3470  *
3480  * ENTRY: RA= YYYYYYYM
3490  *        RX= MMMMDDDD
3500  *
3510  *  EXIT: RA= CENTURY DAY (HI)
3520  *        RX= CENTURY DAY (LO)
3530  *        RY= WEEKDAY CODE
3540  *          1= MONDAY
3550  *          2= TUESDAY
3560  *          3= WEDNESDAY
3570  *          4= THURSDAY
3580  *          5= FRIDAY
3590  *          6= SATURDAY
3600  *          7= SUNDAY
```

```
*          0= UNKNOWABLE
CONV3
*-- EXPLODE TO BINYY,BINMM,BINDD
 JSR EXPLODE.STANDARD.FORMAT
*-- CALCULATE DAYS OF PRIOR YEARS
 LDY BINYY        STORE 256 DAYS
 DEY                : FOR EACH
 STY CENTURY.DAY.HI : PRIOR YEAR
 TYA              STORE 1 DAY
 LSR                : FOR EACH
 LSR                : PRIOR
 STA CENTURY.DAY.LO : LEAP YEAR
 LDA #109         STORE 109 DAYS
 JSR MULTIPLY.8X8   : FOR EACH
 CLC          A     : PRIOR
 ADC CENTURY.DAY.LO : YEAR
 STA CENTURY.DAY.LO :
 TYA                :
 ADC CENTURY.DAY.HI :
 STA CENTURY.DAY.HI :

*-- CALCULATE DAYS OF THIS YEAR
  LDY BINDD   RY= DD
 TYA          (IN CASE WAS JAN)
 LDX BINMM    RX= MM
 DEX          RX= MM-1
 BEQ .7       G-A IF WAS JAN
 CPX #1
 BEQ .3       G-A IF WAS FEB
 LDA BINYY    (WAS MAR - DEC)
 AND #$03     LEAP YEAR?
 BNE .3       NO, G-A
 INY          YES, KICK DAY CTR
.3
 TYA          RA= DD (OR DD+1)
.4
 CLC          ADD A MONTH'S DAYS
 ADC DAYS.COUNT-1,X :
 BCC .5       G-A IF > 255 DAYS
 INC CENTURY.DAY.HI
.5
 DEX          DECR CTR
 BNE .4       LOOP TIL DONE
       .PG
.7
*-- ADD THIS YEAR'S DAYS
*     TO PRIOR YEARS' DAYS
*RA= DAYS THIS YEAR
 CLC
 ADC CENTURY.DAY.LO  :
 STA CENTURY.DAY.LO  :
 BCC .8              :
 INC CENTURY.DAY.HI  :
.8
```

```
4140  *-- CALCULATE WEEKDAY CODE
4150   TAX          RX= CENTURY.DAY.LO
4160   LDA CENTURY.DAY.HI
4170   JSR GET.WEEKDAY
4180  *             RY= WEEKDAY CODE
4190   RTS
4200  *********************************
4210  *    CALCULATE WEEKDAY CODE FROM
4220  *    CENTURY DATE
4230  *
4240  * ENTRY: RA= CENTURY DATE-HI
4250  *        RX= CENTURY DATE-LO
4260  *
4270  *  EXIT: RA/RX= AS ON ENTRY
4280  *        RY= WEEKDAY CODE
4290  *          1= MONDAY
4300  *          2= TUESDAY
4310  *          3= WEDNESDAY
4320  *          4= THURSDAY
4330  *          5= FRIDAY
4340  *          6= SATURDAY
4350  *          7= SUNDAY
4360  *          0= UNKNOWABLE
4370  GET.WEEKDAY
4380   STA ACH
4390   STX ACL
4400   PHA          SAVE RA
4410   TXA          SAVE RX
4420   PHA            :
4430   LDA #0
4440   STA XTNDH    SET DIV'D (HIHI)
4450   STA XTNDL    SET DIV'D (LOLO)
4460   STA AUXH     SET DIVISOR(LO)
4470   LDA #7       SET DIVISOR(HI)
4480   STA AUXL       :
4490   LDY #8       SET FOR 8BIT DIVSR
4500   JSR DIVIDE.32X16
4510   LDA XTNDL
4520   CLC          REMAINDER + WEEKDAY
4530   ADC #0       : OF 12/31/1900
4540    TAY         (PRESET)
4550   SEC
4560   SBC #7
4570   BCC .4       G-A IF RY OK
4580    TAY         (RESET)
4590  .4
4600   INY          ADJ: ANS+1 = CODE
4610   PLA          RESTORE RA/RX
4620   TXA            :
4630   PLA            :
4640   RTS
4645        .PG
4650  *--------------------------------
4660  *     ADD FROM 1 TO 225 DAYS TO
```

```
*     A GIVEN STD FORMAT DATE
*
* ENTRY: RA= YYYYYYYM
*        RX= MMMMDDDD
*        RY= # DAYS TO ADD
*  EXIT: RA/RX UPDATED
CONV4
*-- SAVE RY TO STACK
 STA LOC0
 TYA
 PHA
 LDA LOC0
*-- EXPLODE TO BINYY,BINMM,BINDD
 JSR EXPLODE.STANDARD.FORMAT
*-- INIT FOR LOOP
 PLA          = # DAYS TO KICK
 CLC
 ADC BINDD    RA= WKG CTR
 LDX BINMM    RX= WKG MM
.2
* IN THIS LOOP:
*   RY= UTILITY REGISTER
*   RX= WKG MM TO BE INCREMENTED
*   RA= WKG CTR TO BE DECREMENTED
* LOC3= WKG DAY COUNT FOR THE
*       CURRENT MM (IN RX)
  LDY DAYS.COUNT-1,X
  STY LOC3    = MM'S DAY COUNT
 CPX #2       IS MM FEB?
 BNE .4       NO, G-A
*-- DO FEB
 PHA          SAVE WKG CTR
 LDA BINYY
 AND #$03     LEAP YEAR?
 BNE .3       NO, G-A
 LDA #29      RESET DAY COUNT
 STA LOC3     :
.3
 PLA          RESTORE WKG CTR
.4
 CMP LOC3
 BCC .7       G-A IF DONE
 BEQ .7       : (ALSO DONE)
 SEC          WKG CTR MINUS
 SBC LOC3     : WKG DAY COUNT
 INX          MM+1
 CPX #13      OVFLO?
 BCC .2       NO, LOOP BACK
 LDX #1       YES, SET MM= JAN
 INC BINYY    : AND SET YY+1
 JMP .2       : AND LOOP BACK
.7
 STA BINDD
 STX BINMM
```

```
5210  JSR IMPLODE.STANDARD.FORMAT
5220  RTS
5225        .PG
5230 *********************************
5240 * S/R TO EXPLODE STD FORMAT TO
5250 *   BINYY, BINMM & BINDD
5260 * ENTRY: RA= YYYYYYYM
5270 *        RX= MMMMDDDD
5280 *  EXIT: BINYY,BINMM,BINDD SET
5290 EXPLODE.STANDARD.FORMAT
5300  LSR          RA= 0YYYYYYY CC=M
5310  STA BINYY
5320  TXA          RA= MMMDDDDD
5330  PHA         SAVE MMMDDDDD
5340  ROR          RA= MMMMDDDD
5350  LSR              0MMMMDDD
5360  LSR              00MMMMDD
5370  LSR              000MMMMD
5380  LSR              0000MMMM
5390  STA BINMM
5400  PLA         PULL MMMDDDDD
5410  AND #$1F     RA= 000DDDDD
5420  STA BINDD
5430  RTS
5440 *********************************
5450 * S/R TO IMPLODE BINYY, BINMM &
5460 *   BINDD TO STD FORMAT
5470 * ENTRY: BINYY,BINMM,BINDD PRESET
5480 *  EXIT: RA= YYYYYYYM
5490 *        RX= MMMMDDDD
5500 IMPLODE.STANDARD.FORMAT
5510  LDA BINMM    RA= 0000MMMM
5520  ASL              000MMMM0
5530  ASL              00MMMM00
5540  ASL              0MMMM000
5550  ASL              MMMM0000
5560  ASL              MMM00000 (CC=M)
5570  ORA BINDD        MMMDDDDD
5580  TAX          RX= MMMDDDDD (CC=M)
5590  LDA BINYY    RA= 0YYYYYYY (CC=M)
5600  ROL          RA= YYYYYYYM
5610  RTS
5620 *********************************
5630
5640 OLDEST.YEAR
5650  .DA #75
5660 HIGHEST.YEAR
5670  .DA #84
5680 DAYS.COUNT
5690  .DA #31 (JAN)
5700  .DA #28 (FEB)
5710  .DA #31 (MAR)
5720  .DA #30 (APR)
5730  .DA #31 (MAY)
```

```
5740   .DA #30 (JUN)
5750   .DA #31 (JUL)
5760   .DA #31 (AUG)
5770   .DA #30 (SEP)
5780   .DA #31 (OCT)
5790   .DA #30 (NOV)
5800   .DA #31 (DEC)
5805         .PG
5810  *********************************
5820  *
5830  *       8 X 8 MULTIPLY
5840  *
5850  *  ENTRY: RY= MULTIPLCAND
5860  *         RA= MULTIPLIER
5870  *
5880  *   EXIT: RY= ANSWER-HI
5890  *         RA= ANSWER-LO
5900  *
5910  *  TIMING: 212 US - MAX
5920  *          180 US - MIN
5930  *          192 US - AVER
5940  * NOTE: KEEP CLOSE TO SGN8X8
5950  MULTIPLY.8X8
5960   STA PLIER   SAVE (MULTI)PLIER
5970   STY CAND    SAVE (MULTIPL)CAND
5980   LDA #0      RA= ANSWER-HI
5990   LDY #8      SET 8-BIT CTR
6000  MUL1
6010   LSR PLIER   TEST NEXT BIT
6020   BCC MUL2    IF OFF, GO ROUND
6030   CLC
6040   ADC CAND    IF ON, ADD
6050  MUL2
6060   ROR         SHIFT ANSWER 1 BIT
6070   ROR ANSLO   :
6080   DEY         DECR POSITION CTR
6090   BNE MUL1    LOOP TIL DONE 8 BITS
6100   TAY         RY= ANSWER-HI
6110   LDA ANSLO   RA= ANSWER-LO
6120   RTS
6130  *********************************
6140  *
6150  *        32 X 16 DIVIDE
6160  *
6170  *  PRE-ENTRY:
6180  *    DIVIDEND IN:
6190  *      XTNDH,XTNDL,ACH,ACL
6200  *    DIVISOR --> AUXL,AUXH
6210  *
6220  *   EXIT: QUOTIENT -> ACL,ACH
6230  *    REMAINDER -> XTNDL,XTNDH
6240  DIVIDE.32X16
6250   LDY #$10      INDEX FOR 16 BITS
6260  .2
```

```
6270   ASL ACL
6280   ROL ACH
6290   ROL XTNDL     XTND/AUX
6300   ROL XTNDH     : -> ACCUM
6310   SEC
6320   LDA XTNDL
6330   SBC AUXL      MOD TO XTND.
6340   TAX
6350   LDA XTNDH
6360   SBC AUXH
6370   BCC .3
6380   STX XTNDL
6390   STA XTNDH
6400   INC ACL
6410  .3
6420   DEY
6430   BNE .2
6440   RTS
6450  *********************************
```

```
==============================================================================
DOCUMENT :AAL-8304:DOS3.3:S.FAST.LOAD.txt
==============================================================================

 1000  *--------------------------------
 1010  *   S.FAST LOAD.1
 1020  *
 1030  *      FAST "LOAD" AND "BLOAD"
 1040  *
 1050  *      INSTALLED IN UNUSED AREAS IN DOS 3.3:
 1060  *          $BA69-$BA95   (45 BYTES FREE)
 1070  *          $BCDF-$BCFF   (33 BYTES FREE)
 1080  *--------------------------------
 1090  READ.RANGE          .EQ $AC96
 1100  READ.NEXT.SECTOR    .EQ $B0B6
 1110  END.OF.DATA.ERROR   .EQ $B36F
 1120  RANGE.LENGTH        .EQ $B5C1,C2
 1130  RANGE.ADDRESS       .EQ $B5C3,C4
 1140  BUFFER.ADDRESS      .EQ $B5CB,CC
 1150  SECTOR.COUNT        .EQ $B5E4,E5
 1160  BYTE.OFFSET         .EQ $B5E6
 1170  *--------------------------------
 1180         .OR $BA69
 1190         .TF B.PATCH1
 1200
 1210  PATCH1 LDA BYTE.OFFSET       LAST BYTE OF
 1220         BNE GO.READ.RANGE     A SECTOR?
 1230         LDA RANGE.LENGTH+1    WHOLE SECTOR LEFT?
 1240         BEQ GO.READ.RANGE     NO.
 1250         LDA BUFFER.ADDRESS    SAVE BUFFER ADDRESS
 1260         PHA
 1270         LDA BUFFER.ADDRESS+1
 1280         PHA
 1290         LDA RANGE.ADDRESS     READ DIRECTLY
 1300         STA BUFFER.ADDRESS    INTO RANGE
 1310         LDA RANGE.ADDRESS+1
 1320         STA BUFFER.ADDRESS+1
 1330
 1340  READ.LOOP
 1350         JSR READ.NEXT.SECTOR
 1360         BCS .1
 1370         JMP PATCH2
 1380  .1     JMP END.OF.DATA.ERROR
 1390
 1400  GO.READ.RANGE
 1410         JMP READ.RANGE
 1420  *--------------------------------
 1430         .OR $BCDF
 1440         .TF B.PATCH2
 1450
 1460  PATCH2 INC SECTOR.COUNT
 1470         BNE .1
 1480         INC SECTOR.COUNT+1
```

```
1490  .1     INC RANGE.ADDRESS+1   NEXT PAGE
1500         INC BUFFER.ADDRESS+1
1510         DEC RANGE.LENGTH+1
1520         BNE .2
1530         PLA                 RESTORE BUFFER
1540         STA BUFFER.ADDRESS+1
1550         PLA
1560         STA BUFFER.ADDRESS
1570         JMP READ.RANGE    ONE BYTE AT A TIME
1580
1590  .2     JMP READ.LOOP
1600  *--------------------------------
1610         .LIF
```

```
==============================================================================
DOCUMENT :AAL-8304:DOS3.3:S.ORG.MACRO.txt
==============================================================================

 1000        .MA ORG
 1010        .DO *>]1
 1020   !!! ERROR: ORG ]1 RANGE CROSSED !!!
 1030        .ELSE
 1040        .BS ]1-*
 1050        .FIN
 1060        .EM
 1070  *--------------------------------
 1080        .OR $800
 1090  SAMPLE LDA $1234
 1100        RTS
 1110        >ORG $800
 1120        STA $1234
 1130        RTS
 1140        >ORG $980
 1150  DATA   .DA #1,#2,#3
 1160        .LIF
```

```
==============================================================================
DOCUMENT :AAL-8304:DOS3.3:S.PATCHER.txt
==============================================================================

 1000  *SAVE S.PATCHER
 1010  *---------------------------------
 1020  PNTR   .EQ $00,01
 1030  PATCH  .EQ $02,03
 1040  *---------------------------------
 1050         .OR $300
 1060         .TF B.PATCHER
 1070  *---------------------------------
 1080  PATCHER
 1090         LDA #PATCHES-1
 1100         STA PNTR
 1110         LDA /PATCHES-1
 1120         STA PNTR+1
 1130         LDY #0
 1140
 1150  .1     JSR GET.BYTE LENGTH OF NEXT PATCH
 1160         BEQ .4       FINISHED
 1170         TAX          SAVE LENGTH IN X
 1180         JSR GET.BYTE ADDRESS OF PATCH
 1190         STA PATCH
 1200         JSR GET.BYTE
 1210         STA PATCH+1
 1220
 1230  .2     JSR GET.BYTE
 1240         STA (PATCH),Y
 1250         INC PATCH
 1260         BNE .3
 1270         INC PATCH+1
 1280  .3     DEX
 1290         BNE .2
 1300         BEQ .1    ...ALWAYS
 1310
 1320  .4     RTS
 1330  *---------------------------------
 1340  GET.BYTE
 1350         INC PNTR
 1360         BNE .1
 1370         INC PNTR+1
 1380  .1     LDA (PNTR),Y
 1390         RTS
 1400  *---------------------------------
 1410  P1.ORIGIN .EQ $1000
 1420  P2.ORIGIN .EQ $2000
 1430  P3.ORIGIN .EQ $3000
 1440
 1450  *  OTHER .EQUATES HERE
 1460
 1470  *---------------------------------
 1480  PATCHES
```

```
1490
1500        .DA #P1.LENGTH,P1.ORIGIN
1510        .PH P1.ORIGIN
1520
1530 PATCH1
1540 *      PATCH1 CODE HERE
1550        JMP PATCH2
1560
1570 P1.LENGTH .EQ *-PATCH1
1580        .EP
1590 *--------------------------------
1600        .DA #P2.LENGTH,P2.ORIGIN
1610        .PH P2.ORIGIN
1620
1630 PATCH2
1640 *      PATCH2 CODE HERE
1650        JMP PATCH3
1660
1670 P2.LENGTH .EQ *-PATCH2
1680        .EP
1690 *--------------------------------
1700        .DA #P3.LENGTH,P3.ORIGIN
1710        .PH P3.ORIGIN
1720
1730 PATCH3
1740 *      PATCH3 CODE HERE
1750        JMP PATCH1
1760
1770 P3.LENGTH .EQ *-PATCH3
1780        .EP
1790 *--------------------------------
1800        .DA #0      END OF PATCHES
1810 *--------------------------------
1820        .DO *>$3D0
1830  !!! PATCHER IS TOO BIG !!!
1840        .FIN
```

```
==============================================================================
DOCUMENT :AAL-8305:Articles:AAL.CHART.txt
==============================================================================
```

Apple Assembly Line
-------------------

```
Issue......1......3......5......7......9.....11...
       +------+------+------+------+------+------+------+
Oct 80 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Nov 80 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Dec 80 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Jan 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Feb 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Mar 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Apr 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
May 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Jun 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Jul 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Aug 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Sep 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Oct 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Nov 81 |      |      |      |      |      |      |      |
       |      |      |      |      |      |      |      |
       +------+------+------+------+------+------+------+
Dec 81 |      |      |      |      |      |      |      |plus
       |      |      |      |      |      |      |      |middle sheet
```

```
       +------+------+------+------+------+------+
Jan 82 |      |      |      |      |      |      |
       |      |      |      |      |      |      |
       +------+------+------+------+------+------+
Issue......1......3......5......7......9.....11...
```

```
==================================================================
DOCUMENT :AAL-8305:Articles:APPLE.CHIPS.txt
==================================================================

           Board                 Chip
Chip      Location(s)           Description
----      -----------           --------

555       A13 B3                Timer
558       H13                   3 Timers
741       K13                   Op Amp

2316B     A5  (Rev 7,RFI)       ROM (character generator)
2513      A5  (Rev 0,1)         ROM (character generator)
4116      C3-10 D3-10 E3-10     RAM
6502      H6-9                  Microprocessor
9316B     F3-11 (6 chips)       ROM (monitor and language)

74LS00    A2                    4 2-input NAND
74LS02    A12 A14 B13 B14       4 2-input NOR
74LS04    C11                   6 Inverters
74LS08    B11 H1                4 2-input AND
74LS11    B12                   3 3-input AND
74LS20    D2                    2 4-input NAND
74LS32    C14                   4 2-input OR
74LS51    C13                   AND3-NOR2, AND2-NOR2
74LS74    A11 B10 J13           2 Flip-Flops
74LS86    B2                    4 2-input XOR
74LS138   F12 F13 H2  H12       3-by-8 Decoder
74LS139   E2  F2                2 3-by-4 Decoders
74LS151   A9                    1-of-8 Selector
74LS153   C1  E11 E12 E13       2 1-of-4 selectors
74LS161   D11-14                Counter
74  166   A3                    8-bit Shift Register
74LS174   B5  B8                6 Flip-Flops
74LS175   B1                    4 Flip-Flops
74LS194   A10 B4  B9            4-bit Shift Register
74LS195   C2                    4-bit Shift Register
74LS251   H14                   1-of-8 Selector
74LS257   A8  B7  C12 J1        4 1-of-2 Selectors
74LS259   F14                   8-bit Addressable Latch
74LS283   E14                   4-bit Full Adder
74LS367   H3  H4  H5 (on some models)  6 Bus Drivers

8T97      H3  H4  H5 (on most models)  6 Bus Drivers
8T28      H10 H11    (on rev 0,1,7)    4 Bus Buffers
8304      H10        (on ref RFI)      8 Bus Buffers
!E

Drawing
XC-x       Description          RevisionY
2      Clock Generator
3      Video Adress Generator
```

```
4       Memory address--part 1    0,1
5       Memory address--part 1   7, RFI
6       Memory address--part 2    All
7       RAM                       All
8       Microprocessor           0,1,7
9       Microprocessor            RFI
10      ROM                       All
11      Peripheral I/O            All
12      On-board I/O--part 1      All
13      On-board I/O--part 2      All
14      Video Generator--part 1    0
15      Video Generator--part 1    1
16      Video Generator--part 1  7,RFI
17      Video Generator--part 2    0
18      Video Generator--part 2    1
19      Video Generator--part 2    7
20      Video Generator--part 2   RFI
21      Single-piece keyboard
22      Two-piece keyboard
23      Power Supply

Board
XLocation     Drawing C-x              Y   XLocation     Drawing C-
x           Y
A2 . . . . . . 2, 4, 5
A3 . . . . . . 14-16
A5 . . . . . . 14-16
A7 . . . . . . 13
A8-A10 . . . . 14-16
A11. . . . . . 15, 16
A12. . . . . . 13, 15, 16, 18, 19
A13. . . . . . 13
A14. . . . . . 19, 20

B2 . . . . . . 2, 14-16
B3-B4. . . . . 14-16
B5 . . . . . . 7, 14-16
B6-B7. . . . . 13
B8 . . . . . . 7, 14-16
B9 . . . . . . 14-16
B10. . . . . . 13-16
B11. . . . . . 8, 9, 14-20
B12-B13. . . . 2, 14-20

C1 . . . . . . 2, 4, 5
C2 . . . . . . 2
C3-C10 . . . . 7
C11. . . . . . 3-6, 8, 9, 13, 17-20
C12. . . . . . 4, 5, 6
C13. . . . . . 17-20
C14. . . . . . 4, 7-9, 17-20

D1 . . . . . . 4
D2 . . . . . . 2, 4, 5
```

```
D3-D10 . . . 7
D11-D14. . . 3

E1-E2. . . . 4
E3-E10 . . . 7
E11-E14. . . 6

F1 . . . . . 4
F2 . . . . . 4, 5
F3-F12 . . . 10
F13-F14. . . 12

H1 . . . . . 4, 5, 10
H3 . . . . . 8, 9, 11
H4-H10 . . . 8, 9
H11. . . . . 8
H12. . . . . 11
H14. . . . . 12

J1 . . . . . 4, 5
J13-J14. . . 12

K13. . . . . 1
```

```
=============================================================================
DOCUMENT :AAL-8305:Articles:Apple.Chips.Txt.txt
=============================================================================
```

Apple Chips........................................Bob and Bill

You may recall that when Bill reviewed Apple ][ Circuit Description last month, he bemoaned the lack of a "Cross Reference", by board location, of all the Apple's ICs. Well Bob has worked out a couple of tables to fill that gap, and we'll be including those tables in future shipments of the book.

In the meantime, here's another sort of table, showing the locations and descriptions of all the chips in your Apple. This one is organized by chip number.

| Chip | Board Location(s) | Chip Description |
|---|---|---|
| 555 | A13 B3 | Timer |
| 558 | H13 | 3 Timers |
| 741 | K13 | Op Amp |
| | | |
| 2316B | A5 (Rev 7,RFI) | ROM (character generator) |
| 2513 | A5 (Rev 0,1) | ROM (character generator) |
| 4116 | C3-10 D3-10 E3-10 | RAM |
| 6502 | H6-9 | Microprocessor |
| 9316B | F3-11 (6 chips) | ROM (monitor and language) |
| | | |
| 74LS00 | A2 | 4 2-input NAND |
| 74LS02 | A12 A14 B13 B14 | 4 2-input NOR |
| 74LS04 | C11 | 6 Inverters |
| 74LS08 | B11 H1 | 4 2-input AND |
| 74LS11 | B12 | 3 3-input AND |
| 74LS20 | D2 | 2 4-input NAND |
| 74LS32 | C14 | 4 2-input OR |
| 74LS51 | C13 | AND3-NOR2, AND2-NOR2 |
| 74LS74 | A11 B10 J13 | 2 Flip-Flops |
| 74LS86 | B2 | 4 2-input XOR |
| 74LS138 | F12 F13 H2 H12 | 3-by-8 Decoder |
| 74LS139 | E2 F2 | 2 3-by-4 Decoders |
| 74LS151 | A9 | 1-of-8 Selector |
| 74LS153 | C1 E11 E12 E13 | 2 1-of-4 selectors |
| 74LS161 | D11-14 | Counter |
| 74 166 | A3 | 8-bit Shift Register |
| 74LS174 | B5 B8 | 6 Flip-Flops |
| 74LS175 | B1 | 4 Flip-Flops |
| 74LS194 | A10 B4 B9 | 4-bit Shift Register |
| 74LS195 | C2 | 4-bit Shift Register |
| 74LS251 | H14 | 1-of-8 Selector |
| 74LS257 | A8 B7 C12 J1 | 4 1-of-2 Selectors |
| 74LS259 | F14 | 8-bit Addressable Latch |

```
74LS283   E14                  4-bit Full Adder
74LS367   H3  H4  H5 (on some models)  6 Bus Drivers

8T97      H3  H4  H5 (on most models)  6 Bus Drivers
8T28      H10 H11    (on rev 0,1,7)    4 Bus Buffers
8304      H10        (on ref RFI)      8 Bus Buffers
```

==============================================================================
DOCUMENT :AAL-8305:Articles:Cross.Ad.txt
==============================================================================

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

| | | | |
|---|---|---|---|
| Motorola: | 6800/6801/6802 | now | $32.50 |
| | 6805 | now | $32.50 |
| | 6809 | now | $32.50 |
| | 68000 | now | $50.00 |
| Intel: | 8048 | now | $32.50 |
| | 8051 | now | $32.50 |
| | 8085 | soon | $32.50 |
| Zilog: | Z-80 | now | $32.50 |
| RCA: | 1802/1805 | now | $32.50 |
| Rockwell: | 65C02 | now | $20.00 |
| DEC: | PDP-11/LSI-11 | now | $50.00 |

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support:  competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation   (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

==============================================================================
DOCUMENT :AAL-8305:Articles:Display.CharSet.txt
==============================================================================

Displaying Character Generator EPROMs......Bob Sander-Cederlof

We make our own Character Generator EPROMs for Revision 7 or later Apple II Plusses. I use the Mountain Hardware EPROM Burner to burn the data into 2716 EPROMs. We have several different character sets, and it can be a lot of trouble to check the results.

After designing a character set, and formatting all the bits into the 2048 bytes of EPROM space, and burning it in, we still have to take an Apple apart and plug the chip in to see if all the characters look right.

I decided to write a program which would map the EPROM data onto the hi-res screen, allowing me to test without wasting time burning/erasing EPROMs and dismantling/re-assembling my Apple.

Even if you don't have the same requirements, you can learn a lot about indexing techniques and address shuffling from studying the following program.

Starting at the top.... I set up three page-zero variables in lines 1040-1060. The S-C Macro Assembler is a great environment for making short programs like this one, because I can cycle through edit-assemble-test until it works just right without ever leaving the assembler. S-C Macro allows me to use zero-page locations $00-$1F without fear of inteference ($00-$1E in the Apple //e).

Lines 1080 and 1090 define two buffers where I BLOAD two different EPROM images. I put one at $6800-6FFF, the other at $7000-77FF. There is room on the screen to display one character set in a 16x16 matrix on the left side, and the other on the right side.

For grins, I decided to use the subroutine in Applesoft ROM at $F3E2 to turn on hi-res mode. This is the code executed for the HGR statement, so I called it AS.HGR at line 1110. HGR sets all the soft-switches to hi-res page 1, and clears the screen.

Lines 1160-1180 call the HGR subroutine. Since I was using S-C Macro in the RAM card, and since the Applesoft ROMs are not switched on when a program is executing in the RAM card, I had a problem. The first time I tried to run DISPLAY, I left out lines 1160 and 1180. The result was a total disaster. Line 1170 did a JSR $F3E2 into the RAM card! I had to RESET and reboot the computer to get control again. Look out for these kinds of problems whenever you are trying to use code in both places at once.

Lines 1190-1280 set up the starting addresses to display the first character set on the left half of the screen. Lines 1290-1380 do the same job to show the second set on the right half-screen.

The top line of hi-res page 1 starts at $2000, and goes to $2027. The middle of the line starts at $2014. The starting addresses of subsequent lines can be computed from these two base addresses, although it is a little tricky. More on this later.

The hi-res screen shows the least significant seven bits from each byte. There are forty bytes in each line, making a total of 280 dots across. The dots in each byte are in reverse order: the least significant bit is the leftmost dot. On the other hand, the EPROM image is in normal order. The subroutine DISPLAY.ONE.SET takes care of all the addressing, and REVERSE.BITS handles the reversals.

Lines 1400-1410 pause until I hit any key on the keyboard. During this pause I can examine the screen as long as I wish. When I type any key, the keyboard strobe will be set and $C000 will go negative. Line 1420 will then clear the keyboard strobe, and the RTS at line 1430 returns to the S-C Macro Assembler.

This brings us to a closer examination of the subroutine to actually display a character set, in lines 1440-1770. We will be displaying 16 rows of characters, with 16 characters in each row. It is therefore natural to simplify the problem by writing another subroutine to display one row of characters, and call it sixteen times.

Lines 1480 and 1490 start a loop much like Applesoft's FOR I = 1 TO 16...except in assembly language it is easier to go from 16 to 1. The equivalent to NEXT I is at lines 1750 and 1760, where CNT16 is decremented. In between we have the body of the loop.

Line 1500 calls DISPLAY.ONE.ROW, a subroutine that only gets called from this one line. I made it into a separate subroutine so I could put off writing it until later, and concentrate on one loop at a time. DISPLAY.ONE.ROW expects the addresses at SCREEN.ADR and EPROM.ADR to be already set up for the first byte to be displayed in the current row. After it returns, those addresses will have been modified.

Lines 1510-1580 add 15*8, or 120, to the address in EPROM.ADR. DISPLAY.ONE.ROW already added 8, so the total augment is 128. This moves us up to the beginning of the next set of sixteen characters.

Lines 1590-1740 assume that DISPLAY.ONE.ROW already added $2000 to the address in SCREEN.ADR, and subtracts that value back out. At the same time, we add back in $80, to move to the next group of eight screen lines for the next row of characters. This is sufficient for the first eight rows of characters, but in moving to the ninth row there is a discontinuity which requires adding $28 and subtracting $400 to get the right address. The fact that the ninth row has arrived is apparent by the fact that the high byte of the address goes above $23 (lines 1670 and 1680).

Here is a table of the starting addresses for each of the 24 character rows (we only use the first 16):

| Row | Address | Row | Address | Row | Address |
|---|---|---|---|---|---|
| 1 | $2000 | 9 | $2028 | 17 | $2050 |
| 2 | $2080 | 10 | $20A8 | 18 | $20D0 |
| 3 | $2100 | 11 | $2128 | 19 | $2150 |
| 4 | $2180 | 12 | $21A8 | 20 | $21D0 |
| 5 | $2200 | 13 | $2228 | 21 | $2250 |
| 6 | $2280 | 14 | $22A8 | 22 | $22D0 |
| 7 | $2300 | 15 | $2328 | 23 | $2350 |
| 8 | $2380 | 16 | $23A8 | 24 | $23D0 |

The starting addresses for the right half-screen can be obtained by just adding $14 to all of the above addresses. What we do is START at $2014, and all the rest are computed automatically.

Now we can talk about what goes on inside one row of characters. Lines 1810-2000 do the job of moving bytes from the EPROM image to the eight screen lines which form the row of characters. Lines 1820-1830 start a loop to count out eight repetitions, and lines 1980-1990 perform the NEXT on this loop.

On each pass through the loop the subroutine GET.PUT is called sixteen times to move a byte for each character to the screen image. GET.PUT is another subroutine only called from one place, but made into a subroutine for ease of understanding. The inner loop of 0 through 15 is controlled by the X-register. Line 1850 sets X=0, and lines 1910-1930 increment, test, and branch ("NEXT X" sequence). The X-register also indexes the STA instruction inside GET.PUT, so that the screen byte for each character is stored into the right place on the screen line. The Y-register is used as an index into the EPROM data by GET.PUT, and parallels the X-register but with an increment of 8 rather than 1. Lines 1870-1900 bump the Y-register by 8 each time through the inner loop.

GET.PUT (lines 2230-2340) does the very simple job of moving one byte from one place in memory to another. Or is it so simple.... Notice that the addresses inside the LDA and STA instructions are filled in when the program runs. This is called self-modifying code, and I normally avoid such code at all costs. It can lead to all sorts of devastating things. Nevertheless, there are exceptions to most rules, and a time for nearly everything. This is one of those, I think. Isolating the offensive code into its own little subroutine appeases my conscience somewhat.

In between LDA and STA I call REVERSE.BITS, yet another simple subroutine which could be written in-line. I prefer making it separate for nicer modularity. The comments show what is going on, bit-by-bit. If you were working from character generator data written for the DOS TOOL KIT or HIGHER TEXT, the bits would already be in the right order. It is just because I am using data for the character generator EPROM that we need to reverse the bits.

Here is a printout done with my NEC PC-8023 and a Grappler+ interface card. The two character sets shown are the ones we sell. The one on the left uses regular lower case characters, with descenders. All the lower case characters are raised up one screen line to leave room for the descenders. The set on the right uses small caps for the lower case, and is the one we use in all the Apples here. The first four rows are the characters used in INVERSE mode, and the next four rows are for FLASH mode. (Doesn't flash too well on paper!)

<character sets and program follow>

==============================================================================
DOCUMENT :AAL-8305:Articles:FADD.txt
==============================================================================

FADD -- Find ADDress references..................Brooke Boering

Recently I have been messing around with modifications to DOS.  Since I didn't have the complete source code for it, I simply used the explanations in "Beneath Apple DOS".  I did find that I also needed a utility to locate all references to certain addresses.  FADD was the result, and it's mighty useful.  It's much quicker than doing a complete disassembly.

FADD will locate all instructions within 64K of memory referring to a given address.  It skips the $C000-$CFFF (I/O) pages and avoids missing memory by doing a double read test.

It is intended to be used by the serious assembly language programmer for debugging and analysis.  I'ts faster than doing a disassembly, though not quite so informative.

FADD is origined at $300 (what else?) and uses 8 zero page locations that are generally unused by programs except as scratch.  You can alter both the origin point and zero page locations to suit your individual needs.

To use FADD:

 1- BLOAD B.FADD
 2- Get to Monitor
 3- 'Fat finger' your address into 6-7 in HI-LO order.
 4- Execute with a '300G'

 NOTE: Use the spacebar to pause/release listing.

==============================================================================
DOCUMENT :AAL-8305:Articles:Front.Page.txt
==============================================================================

$1.50

Volume 3 -- Issue 8 May, 1983

In This Issue...



Yet Another Cross Assembler: PDP-11

We are turning the tables at last. When the 6502 was created six or seven years ago, programmers used PDP-11 development systems with cross assemblers to write 6502 code. Now you can use your Apple to write programs for the Digital Equipment Corporation's -11 family. Thanks to Martin Buchholtz for encouraging us to develop this one. He plans to use it for writing programs to run in DEC Falcon SBC-11 based systems. Only $50, if you already own the S-C Macro Assembler. See our ad on page 16 for more about the Cross Assemblers.

We Need Your Help

Does anybody have complete details of the file format of the Apple ///'s relocatable object files? That's the last remaining stumbling block on the road to the S-C Macro Assembler ///. Has anyone figured it out yet?

All Around the World

We are now sending the Apple Assembly Line to subscribers in 32 different countries. (That's about 1200 copies to the USA, and about 100 copies to the other 31 nations.)

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

Inc.)

==============================================================================
DOCUMENT :AAL-8305:Articles:Mikes80ColCmts.txt
==============================================================================

About the 80 column Macro Assembler versions.

The 80 column versions of the Assembler have several changes made in the editing section of the assembler.  Therefore, you must be aware when making patches to the assembler that the different versions will have different patch locations.

When using the ESCape key editing functions (keys A-F, IJKM, and @) you should exit the ESCape mode with some key other than right arrow. This is because the right arrow key will send the character beneath the cursor to the escape key processor and some funny things might happen to the screen if the key matches one of the valid ESCape key functions.

To exit from the assembler you should use PR#3 to unhook the assembler I/O hooks.  Then type FP, or INT.  If you do not do this and leave the assembler you could crash the computer.  This is especially critical in the Language card assembler versions.

To return to the assembler after using a printer, use RESET to turn off the printer and return to the assembler.  Although PR#3 appears to let you return to the assembler you will find that some of the keyboard editing functions may not work right because the assembler is not hooked into the I/O hooks.  The RESET will cause the assembler to hook itself back into the I/O hooks and return I/O to your 80 column card.

In using the MNTR command, the ESCape editing functions will not work from the apple monitor unless you type PR#3 to unhook the assembler before typing MNTR.  You should then return to the assembler with 3D0G or RESET to have the assembler rehook itself back into the I/O hooks.

```
==============================================================================
DOCUMENT :AAL-8305:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler (the best there is!)...........................$80.00
S-C Macro Assembler Version 1.1 Update.............................$12.50
    For registered owners of the S-C Macro Assembler.
Source code of S-C Assembler II, Version 4.0, on disk..............$95.00
    Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// .........................................$100.00
    Preliminary version.  Call or write for details.

S-C Cross Reference Utility .......................................$20.00
S-C Cross Reference Utility with Complete Source Code..............$50.00

S-C Word Processor.................................................$50.00
  As is, with fully commented source code.  Needs S-C Macro Assembler.
Applesoft Source Code on Disk......................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor....................$60.00

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983

Double Precision Floating Point for Applesoft......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)................... $79.00
Source Code for FLASH! Runtime Package.............................$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).......$49.00

The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00
Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
DISASM Dis-Assembler (Rak-Ware)....................................$30.00

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges...........each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender...........................................$20.00
Ashby Shift-Key Mod................................................$15.00
Lower-Case Display Encoder ROM.....................................$25.00
     Only Revision level 7 or later Apples.

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
```

```
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
    "6502 Subroutines", Leventhal........................($12.99)  $12.00

     Add $1 per book for US postage.  Foreign orders add postage needed.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We take Master Charge, VISA and American Express ***
```

```
==============================================================================
DOCUMENT :AAL-8305:Articles:New.Cards.txt
==============================================================================
```

Some New Cards

1. Bob Stout just called from Houston to renew his subscription to AAL, and to tell me about a new toy he's getting. It seems that Legend Industries has a new kind of RAM card, containing 18K of static RAM, with battery backup.

16K of the memory on the card is mapped just like a language card, so it can be used in slot 0. The card also has a hardware write-protect switch, that you can throw to completely protect the memory. Once you have done that whatever you have stored in the card is there to stay.

The card can also be used in a higher slot for boot-up operation. The other 2K of memory is mapped at $CN00 and $C800, just like the ROM on a standard peripheral card. Think of the possibilities!

This new card from Legend is available with either NiCad or Lithium batteries. This gives you a choice between rechargeability or very long power-off life (about 2 years). The price is $149.95.

2. Saturn Systems has introduced a card with 64K RAM and a 6502 on it. The CPU runs at 3.6 MHz, compared to Apple's roughly 1 MHz. Comes with a pre-boot disk to let you use this faster processor with Applesoft, Pascal, and Integer BASIC. Price is $599. See their ad in the latest Softalk Magazine.

3. Analytical Engines, Inc. has one-upped the DTACK Grounded board. For only $1550, you can plug in an 8 MHZ 68000 card with 128K RAM (expandable to 512K on the card!). You can upgrade to a 12.5 MHz chip if you really need it. DTACK is NOT grounded on this board, so you have access to the full 16-megabyte address space. The 16K ROM on the board contains monitor functions and diagnostics. YOu can replace the ROM with up to 64K of EPROM if you want. Software? The price includes a complete UCSD P-system (I think he said version 4.1) with Pascal, Basic, and Fortran compilers. You also get an Applesoft-compatible BASIC interpreter that runs entirely inside the 68000. CP/M-68 is optional, and Unix is supposed to be available soon. See their ad in the latest Nibble Magazine.

4. Lee Meador has designed a board with 64K RAM, 4K EPROM, and a 2MHz 6502 on it. This unique board does not talk directly to the Apple bus; instead, there are two parallel ports (I presume implemented with a 6522 chip). One 8-bit port talks to the Apple I/O bus, and the other is available to outside devices. Software runs on the board at 2MHz, and at the same time your Apple chips do their 1MHz processing. I can think of a lot of neat ways to use Lee's board, including as a printer buffer/controller, as a high-speed math processor, as a hard disk interface, and so on. If enough of you are interested, Lee will sell these for around $500 each, along with some demonstration

software.

```
==============================================================================
DOCUMENT :AAL-8305:Articles:ORDER.FORM.txt
==============================================================================

!A
S-C Software Corporation
2331 Gus Thomasson, #125
    Dallas, TX 75228
     (214) 324-2050
BE

Order Date:  ___/___/ 84                    Ship Date:  ___/___/ 84

Ship to:  ______________________________Bill to:
______________________________

          ______________________________
______________________________

          ______________________________
______________________________

          ______________________________
______________________________

Phone:    (_____) _____-____________      P/O #:
______________________________

Terms:  [  ]  Bill ___ days                 Ship by:
______________________________

        [  ]  COD

        [  ]  Charge MC/VISA  _______ _______ _______ _______   ___/___
________

        [  ]  Charge Am. Ex.  ______ _____________ __________   ___/___
________
                                                                exp
date  approv

Quantity   |                 Item                     |   Price   |
Total
-----------|--------------------------------------------|-----------|-
----------
1          |                                            |           |
           |                                            |           |
-----------|--------------------------------------------|-----------|-
----------
2          |                                            |           |
           |                                            |           |
-----------|--------------------------------------------|-----------|-
----------
```

```
3          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------
4          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------
5          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------
6          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------
7          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------
8          |                                          |           |
           |                                          |           |
-----------|------------------------------------------|-----------|-
----------

                                                     Total:    $

                                                  Shipping:    $

=============

                                                               $
```

```
==============================================================================
DOCUMENT :AAL-8305:Articles:Parity.txt
==============================================================================
```

Generating Parity..........................Bob Sander-Cederlof

When large amounts of data are being moved around it is easy to garble some. When you transmit characters over the telephone, or read them from a tape or disk, you want some kind of assurance that the message does not get modified by the medium.

Lots of schemes have been invented to prevent, detect, and even correct transmission errors: checksums, parity, cyclic redundancy codes, and more.

Checksums are used inside the Apple all the time. If you ever used cassette tapes with your Apple, you were re-assured to know that each program was recorded with a checksum. DOS 3.3 adds a checksum to the end of every sector on the disk. The checksum is re-computed when you read a tape or disk sector; if the result is different, at least one bit in the data is wrong.

Most of the checksums I have seen are of the exclusive-or type. All the bytes in the data record are EORed together, and the result is written at the end of the record. When the data is read, the in-coming bytes are again EORed together, and finally EORed with the checksum itself. If the final result is non-zero, an error occurred.

Checksums in the Apple are usually one byte wide. However, for more security, you could form a wider checksum. Or you could ADD the bytes together and store a two byte sum. Or store the complement of the sum, so that adding all the bytes plus the complement will give a zero result if there are no errors. [Checksums may check out OK even though errors occur, if the errors are sneaky enough to cancel each other out.]

Parity is really a kind of checksum, but only one bit wide. A series of bits is EORed together, and the single-bit result is the parity value. In an ASCII character there is provision for the leading bit position to be used for storing a parity bit. An eight-bit byte holds seven data bits plus a parity bit.

There are two kinds of parity in use: even and odd. Even parity makes the total number of 1-bits in the stream of bits even; odd parity makes the total number of 1-bits odd. Both even and odd are in use today in various kinds of equipment. Many terminals and serial communication boards allow you to select even, odd, or no parity. Looking at the ASCII code for a couple of letters, each could be transmitted in four ways:

```
Letter "M"  0 1 0 0 1 1 0 1 -- No parity, 8th bit always 0
            1 1 0 0 1 1 0 1 -- No parity, 8th bit always 1
            0 1 0 0 1 1 0 1 -- Even parity
```

```
            1 1 0 0 1 1 0 1 -- Odd parity

Letter "Q"  0 1 0 1 0 0 0 1 -- No parity, 8th bit always 0
            1 1 0 1 0 0 0 1 -- No parity, 8th bit always 1
            1 1 0 1 0 0 0 1 -- Even parity
            0 1 0 1 0 0 0 1 -- Odd parity
```

Sometimes I have needed a quick way to generate or verify a parity bit with software. These matters are usually handled in hardware, but not always.

In the 6502, it is a very simple matter to rotate a byte around and count the number of one bits present. Then the parity bit can be merged with the byte, or compared with what is already there.

The following subroutine (PARITY) computes the parity bit and merges it with the data byte. Call PARITY with the character to be merged in the A-register. Only the seven data bits will be counted. As written, the subroutine computes an odd parity bit. You can change line 1030 to "LDX #0" to compute even parity.

```
1000 *--------------------------------
1010 *   Compute and merge parity bit
1020 *--------------------------------
1030 PARITY LDX #1     (#0 FOR EVEN PARITY)
1040        ASL        SHIFT PARITY POSITION OUT
1050        PHA        SAVE SHIFTED CHARACTER
1060 .1     BPL .2     IF NEXT BIT = 0, DON'T COUNT
1070        INX        IF NEXT BIT = 1, COUNT IT
1080 .2     ASL        SHIFT IN NEXT BIT
1090        BNE .1     IF ANY REMAINING BITS = 1
1100        TXA        GET COUNT OF 1-BITS
1110        LSR        EVEN/ODD BIT OF COUNT INTO CARRY
1120        PLA        ORIGINAL CHAR BUT SHIFTED
1130        ROR        SHIFT PARITY BIT INTO BIT 8
1140        RTS
```

I wrote a little program to drive the PARITY subroutine, using all possible values from 0 through 127, and print out the results:

```
1150 *--------------------------------
1160 PRHEX  .EQ $FDDA  MONITOR PRINT (A) IN HEX
1170 COUT   .EQ $FDED  MONITOR PRINT CHAR IN (A)
1180 *--------------------------------
1190 DEMO   LDA #0     FOR CHAR = $00 TO $7F
1200        STA CHAR
1210 .1     JSR PARITY CALL THE PARITY SUBROUTINE
1220        JSR PRHEX  PRINT THE CHARACTER
1230        INC CHAR   NEXT CHAR
1240        LDA CHAR   SEE IF TIME FOR A NEW LINT
1250        AND #$07
1260        BEQ .2     YES
1270        LDA #$A0   <SPACE>
1280        BNE .3     ...ALWAYS
```

```
1290 .2     LDA #$8D   <RETURN>
1300 .3     JSR COUT   PRINT SPACE OR RETURN
1310        LDA CHAR
1320        BPL .1     STILL LESS THAN $80
1330        RTS        DEMO FINISHED
```

When I set it up for odd parity, here is part of the table printed out by DEMO:

```
80 01 02 83 04 85 86 07
08 89 8A 0B 8C 0D 0E 8F
10 91 92 13 94 15 16 97
   .
   .
   .

F8 79 7A FB 7C FD FE 7F
```

Now, how about a subroutine to check parity? Here is a version that checks an 8-bit value for odd parity. Simply change line 1420 to "LDX #0" to check for even parity instead. The subroutine returns with CARRY CLEAR for good parity, or CARRY SET for bad parity.

```
1400 *----------------------------
1410 CHECK.PARITY
1420        LDX #1     (OR #0 FOR EVEN PARITY)
1430        PHA        SAVE ORIGINAL CHAR
1440 .1     ASL        SHIFT NEXT BIT INTO CARRY
1450        BEQ .2     NO REMAINING 1-BITS
1460        BCC .1     LEADING BIT NOT 1-BIT
1470        INX        COUNT THE 1-BIT
1480        BCS .1     ...ALWAYS
1490 .2     BCC .3     LATEST SHIFTED BIT WAS 0
1500        INX        LATEST SHIFTED BIT WAS 1
1510 .3     TXA        BIT COUNT
1520        LSR        SHIFT EVEN/ODD BIT INTO CARRY
1530        PLA        RESTORE ORIGINAL CHARACTER
1540        RTS
```

=====================================================================
DOCUMENT :AAL-8305:Articles:Pause.Direct.txt
=====================================================================

A PAUSE Directive..................................Mike Laumer

Maybe your source code has outgrown even two disks and you need to know when to swap disks during assembly. Maybe you're using a single-sheet printer and need to change pages. Maybe you want to change typefaces on your letter-quality printer. Maybe you want to check the address of a routine or variable, without having to constantly watch the screen until it comes along. For whatever reason, you need to have the S-C Macro Assembler pause during assembly. Here is a new .US directive to let you do just that!

With this directive, you can insert a line like this anywhere in your code:

```
     1300        .US SWAP SOURCE DISK
```

In each pass, when the assembler encounters this line it will pause, display "SWAP SOURCE DISK" in inverse text at the bottom of the screen, beep twice, and wait for a keypress. You can take whatever action you need to, and press any key to resume assembly.

The listing is for the Language Card version of the assembler. If you are using the main memory version, you don't need to worry about write-enabling and -protecting, so you can just delete lines 1220, 1230 and 1280.

The values for the .EQ statements in lines 1170-1180 depend on whether you are using the Main Memory or the Language Card assembler, and whether you have Version 1.0 or 1.1. Here's a table of the values for US.VCTR and SC.CMNT:

| | Main Memory | Language Card | Version |
|---|---|---|---|
| US.VCTR | $100C | $D00C | Both |
| SC.CMNT | $1FD8 | $E124 | 1.0 |
| | $1FCA | $E0E4 | 1.1 |

That's all there is to it! Now you don't have to constantly stare at the screen during those long assemblies. Now you can sit back and wait for your Apple to call you when it needs you.

```
=============================================================================
DOCUMENT :AAL-8305:Articles:PDP11.XAsm.txt
=============================================================================
```

Yet Another Cross Assembler:  PDP-11

We are turning the tables at last.  When the 6502 was created six or seven years ago, programmers used PDP-11 development systems with cross assemblers to write 6502 code.  Now you can use your Apple to write programs for the Digital Equipment Corporation's -11 family. Thanks to Martin Buchholtz for encouraging us to develop this one.  He plans to use it for writing programs to run in DEC Falcon SBC-11 based systems.  Only $50, if you already own the S-C Macro Assembler.

```
==================================================================
DOCUMENT :AAL-8305:Articles:Rogram.2.Large.txt
==================================================================
```

ROGRAM TOO LARGE???..................................Lee Meador

I was writing an ampersand file-handling routine, using the File Manager in DOS as described in chapter 6 of Beneath Apple DOS. I wanted my Applesoft program to be able to set ONERR and catch errors in files (wrong name, too short, etc.) But I also wanted the error messages to come out in immediate mode or with no ONERR set. Since I was doing my own file-handling, I was going to have to provide my own error outputs. Originally I tried this:

```
ERROR  LDY #$0A        error code offset
       LDA ($04),Y     $04 -> FM parmlist
       JMP $A6D2       jump into DOS error handler
```

This worked OK when used in code that was called from an Applesoft program, but when I called it in immediate mode (from the "]") I would always get "ROGRAM TOO LARGE" when an error occurred.

You might guess that I have found a solution. The problem is caused when we jump into DOS at $A6D2 with the IO hooks still pointing into DOS. The routine starting at $A6D2 saves the error code in a temporary location at $AA5C and calls $A702 to print a "<return><beep><return>". Since we entered illegally that output goes to DOS at $9EBD, then via a JSR to $9ED1 where the accumulator is saved in a temporary location -- $AA5C! This leaves that last <return> in $AA5C.

When control returns to the error handler DOS then tries to look up error message number 141 ($8D) in the 16-entry table of offsets starting at $AA3F. This loads the offset from location $AACC, which happens to contain the high-order byte of the address of the OPEN command handler ($AB22)! This leaves the error message printer with an offset of $AB into the messages at $A971. And that is what produces "ROGRAM TOO LARGE". Look at the routines at $A6D2 and $A702 for more detail. $A702 is meant to be called with the error code in the X register.

Now here's a method to have the error handled correctly:

```
OUT.HOOK  .EQ $36
HARD.COUT .EQ $FDF0

ERROR  LDA #HARD.COUT  point hook out of DOS
       STA OUT.HOOK
       LDA /HARD.COUT
       STA OUT.HOOK+1  DOS will fix it back
       LDY #$0A        index to error code
       LDA ($04),Y     $04 -> FM parmlist
       JMP $A6D2       do it ...
```

That takes care of getting the right error messages.  Now if I could just figure out some way to make sure that no errors ever occur. . .

```
=======================================================================
DOCUMENT :AAL-8305:Articles:SC.Capture.txt
=======================================================================
```

S-C CAPTURE -- A Modem Program for the Word Processor..........
Jim Church

If you like to sign on to the The Source or CompuServe or some such system, you should get a copy of the S-C Word Processor.  I like to receive the programs from CALL-A.P.P.L.E. magazine by modem and the S-C Word Processor really makes that easy.

What you do is quite simple.  Just put a copy of B.SC.CAPTURE on the disk with the Word Processor.  Then, whenever you want to capture a session with a remote system, you can choose D from the word processor menu and BLOAD B.SC.CAPTURE.  After the routine is loaded, return to the main menu and choose L to load a sign-on file containing the commands necessary to dial the number you want to call.  Here is a sample sign-on file, which I use to call up The Source.

```
       !pr2
       Q_*367-6021      (The Q_ is a Control-Q)
       !pr768
```

Now choose P from the menu, and your word processor will start dialing the phone!  From here on you just operate the remote system as usual. The top line of the screen will show the address where characters are being stored, and the rest of the screen shows the text you are entering and receiving.

When you want to quit, just type a Control Z to hang up your phone and return to the word processor's main menu.  Select E and you will see a copy of everything that transpired.  Now you can edit the text however you want to, and save it all to your disk.

The !pr768 command above is intended to provide a hook for a user-written printer driver.  It sets the output hook at $36-37 to $300. The next time the Word Processor tries to output a character, it wakes up the capture routine, which completely takes over until it is turned off with a Control Z.  This is slightly abusing the !pr directive, so if you follow this example for other routines, be sure to have lines like 1570-1590 at the beginning of your routine, and exit to $803 at the end, so the Word Processor can reconnect itself correctly.

That's all there is to it.  You could probably do a lot to "smarten up" this dumb terminal program.  The way I have done it, it recognizes a Control Z from the keyboard and filters out incoming Control J's. That's all it does.  Probably it should filter out Control G too, at the very least.  My intention is to demonstrate the simple fact that the word processor is a very versatile creature.

This works, the way it is, with the D. C. Hayes Micromodem II in Slot 2. If your modem is in a different slot, just change line 1260 to show the correct slot number.

```
==============================================================================
DOCUMENT :AAL-8305:DOS3.3:S.DispCharSet.txt
==============================================================================

 1000  *SAVE S.DISPLAY CHAR SET
 1010  *--------------------------------
 1020  *      DISPLAY CHARACTER SET
 1030  *--------------------------------
 1040  CNT8       .EQ $00
 1050  B          .EQ $01
 1060  CNT16      .EQ $02
 1070  *--------------------------------
 1080  EPROM.A.IMAGE  .EQ $6800
 1090  EPROM.B.IMAGE  .EQ $7000
 1100  *--------------------------------
 1110  AS.HGR     .EQ $F3E2
 1120  *--------------------------------
 1130         .OR $803
 1140  DISPLAY
 1150  *---TURN ON HI-RES GRAPHICS------
 1160         LDA $C081    GET A/S ROMS ON MOTHERBOARD
 1170         JSR AS.HGR
 1180         LDA $C080    BACK TO S-C ASM IN RAM CARD
 1190  *---FIRST CHAR SET---------------
 1200         LDA /$2000   TOP LINE, LEFT SIDE
 1210         STA SCREEN.ADR+1
 1220         LDA #$2000
 1230         STA SCREEN.ADR
 1240         LDA /EPROM.A.IMAGE  FIRST CHARACTER SET
 1250         STA EPROM.ADR+1
 1260         LDA #EPROM.A.IMAGE
 1270         STA EPROM.ADR
 1280         JSR DISPLAY.ONE.SET
 1290  *---SECOND CHAR SET--------------
 1300         LDA /$2014   TOP LINE, RIGHT SIDE
 1310         STA SCREEN.ADR+1
 1320         LDA #$2014
 1330         STA SCREEN.ADR
 1340         LDA /EPROM.B.IMAGE  SECOND CHARACTER SET
 1350         STA EPROM.ADR+1
 1360         LDA #EPROM.B.IMAGE
 1370         STA EPROM.ADR
 1380         JSR DISPLAY.ONE.SET
 1390  *---PAUSE UNTIL KEYSTROKE--------
 1400  .1     LDA $C000
 1410         BPL .1
 1420         STA $C010
 1430         RTS          RETURN TO ASSEMBLER
 1440  *--------------------------------
 1450  *      DISPLAY ONE CHARACTER SET IN 16-BY-16 FORMAT
 1460  *--------------------------------
 1470  DISPLAY.ONE.SET
 1480         LDA #16      COUNT 16 ROWS
```

```
1490        STA CNT16
1500 .1     JSR DISPLAY.ONE.ROW
1510 *---NEXT ROW IN EPROM DATA-------
1520        CLC
1530        LDA EPROM.ADR
1540        ADC #15*8
1550        STA EPROM.ADR
1560        LDA EPROM.ADR+1
1570        ADC #0
1580        STA EPROM.ADR+1
1590 *---NEXT ROW ON SCREEN-----------
1600        SEC
1610        LDA SCREEN.ADR
1620        SBC #$2000-$80
1630        STA SCREEN.ADR
1640        LDA SCREEN.ADR+1
1650        SBC /$2000-$80
1660        STA SCREEN.ADR+1
1670        CMP #$24          HIT THE BREAK YET?
1680        BCC .2            NO, GO ON
1690        LDA SCREEN.ADR    YES, ADJUST THE ADDRESSES
1700        SBC #$400-$28
1710        STA SCREEN.ADR
1720        LDA SCREEN.ADR+1
1730        SBC /$400-$28
1740        STA SCREEN.ADR+1
1750 .2     DEC CNT16    LAST ROW YET?
1760        BNE .1       ...NO
1770        RTS          ...YES, RETURN
1780 *--------------------------------
1790 *      DISPLAY ONE ROW OF 16 CHARACTERS
1800 *--------------------------------
1810 DISPLAY.ONE.ROW
1820        LDA #8       8 SCREEN LINES FOR ONE ROW
1830        STA CNT8
1840 .1     LDY #0       EPROM DATA INDEX
1850        LDX #0       SCREEN IMAGE INDEX
1860 .2     JSR GET.PUT  MOVE ONE BYTE TO SCREEN
1870        TYA          ADD 8 TO EPROM DATA INDEX
1880        CLC
1890        ADC #8
1900        TAY
1910        INX          BUMP SCREEN IMAGE INDEX
1920        CPX #16
1930        BCC .2       MORE CHARACTERS
1940        INC EPROM.ADR  BUMP TO NEXT LINE OF EPROM DATA
1950        LDA SCREEN.ADR+1   +$400
1960        ADC #3       (CARRY = 1)
1970        STA SCREEN.ADR+1
1980        DEC CNT8     NEXT SCREEN LINE
1990        BNE .1       ...IF ANY
2000        RTS          RETURN
2010 *--------------------------------
2020 *      REVERSE THE ORDER OF BITS 6-0 IN A-REG
```

```
*       (CHANGE XABCDEFG TO 0GFEDCBA)
*--------------------------------
REVERSE.BITS
        LSR          REVERSE 7 BITS
        ROL B        A=0XABCDEF  B=XXXXXXXG
        LSR
        ROL B        A=00XABCDE  B=XXXXXXGF
        LSR
        ROL B        A=000XABCD  B=XXXXXGFE
        LSR
        ROL B        A=0000XABC  B=XXXXGFED
        LSR
        ROL B        A=00000XAB  B=XXXGFEDC
        LSR
        ROL B        A=000000XA  B=XXGFEDCB
        LSR
        ROL B        A=0000000X  B=XGFEDCBA
        LDA B
        AND #$7F     0GFEDCBA
        RTS
*--------------------------------
*       PICK UP A BYTE OF EPROM DATA,
*       REVERSE THE BITS, AND STORE
*       IT ON THE SCREEN.
*--------------------------------
GET.PUT
        LDA $FFFF,Y
EPROM.ADR .EQ *-2
        JSR REVERSE.BITS
        STA $FFFF,X
SCREEN.ADR .EQ *-2
        RTS
        .LIF
```

```
==============================================================================
DOCUMENT :AAL-8305:DOS3.3:S.FADD.txt
==============================================================================

 1000  *********************************
 1010  *                               *
 1020  *           F A D D             *
 1030  *                               *
 1040  *  ( FIND ADDRESS REFERENCES )  *
 1050  *   -------------------------   *
 1060  *                               *
 1070  *    A PUBLIC DOMAIN UTILITY    *
 1080  *                               *
 1090  *     BY.. BROOKE W BOERING     *
 1100  *                               *
 1110  *********************************
 1350
 1360  * TO USE:
 1370  * 1- BLOAD FADD.OBJ
 1380  * 2- GET TO MONITOR
 1390  * 3- 'FAT FINGER' YOUR ADDRESS
 1400  *    INTO 6-7 IN HI-LO ORDER.
 1410  *   NOTE ------> ^^ ^^ <-------
 1420  * 4- EXECUTE WITH A '300G'
 1430
 1460  *---------------------------------
 1470  *         E Q U A T E S
 1480
 1490  TARGHI .EQ $6
 1500  TARGLO .EQ $7
 1510  * NOTE: ABOVE REVERSES NORMAL LO/HI-BYTE
 1520  *  ORDER FOR EASIER KEYIN FROM MONITOR
 1530  WHER   .EQ $8
 1540  WHERLO .EQ $8
 1550  WHERHI .EQ $9
 1560
 1570  LENGTH  .EQ $2F
 1580  PCL     .EQ $3A
 1590  PCH     .EQ $3B
 1600  COLOR   .EQ $30
 1610
 1620  INSDS2  .EQ $F88E
 1630  INSTDSP .EQ $F8D0
 1640  PCADJ3  .EQ $F956
 1650  CROUT   .EQ $FD8E
 1660  *---------------------------------
 1670         .OR $300
 1680  *      .TF B.FADD
 1690  *---------------------------------
 1700  START
 1710
 1720   LDX #0
 1730   STX WHERLO   START AT BEGINNING
```

```
1740   STX WHERHI   OF MEMORY
1750
1760  *-- CHECK FOR DIRECT REFERENCE
1770  .1
1780   LDY #0
1790   LDA (WHER),Y GET WHERAT-LO
1800   STA COLOR    SAVE TEMP
1810   LDA (WHER),Y GET IT AGAIN
1820   CMP COLOR    STILL THE SAME?
1830   BNE .8       NO, SKIP IT, NO MEMORY HERE
1840  * (FALL THROUGH IF MEMORY AT THIS ADDRESS)
1850
1860   CMP TARGLO   ? TARGET-LO ?
1870   BNE .3       NO, GO AHEAD
1880   INY
1890   LDA (WHER),Y GET WHERAT-HI
1900   CMP TARGHI   ? TARGET-HI ?
1910   BNE .3       NO, GO AHEAD
1920  * (FALL THROUGH IF 2-BYTE MATCH ON TARGET)
1930
1940  *-- APPARENT MATCH;
1950  *      MAKE SURE IT'S A 3-BYTE INSTRUCTION
1960  .2
1970   LDY WHERHI   GET ADDRESS OF MATCH
1980   LDX WHERLO
1990   BNE .24
2000   DEY          POINT TO INSTRUCTION BYTE
2010  .24
2020   DEX
2030   STX PCL      AND SET PROGRAM COUNTER
2040   STY PCH
2050
2060   LDX #0
2070   LDA (PCL,X)  GET OPCODE
2080   JSR INSDS2   USE MONITOR DISASSEMBLER ROUTINE
2090   LDA LENGTH
2100   CMP #2       3-BYTE INSTRUCTION?
2110   BEQ .6       OK; GO AHEAD TO DISPLAY
2120  * (FALL THROUGH WHEN NOT A 3-BYTE INSTR)
2130
2140  *-- CHECK FOR RELATIVE BRANCH
2150  .3
2160   LDY #0
2170   LDA (WHER),Y GET INSTRUCTION BYTE
2180   AND #$1F     ISOLATE SIGNIFICANT BITS
2190   CMP #$10     A BRANCH INSTRUCTION?
2200   BNE .8       DEFINITELY NOT
2210  * (FALL THROUGH WHEN A BRANCH INSTRUCTION)
2220
2230  *-- TEST IF BRANCHING TO TARGET
2240  *      NOTE: USING MONITOR TECHNIQUE
2250  .4
2260   LDX WHERLO   PRESET FOR PCADJ3
2270   LDY WHERHI
```

```
2280   STX PCL      SET PC TO OPCODE BYTE
2290   STY PCH
2300   LDY #1
2310   LDA (WHER),Y GET OFFSET BYTE
2320   JSR PCADJ3   LEAVES EFFECTIVE ADDRESS-1
2330  *             IN Y AND A
2340   TAX
2350   INX
2360   BNE .43
2370   INY
2380  .43
2390  *-- NOW 'BRANCH TO' ADDRESS IS IN Y AND X
2400   CPX TARGLO
2410   BNE .8
2420   CPY TARGHI
2430   BNE .8
2440  * (FALL THROUGH ON MATCH)
2450
2460  *-- DISPLAY MATCHED INSTRUCTION
2470  .6
2480  * PCL/PCH ARE SET
2490   JSR INSTDSP  <= MONITOR ROUTINE
2500
2510  *-- ALLOW KEYED PAUSE/RELEASE
2520  .7
2530   BIT $C000    KEY DOWN?
2540   BPL .8       NO, GO AHEAD
2550   BIT $C010    YES, CLEAR STROBE
2560  .77
2570   BIT $C000    RELEASED?
2580   BPL .77      NO, LOOP TIL SO
2590   BIT $C010    YES, CLEAR STROBE
2600
2610  *-- POST DISPLAY (OR NO MATCH)
2620  .8
2630   INC WHERLO   KICK ADDRESS
2640   BNE .1       LOOP 255 OF 256
2650   INC WHERHI   KICK ADDR PAGE#
2660   BEQ .9       EXIT AT 65536 OVFLO
2670
2680  *-- AT NEW PAGE !!
2690   LDA WHERHI
2700   CMP #$C0     AT THE I/O PORTS ?
2710   BNE .1       NO, LOOP BACK
2720   LDA #$D0     YES, SKIP 'EM
2730   STA WHERHI   : (AVOID PROBLEMS)
2740   BNE .1       LOOP BACK
2750
2760  .9
2770   JMP CROUT    RETURN THROUGH CROUT
```

```
==============================================================================
DOCUMENT :AAL-8305:DOS3.3:S.PARITY.txt
==============================================================================

 1000  *--------------------------------
 1010  *      DEMONSTRATE PARITY SUBROUTINE
 1020  *--------------------------------
 1030  PRHEX  .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  DEMO   LDA #0       FOR CHAR = $00 TO $7F
 1060         STA CHAR
 1070  .1     LDA CHAR
 1080         JSR PARITY   MERGE WITH PARITY BIT
 1090         JSR PRHEX    PRINT AS TWO HEX DIGITS
 1100         LDA #$A0     SPACE
 1110         JSR COUT     SEPARATE WITH TWO SPACES
 1120         JSR COUT
 1130         INC CHAR     NEXT CHAR
 1140         BPL .1
 1150         RTS          RETURN
 1160  *--------------------------------
 1170  CHAR   .BS 1
 1180  *--------------------------------
 1190  *      COMPUTE PARITY BIT AND MERGE WITH CHAR
 1200  *      CALL:  (A) = 7-BIT CHARACTER, HIGH BIT IGNORED
 1210  *    RETURN:  (A) = SAME CHARACTER, WITH PARITY IN HIGH BIT
 1220  *--------------------------------
 1230  PARITY LDX #1       OR #0 FOR EVEN PARITY
 1240         ASL          SHIFT PARITY POSITION OUT
 1250         PHA          SAVE SHIFTED CHAR
 1260  .1     BPL .2       IF NEXT BIT = 0
 1270         INX          IF NEXT BIT = 1, COUNT IT
 1280  .2     ASL          SHIFT IN NEXT BIT
 1290         BNE .1       IF ANY REMAINING BITS = 1
 1300         TXA          GET 1-BIT COUNT
 1310         LSR          PUT PARITY BIT INTO CARRY
 1320         PLA          ORIGINAL CHAR BUT SHIFTED
 1330         ROR          MERGE PARITY BIT
 1340         RTS
```

```
==============================================================================
DOCUMENT :AAL-8305:DOS3.3:S.PauseDirect.txt
==============================================================================
```

```
1000  *--------------------------------
1010  *   .US DIRECTIVE TO PAUSE DURING ASSEMBLY
1020  *
1030  *      SYNTAX:  .US <phrase>
1040  *      RESULT:  Displays <phrase> in inverse text
1050  *               and waits for a keypress
1060  *
1070  *--------------------------------
1080  CHR.PTR  .EQ $7B
1090  WBUF     .EQ $200
1100  CORNER   .EQ $7D0
1110  KEYBOARD .EQ $C000
1120  STROBE   .EQ $C010
1130  PROTECT  .EQ $C080
1140  ENABLE   .EQ $C083
1150  BELL     .EQ $FBE2
1160
1170  US.VCTR  .EQ $D00C
1180  SC.CMNT  .EQ $E124
1190  *--------------------------------
1200         .OR $300
1210  *--------------------------------
1220         LDA ENABLE       WRITE ENABLE
1230         LDA ENABLE       RAM CARD
1240         LDA #PAUSE
1250         STA US.VCTR+1    POINT .US VECTOR
1260         LDA /PAUSE
1270         STA US.VCTR+2    TO PAUSE ROUTINE
1280         LDA PROTECT      PROTECT CARD
1290         RTS
1300  *--------------------------------
1310  PAUSE  JSR BELL     BEEP
1320         LDX #0
1330         LDY CHR.PTR  CHAR POINTER
1340  .1     LDA WBUF,Y   GET CHAR FROM CALL LINE
1350         BEQ .2       END OF LINE?
1360         AND #$3F     NO, INVERT CHAR
1370         STA CORNER,X AND PUT IT AT BOTTOM OF SCREEN
1380         INX
1390         INY
1400         CPX #40      LINE FULL?
1410         BCC .1       NO, GET ANOTHER CHAR
1420
1430  .2     JSR BELL     BEEP
1440  .3     LDA KEYBOARD
1450         BPL .3       WAIT FOR KEYPRESS
1460         STA STROBE
1470         JMP SC.CMNT  RETURN TO ASSEMBLY
1480  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8305:DOS3.3:S.SC.CAPTURE.txt
==============================================================================

 1000  *--------------------------------
 1010  *
 1020  *           S-C CAPTURE
 1030  *
 1040  *   A COMMUNICATIONS MODULE FOR
 1050  *     THE S-C WORD PROCESSOR
 1060  *
 1070  *           BY JIM CHURCH
 1080  *
 1090  *--------------------------------
 1100  * FULL DUPLEX CAPTURE PROGRAM
 1110  * WORKS WITH MICROMODEM II
 1120  * AND S-C WORD PROCESSOR
 1130  *
 1140  * GO INTO EDITOR W/EMPTY BUFFER
 1150  * ENTER COMMANDS AS FOLLOW:
 1160  *
 1170  * !pr2
 1180  * Q*367-6021    THE "Q" IS A CONTROL-Q
 1190  * !pr768
 1200  *
 1210  * LEAVE EDITOR, CHOOSE P ON MENU
 1220  *--------------------------------
 1230         .OR $300
 1240         .TF B.SC.CAPTURE
 1250
 1260  SLOT   .EQ $02
 1270  SLOT16 .EQ SLOT*16
 1280
 1290  PTR    .EQ $00
 1300  WNDTOP .EQ $22
 1310  CH     .EQ $24
 1320
 1330  HOOK   .EQ $3EA
 1340
 1350  BUFFER .EQ $2000
 1360
 1370  KEYBOARD  .EQ $C000
 1380  STROBE    .EQ $C010
 1390  MM.CR2    .EQ $C085+SLOT16
 1400  MM.STATUS .EQ $C086+SLOT16
 1410  MM.DATA   .EQ $C087+SLOT16
 1420
 1430  PRNTAX .EQ $F941
 1440  INIT   .EQ $FB2F
 1450  VTAB   .EQ $FC22
 1460  VTABZ  .EQ $FC24
 1470  HOME   .EQ $FC58
 1480  COUT1  .EQ $FDF0
```

```
1490  SETKBD .EQ $FE89
1500  SETVID .EQ $FE93
1510  *--------------------------------
1520  SC.CAPTURE
1530         JSR INIT     FIX SCREEN
1540         JSR HOME     CLEAR SCREEN
1550         LDA #1       RESERVE TOP LINE
1560         STA WNDTOP    FOR LOCATION COUNTER
1570         JSR SETVID   PR#0
1580         JSR SETKBD   IN#0
1590         JSR HOOK     TELL DOS
1600         LDX #0       WORD PROCESSOR
1610         STX BUFFER    NEEDS 0 AT $2000
1620         INX
1630         STX PTR      START POINTER
1640         LDA /BUFFER   AT $2001
1650         STA PTR+1
1660
1670  TERMINAL
1680         LDA KEYBOARD KEY DOWN?
1690         BPL MODEM    NO, CHECK MODEM
1700         STA STROBE   YES, CLEAR STROBE
1710         CMP #$9A     CONTROL Z?
1720         BEQ QUIT     YES, LEAVE
1730         PHA          SAVE KEYPRESS
1740  .1     LDA MM.STATUS CHECK IF THE TRANSMIT
1750         AND #$02       REGISTER EMPTY BIT IS SET
1760         BEQ .1       NO, WAIT FOR IT
1770         PLA          YES, GET KEY BACK
1780         STA MM.DATA  SEND IT
1790         BMI TERMINAL  AND LOOP AGAIN
1800
1810  MODEM  LDA MM.STATUS CHECK IF THE RECEIVER
1820         AND #$01       REGISTER FULL BIT IS SET
1830         BEQ TERMINAL NO, LOOP AGAIN
1840         LDA MM.DATA  YES, GET CHARACTER
1850         ORA #$80     SET HI BIT
1860         CMP #$8A     CONTROL J?
1870         BEQ TERMINAL IGNORE IT
1880         JSR COUT1    PRINT CHAR
1890         LDY #0
1900         STA (PTR),Y  CAPTURE IT IN BUFFER
1910
1920  INCR   INC PTR      BUMP POINTER LO
1930         BNE COUNT     NOT 0
1940         INC PTR+1    BUMP POINTER HI
1950         LDA PTR+1    CHECK IF
1960         CMP #$96      BUFFER END?
1970         BCS QUIT     FULL BUFFER, LEAVE
1980
1990  COUNT  LDA CH       SAVE CH
2000         PHA           ON STACK
2010         LDA #0       TOP LINE
2020         JSR VTABZ     FOR LOCATION COUNTER
```

```
2030        LDA #$14     COL 20
2040        STA CH        IN CH
2050        LDA PTR+1    HI BYTE OF LOCATION
2060        LDX PTR      LO BYTE
2070        JSR PRNTAX   PRINT ADDRESS
2080        PLA          GET OLD CH AND RETURN
2090        STA CH        TO WHERE WE WERE
2100        JSR VTAB     OLD LINE
2110        BCC TERMINAL START OVER
2120
2130  QUIT  LDA #$00     END-OF-TEXT MARKER
2140        STA (PTR),Y   FOR WORD PROCESSOR
2150        LDA #$05     HANG UP PHONE
2160        STA MM.CR2    AT CONTROL REGISTER
2170        JMP $803     COLDSTART WORD PROCESSOR
2190        .LIF
```

==============================================================================
DOCUMENT :AAL-8307:Articles:Cross.Ad.txt
==============================================================================

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

| | | | |
|---|---|---|---|
| Motorola: | 6800/6801/6802 | now | $32.50 |
| | 6805 | now | $32.50 |
| | 6809 | now | $32.50 |
| | 68000 | now | $50.00 |
| Intel: | 8048 | now | $32.50 |
| | 8051 | now | $32.50 |
| | 8085 | now | $32.50 |
| Zilog: | Z-80 | now | $32.50 |
| RCA: | 1802/1805 | now | $32.50 |
| Rockwell: | 65C02 | now | $20.00 |
| DEC: | PDP-11/LSI-11 | now | $50.00 |

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstanding support: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation   (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

==============================================================================
DOCUMENT :AAL-8307:Articles:FastTextFileIO.txt
==============================================================================

Paul Schlyter's DOS patch
Speeding-up Text File I/O......................Paul Schlyter

In the April 1983 AAL (pages 2-8), Bob Sander-Cederlof presented a small patch that I had sent him almost a year earlier. The patch greatly speeded up LOAD/BLOAD of long files. At the moment, I had recreated a lot of very long assembler source files, such as the source code to DOS and Applesoft. The long assembly times grew annoying, especially when I realized how much time was wasted inside RWTS just waiting for the right sector to pass under the R/W head of the disk drive!

Just one note about what was written on the bottom of page 2 of that issue: my patch does not influence SAVE/BSAVE at all. The read-after-write during a SAVE/BSAVE is made using the VERIFY command, and that command already works at top speed; in fact, VERIFY's speed was a major inspiration for my LOAD/BLOAD patch.

Next I tried to speed up SAVE/BSAVE with an equally simple patch. I found it was not so easy, mainly because SAVE/BSAVE might have to allocate new sectors for the file. I also felt it wasn't worth the trouble writing a more complicated patch, since SAVE/BSAVE isn't really used that often.

Next in line was a speedup of text file read and write. Here I found a great "time-hog" in DOS. The innocent-looking routines at $AE68 and $AE7E each require about 800 cycles to execute. All they do is to swap a 45-byte area back and forth between the file buffers and a local workarea inside the file manager. This is of course necessary when you open/close files or switch from file to file. But if you're reading the same text file, the swapping may not be needed. Nevertheless, file manager swaps the buffer in and out for each and every character you read or write! This amounts to 256*(800+800) = roughly 410,000 cycles or 0.4 seconds for each sector you read or write! This is about six seconds for each track! And all it does during those six seconds is needlessly swap the same 45 bytes back and forth!

The principle of my patch is this: When entering or exiting the file manager, first check to see if you're doing something else besides reading/writing. If so, just go on as usual. If you are reading/writing, check to see if the local workarea belongs to the file being read/written. If so, just exit and save 800 clock cycles. If not, check to see if it belongs to another file. If the workarea contains another file's data, put it back into the file buffer where it belongs and then get the workarea for the current file. All occurs this when you enter the file manager.

Upon exit from the file manager, if you're reading/writing, just set a flag to mark that the local workarea is being used, and save the address of the file buffer it came from.  This always saves 800 cycles.

Practical tests show that text file reading/writing is done up to about 40% faster with this patch installed.  This is slower than Diversi-DOS, but on the other hand this patch is compatible with S-C assemblers (and almost everything else in sight).  Also, this patch works equally well for all file types; it even speeds up the loading of type-R files with RBOOT/RLOAD (from DOS Tool Kit).  Diversi-DOS treats T type files in a special way, but does nothing to speed up type-R files.  And mine is free!

I put the patch at $300, because there's no free area large enough inside DOS where you can put it...especially if you have already installed the LOAD/BLOAD speedup described by Bob last April.  The listing which follows includes code to hook in the patches by overwriting the file manager where it calls the two workarea transfer subroutines.

Making Paul's Patches Fit in DOS...........Bob Sander-Cederlof

Don't tell me it won't fit!  It is so good, it MUST fit!

Let's see...there are 74 bytes available from $B6B3 thru $B6FC.  But Paul's patches are 93 bytes long.  Maybe if I twist it sideways and then hold my mouth just right....

Ha!  It worked!

Let me tell you how, but please don't think I am trying to pick Paul apart.  His analysis and creative programming are terrific!  He has taught me a lot.

First I noticed some common code in PATCH1 and PATCH2.  I made a subroutine called CHECK.OPCODE to test for the read or write command. I used the carry status to pass back the answer to the caller.  Then I put the call to POINT.TO.WORKAREA (which loads an address into $42 and $43) at the top of the subroutine.  There's no need to duplicate it in the two callers.  These changes saved two or three bytes, for a tiny penalty in speed.

I noticed Paul used CLC, ROR FLAG to clear the sign bit of FLAG.  I save one byte two times by replacing these with LSR FLAG.  I set up the carry status info in CHECK.OPCODE so that carry SET means read/write...this lets me omit the SEC before ROR FLAG when I want to turn on the sign bit.

I noticed that both patches used the current contents of PNTR:  PATCH1 compared PNTR to PNTR.SAVE, while PATCH2 copied PNTR into PNTR.SAVE. So I loaded up the contents of PNTR into the A- and X-registers inside my CHECK.OPCODE subroutine.  This saves a few more bytes.

At lines 1320-1330 in Paul's program he uses BNE to jump around an RTS. I changed that to BEQ to an existing RTS further down in the program, saving one byte.

I moved the PNTR.SAVE variable, two bytes, to another area. $B5CF and $B5D0 are unused, at the end of the file manager parameter list. Conveniently, the subroutines which load addresses into PNTR refer to three such addresses inside the parameter list. (See the code at $AF08-$AF1C.) The X-register is loaded with 0, 2, or 4 to index into the list. By putting PNTR.SAVE at the end of the list, I can load the X-register with 8 (PNTR.SAVE-$B5C7) and use the same subroutine, entering at $AF12. This takes five bytes instead of twelve for LDA-STA- LDA-STA.

The final shortener I applied was to make the code which clears FLAG and copies the workarea to a buffer into a subroutine. This is called PATCH4 in my listing. The two lines at PATCH4 look just like what was in line inside the PATCH1 code, but different from what was done by the PATCH2 code.

PATCH2 falls into PATCH4 if the opcode was not read/write. This used to clear the flag and call $AE7E; now it is $AE81. Since the difference between $AE7E and $AE81 is a JSR to setup PNTR with the workarea address, and since that was already arranged by CHECK.OPCODE, I can safely enter at $AE81.

No doubt if you followed me this far, you can see even more ways to save bytes. In fact, I see one extra byte myself! But the program is now just the right size for that hole at $B6B3, so enough is enough.

My listing includes some code to install the patches. If you assemble my version, and BSAVE it on a binary file (A$300,L$6A), you can BRUN it whenever you want to install the patches. Or, with version 1.1 of the Macro Assembler just add these lines:

```
1195       .TF B.FAST TEXT
1380       .PH $B6B3
1790       .EP
```

I also worked out the code for using Applesoft POKEs to patch it all in, and here it is:

```
100  REM  TEXT FILE SPEEDUP PATCH
110  READ N
   : IF N = 0 THEN  END
120  READ A
   : FOR I = 0 TO N - 1
   : READ X
   : POKE A + I,X
   : NEXT
   : GOTO 110
200  DATA 74,46771,32,210,182,144,10,205,207,181,
          208,5,236,208,181,240,51,44,252,182,16,
          8,162,8,32,18,175,32,246,182,76,106,174,
```

```
          32,8,175,173
210  DATA 187,181,56,73,3,240,5,73,7,240,1,24,165,
          66,166,67,96,32,210,182,144,10,110,252,182,
          141,207,181,142,208,181,96,78,252,182,76,
          129,174,0
220  DATA 2,43787,179,182
230  DATA 2,45967,231,182
240  DATA 0
```

I tested the patches on a 24-sector text file. The file was created by using the TEXT command in the S-C Macro Assembler. I used EXEC to read it back in. I also wrote a short Applesoft program which read the whole file with GET A$ in a loop. Here are the results:

| | NORMAL | PATCHED | CHANGE |
|---|---|---|---|
| TEXT | 24 sec | 18 sec | 25% faster |
| EXEC | 52 sec | 34 sec | 35% faster |
| GET A$ | 30 sec | 21 sec | 30% faster |

I think you get the most benefit if the un-patched DOS has to work so long between calls to RWTS that the disk motor stops, but the patched DOS keeps the motor alive. You save 0.4 seconds per sector anyway, but you can also save waiting for the motor to come up to speed.

Warning: One danger I noted, and which I am wary of, is that FLAG could get out of sync with reality. For example, if somehow FLAG was set with the sign bit on before ever calling the file manager, it could try to copy the workarea to any-old-place in RAM (or ROM, or I/O space). If you install the patches after booting, there should be no problem. But what happens if you initialize a disk with the patched DOS? I think the flag MIGHT turn out wrong. Maybe a little patch is needed to insure FLAG starts out clear, and is cleared after abnormal exits from file manager (such as RESET).

==============================================================================
DOCUMENT :AAL-8307:Articles:Feature.txt
==============================================================================

Assembly Listing Into a Text File...................Bill Morgan

"That's not a bug, that's a feature!"  We've all heard (or said?) that before, but this time it really seems to be true.  We have just discovered an undocumented feature in Version 1.1 of the S-C Macro Assembler.

I was trying to see if a program would assemble, and wanted the assembly to be as fast as possible.  For some reason I didn't want to do the obvious thing and just switch the listing off, and using a .TA wasn't convenient.  So, I stuck a .DU (DUmmy) directive at the beginning of my program, and a .ED (End Dummy)at the end, figuring that would eliminate the time spent writing object code to the disk. When I typed ASM the assembler paused a moment for pass one, then started listing the beginning of the program.  But, when the assembler got to the .TF directive the listing stopped and the disk drive started spinning.  That wasn't supposed to happen!

When the assembly finished, and the drive stopped turning, I CATALOGed the disk to see what had happened.  There was a Binary file, with the filename from my .TF directive, but instead of being much smaller that the source file, it was about twice as big.  What could be in that file?

It seemed dangerous to just BLOAD a file that shouldn't exist, so I booted up a disk zap program and inspected the disk.  That Binary file contained the text of the assembly listing, starting with the .TF line.  Also, the file had no load address and length bytes.  The first four bytes were "A0 A0 A0 A0", or the ASCII codes for four spaces.  If I had tried to BLOAD the file, it would have loaded at $A0A0, which would have immediately clobbered DOS!

I was preparing a note to warn everybody not to use .TF within a .DU - .ED block, when Bob reminded me of how often we WANT an assembly listing on a Text file, to read into the S-C Word Processor and merge into an article.  Why didn't I find out what the Word Processor would make of this file?  Well, it read the file just fine, but discarded the first four bytes, since it expects load address and length bytes in a Binary file.  In most cases that is no problem, since the first line is the .TF <filename> directive, and will be discarded anyway.

Now we have a Binary file containing the text.  That's fine with the S-C Word Processor, but what about other programs, that might require Text files?  As it happens, the Macro Assembler creates a Target File as a Text file, then updates the Catalog to turn it into a Binary file.  All we have to do is patch the assembler to prevent that change in the file type.  That is only a 1-byte patch.

So, all it takes to send the assembly listing to a disk file is to begin the program with a .DU and a .TF, and end it with a .ED. If you want a real Text file, you only have to patch one byte in the assembler.

To make a long story short, here's how to create a Text file containing an assembly listing:

1) At the beginning of your program, put the lines:

```
0000       .DU
0001       .TF LISTING
```

and at the end put:

```
65535      .ED
```

2) If there is already another .TF directive in your program, insert a "*" at the beginning of the line, to make it into a comment.

3) Enter one of the following patches:

```
$1000 Versions: $29DF:0
$D000 Versions: $C083 C083 EAF9:0 N C080
```

4) Type ASM.

5) And restore the patched location:

```
$1000 Versions: $29DF:4
$D000 Versions: $C083 C083 EAF9:4 N C080
```

Now you can load LISTING into a word processor, delete the first and last lines, and do whatever you want with it!

I tried creating an EXEC file to do all those steps automatically, but ran into trouble. When the assembly ends the EXEC file loses control, and the Text file LISTING doesn't get closed. When I can solve that one I'll let you know.

```
==============================================================================
DOCUMENT :AAL-8307:Articles:Front.Page.txt
==============================================================================
```

$1.50

Volume 3 -- Issue 10 July, 1983

In This Issue...



Latest 65C02 Word

The 65C02 really does exist, and we now have a couple of them. As reported inside, the chips we received work in an Apple //e, but not in a ][+. Well that seems to be a problem with the NCR chips that I have. Don Lancaster reports that his GTE chips work just fine in all flavors of Apples. I'm swapping an NCR processor for one of his GTE's, and will have more details next month. For the time being, if you buy a 65C02, be sure to get a guarantee that it will work in your Apple.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

=============================================================================
DOCUMENT :AAL-8307:Articles:Mini.Assembler.txt
=============================================================================

6502 Mini-Assembler in Applesoft...........Bob Sander-Cederlof

The original Apple II came with a built-in mini-assembler. By typing "F666G" in the monitor, you entered a new realm. The prompt changed from "*" to "!"; errors not only earned a "beep" but also a printed "?"; and monitor commands were still available by typing an initial "$". I learned 6502 programming using this little tool, together with the handy "L" disassembly command. At the time, none of the other computer systems on the market came with either mini-assembler or "L" command.

A mini-assembler allows you to type in mnemonics rather than converting them "by hand". It also will translate branch addresses to the relative offsets needed in relative branch instructions. It will not retain the source code on a file, and will not handle labels. If you want to modify a program, you have to use patches or retype the whole thing. A full assembler will accept labels and comments, and will have some method for working with stored source programs. The S-C Macro Assembler, for example, includes a co-resident source program editor. The extra features a full assembler can include are limited only by the potential market. But mini-assemblers are free.

A long time ago MICRO published a 6502 mini-assembler written in Commodore or OSI BASIC. I started converting it, just for fun, into Applesoft. It wasn't long before I realized that my thought processes were totally incompatible with the author's programming techniques. So I essentially started over. Last month the partially finished listing appeared out of some long forgotten crack, so I dusted it off and finished the program.

It operates a lot like the old "F666G" mini-assembler by Steve Wozniak. (And, even though it is in Applesoft, it is almost as fast.) The initial display is the address "0300" at the left margin, and the cursor in column 20 of the top line on an otherwise empty screen. You can type RETURN to quit, a colon followed by a hex address to change the assembly address, or an instruction mnemonic to be assembled.

I could go into a long-winded explanation of how the program works, describing each subroutine. But you can probably read the listing easily enough, and there are identifying REM statements with each subroutine. The really interesting part to me is the structure of the opcode tables which are contained as strings in OP$, F$, and E$. These tables are set up in lines 2030 through 2050.

OP$ contains the opcodes names. OP$(1) holds the names of all the single byte opcodes. If the input line has no operand data after the opcode mnemonic, the program will search through OP$(1) and had better find your mnemonic. If not, it is "???" for you! Note that the opcode names are three characters each, packed into one long string.

Also note that ASL, LSR, ROL, and ROR appear in this string. These four opcodes can have an operand-less mode, as well as any of four modes with operands.

OP$(2) contains the mnemonics for the relative branches. OP$(3) holds "JMP" and "JSR". And OP$(4) holds all the rest, which I call the complex opcodes. These are the ones which can have a variety of addressing modes.

F$(1) through F$(4) correspond to OP$(1) through OP$(4). Each three digit group in one of the F$ strings is the opcode value (in decimal) for the corresponding mnemonic from OP$. F$(4) contains a base value, which will be augmented to obtain a specific value for the particular address mode chosen.

The complex opcodes can be classified in many different ways...I tried so many I lost count. I finally settled on the scheme shown in the two tables below:

```
             Imm Zp  Abs Z,X A,X Z,Y A,Y (X) ()Y
          +  08  04  0C  14  1C  --  18  00  10    Base
------------------------------------------------------------
ADC       0  69  65  6D  75  7D  --  79  61  71    61  097
AND       0  29  25  2D  35  3D  --  39  21  31    21  033
CMP       0  C9  C5  CD  D5  DD  --  D9  C1  D1    C1  193
EOR       0  49  45  4D  55  5D  --  59  41  51    41  065
LDA       0  A9  A5  AD  B5  BD  --  B9  A1  B1    A1  161
ORA       0  09  05  0D  15  1D  --  19  01  11    01  001
SBC       0  E9  E5  ED  F5  FD  --  F9  E1  F1    E1  225

STA       1  --  85  8D  95  9D  --  99  81  91    81  129

             Imm Zp  Abs Z,X A,X Z,Y A,Y (X) ()Y
          +  00  04  0C  14  1C  14  1C  --  --    Base
------------------------------------------------------------
ASL       2  --  06  0E  16  1E  --  --  --  --    02  002
LSR       2  --  46  4E  56  5E  --  --  --  --    42  066
ROL       2  --  26  2E  36  3E  --  --  --  --    22  034
ROR       2  --  66  6E  76  7E  --  --  --  --    62  098

BIT       3  --  24  2C  --  --  --  --  --  --    20  032

CPX       4  E0  E4  EC  --  --  --  --  --  --    E0  224
CPY       4  C0  C4  CC  --  --  --  --  --  --    C0  192

DEC       5  --  C6  CE  D6  DE  --  --  --  --    C2  194
INC       5  --  E6  EE  F6  FE  --  --  --  --    E2  226

LDX       6  A2  A6  AE  --  --  B6  BE  --  --    A2  162

LDY       7  A0  A4  AC  B4  BC  --  --  --  --    A0  160

STX       8  --  86  8E  --  --  96  --  --  --    82  130
```

```
STY    9  --  84  8C  94  --  --  --  --  --   80  128
```

The first column of numbers is the opcode class number. These numbers are stored in E$ (see line 2050). The next nine columns show the hex opcode values for each valid combination of opcode and address mode. The last two columns show the "base" value in both hex and decimal.

The top row of numbers (above the dashed lines) shows the augment needed to transform a "base" opcode value into the value for a specific address mode. I broke the data into two separate tables because the Imm and A,Y columns have one pair of values for class 0 and 1 opcodes and another for classes 2 through 9. The class number is used to select which address modes are legal for a given opcode, as well as in selecting the augment values.

If you have ever studied the listing of Wozniak's mini- assembler, you know that his approach was entirely different. If you look inside the S-C Macro Assembler you will find yet another approach. I suppose there are more approaches than existing assemblers. In our line of Cross Assemblers we use about five or six different techniques. The choice depends on the syntax of the operands and the bit structure of the opcodes, as well as whim.

I have also written a disassembler in Applesoft, and the beginnings of a simulator for 6502 code. Maybe they will see print in the near future. There is a lot to be learned from studying or even writing these kinds of programs, and they can even be useful.

==================================================================
DOCUMENT :AAL-8307:Articles:Miracle.txt
==================================================================

Answered Prayer............................Bob Sander-Cederlof

Last month's headlines bemoaned our burglary, with equipment worth over $11000 missing from our offices. And un-measurable amounts of software. And a damaged Spinwriter. And no insurance. We didn't even have all of the serial numbers recorded. The police indicated we should have no hope of recovering anything.

I know that God, who made the heavens and the earth and all that is in them, is sovereign. I said, "Thank you for this, too. And thank you that we still have enough left to continue business. And that nothing irreplaceable was taken."

And we tried to to put the pieces back together. We bought insurance, and recorded all the remaining serial numbers. We made backup copies of critical software to be kept at other locations. We engraved our driver's license numbers on our equipment. We even installed an alarm system.

After about two hours with a screwdriver and needlenose pliers the Spinwriter was back in working condition. Almost as good as new...just one crippled foot where it landed when dropped. NEC makes durable gear. I spent another 8 hours figuring out how to talk to it with a serial interface card (with no documentation), and writing the driver program. Once it all worked, we were able to print the mailing labels for last month's AAL.

The burglary occured sometime after 8:30 pm, Wednesday night, May 25th. The next Wednesday night, after choir practice, we took some time to pray. Among other concerns, we prayed about the burglary. I suggested, "Let's pray that the burglars be caught and the things they took be recovered. It can't hurt to ask!" So we did.

The next day the police received a tip from an informer. They went to investigate, just in time to catch two 18-year-olds carrying computers from apartment to car. One of them was a well-known burglar, with at least six-year record. The equipment matched the description I had given them. Friday morning the investigator called: "We have some of your computers. You can come and pick them up at noon." Although a little dirty, none of the equipment or software was damaged. Two thirds of all that had been stolen was recovered! "A miracle", the police said. "Amen."

The following Monday the police called again. "We have some more." The third computer system, a brand new Apple //e with extended 80-column card, two disk drives, monitor, and Epson printer had been sold to a technically-minded friend (of the burglars) for only $100. Responding to the alternatives offered by our excellent police

("Return the computer, or go to jail"), the friend brought in all he had bought.  Almost everything was back in our office!

Thursday, June 16th, I was called a third time.  "We have another disk drive."  They also had another FlipFile with about 15 more diskettes. Now all that is missing is a TI Programmer calculator and an old Panasonic Cassette Recorder.

Yes, God is sovereign, and also He cares about us as individuals.  He allowed our things to be taken, but not everything.  He gave us faith to ask for them to be returned.  And He caused them to be returned. "Trust in the Lord with all your heart, and do not lean on your own understanding.  In all your ways acknowledge Him, and He shall direct your paths."  [Proverbs 3-5,6]

=======================================================================
DOCUMENT :AAL-8307:Articles:MonAsciiDisplay.txt
=======================================================================

Revised Monitor Patch for ASCII Display....Bob Sander-Cederlof

Peter Bartlett gave us a nice patch to the Apple Monitor to add ASCII display to the memory dump command. It was published in the Dec 1981 issue of Apple Assembly Line, pages 18-20. You may remember that Peter's patch over-wrote the cassette tape code. Last summer I received two suggested modifications to Peter's code, and at last I pass them on to you.

Bruce Field, from Rockville, Maryland: "I finally got around to building my own EPROM burner the other day, and one of the first things I did was to modify my F8 ROM to include an ASCII listing with the hex dump. I used the routine originally submitted by Peter Bartlett. I found a minor problem with this code.

"The problem is that I have the modified ROM on an Integer BASIC card, and an unmodified ROM on the mother board. If I am in the modified ROM and want to soft-switch back to the mother board, typing 'C081' should do it. But with Peter's patch location C081 is accessed inside the patch itself, so the card switches off with PC pointing inside the cassette tape code!

"My solution is to leave the loading of the memory location in its original position. This makes the patch slightly longer, but it still fits inside the cassette tape space. Also, since I detest flashing characters, I filter these out. I force control characters to inverse mode, and all others to normal video."

<Bruce's code here>

Brooke Boering, from Schaumburg, Illinois: "Here is a slightly modified version of Peter Bartlett's monitor patch. I modify control characters to display as an underline character, and lower case codes to inverse video. Other characters display in normal video."

<Brooke's code here>

After assembling Bruce's version above, using the S-C Macro Assembler resident in my language card, I installed the patch by typing:

```
     $C083 C083     (write enable RAM card)
     $FCC9<CC9.CE3M (move the patch into the
                    cassette space)
     $FDBE:C9 FC    (install patch address in JSR)
```

And it worked! To install Brooke's code I had to move a few more bytes:

```
     $FCC9<CC9.CE9M
```

If you have an Apple //e, or a lower case display adapter in an older Apple, you will not want to display lower case characters in inverse mode.  Everyone seems to have their own preferences about how to display the 256 possible hex values on Apple's screen.  Choose your own favorite

```
===========================================================================
DOCUMENT :AAL-8307:Articles:My.Ad.txt
===========================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50

S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00

S-C Word Processor................................................$50.00
  As is, with fully commented source code.  Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)................... $79.00
Source Code for FLASH! Runtime Package............................$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).......$49.00

The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00
Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00

Blank Diskettes (with hub rings).................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Ashby Shift-Key Mod...............................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
     Only Revision level 7 or later Apples.

STB-80 80-column Display Board (STB Systems)...........($249.00)  $225.00
STB-128 128K RAM Card (STB Systems)....................($399.00)  $350.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
```

```
  "THE Book of Apple Software 1983 (with supplement)...($24.90)  $18.00
  "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
  "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
  "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
  "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
  "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
  "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
  "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
  "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
  "What's Where in the Apple", Second Edition..........($24.95)  $23.00
  "What's Where Guide" (updates first edition)..........($9.95)   $9.00
  "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
  "6502 Subroutines", Leventhal........................($17.95)  $17.00

Add $1.50 per book for US postage.  Foreign orders add postage needed.

      *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
      ***                  (214) 324-2050                  ***
      *** We take Master Charge, VISA and American Express ***
```

=============================================================================
DOCUMENT :AAL-8307:Articles:New.DOS3.3.txt
=============================================================================

The new 1983 edition of DOS 3.3...........Bob Sander-Cederlof

Co-incident with the release of the //e, Apple started shipping a slightly modified version of DOS 3.3. Three changes are evident: the sample programs have been moved to a separate diskette; a few instructions to kill 80-column display during a boot were added; and yet another patch to the APPEND command.

I booted an old DOS 3.3, and then used monitor move to make a copy in memory running from 5D00-7FFF of the DOS image. Then I booted the new DOS, which loaded into 9D00-BFFF. Using the monitor "V" command, I located all of the changes. It was a little tricky skipping over the variables and buffers, but with the aid of a well-worn copy of "Beneath Apple DOS" I managed. Here are all the changes I found:

```
 Old DOS 3.3                 New DOS 3.3
----------------------     ----------------------
A6BB:EA       NOP          A6BB:20 69 BA JSR $BA69
A6BC:EA       NOP
A6BD:EA       NOP
A6BE:A2 00    LDX #0       A6BE:unchanged
A6C0:8E C3 B5 STX $B5C3
A6C3:60       RTS
```

The code above is jumped to from one of the older APPEND patches at $B6A8. It used to be JMP $A6BC, and has been changed to JMP $A6BB to pick up the new JSR there.

The latter part of the file position calculator has been re-written to assure carry is clear before adding record size to previous position.

```
 Old DOS 3.3                 New DOS 3.3
----------------------     ----------------------
B33E:AD BF B5 LDA $B5BF    B33E:18       CLC
B341:8D EC B5 STA $B5EC    B33F:AD BF B5 LDA $B5BF
B344:6D E6 B5 ADC $B5E6    B342:8D EC B5 STA $B5EC
B347:8D E6 B5 STA $B5E6    B345:6D E6 B5 ADC $B5E6
B34A:AD C0 B5 LDA $B5C0    B348:8D E6 B5 STA $B5E6
B34D:8D ED B5 STA $B5ED    B34B:AD C0 B5 LDA $B5C0
B350:6D E4 B5 ADC $B5E4    B34E:8D ED B5 STA $B5ED
B353:8D E4 B5 STA $B5E4    B351:6D E4 B5 ADC $B5E4
B356:A9 00    LDA #0       B354:8D E4 B5 STA $B5E4
B358:6D E5 B5 ADC $B5E6    BE57:90 03    BCC $B35C
B35B:8D E5 B5 STA $B5E5    BE59:EE E5 B5 INC $B5E5
B35E:60       RTS          BE5C:60       RTS
                           BE5D:00 00    filler
```

Note that there was room for adding the CLC at the top, because of the in-efficiency of the original code.

Code executed at the end of a boot has been modified to clear 80-column mode in case you are booting in an Apple //e.

```
 Old DOS 3.3                New DOS 3.3
----------------------     ----------------------
BFD6:4C 44 B7 JMP $B744    BFD6:20 76 BA JSR $BA76
                           BFD9:4C 44 B7 JMP $B744
```

Three patches were stuffed into the hole at $BA69.

```
       Called from $A6BB:

BA69:AE 5F AA LDX $AA5F   If last command was
BA6C:E0 1C    CPX #$1C         APPEND, clear flag
BA6E:FO 05    BEQ $BA75        Not APPEND
BA70:A2 00    LDX #0           Yes, APPEND
BA72:8E 5D B6 STX $B65D        Clear APPEND flag
BA75:60       RTS

       Called from $BFD6:

BA76:A9 FF    LDA #$FF
BA78:8D FB 04 STA $04FB        MODE = $FF
BA7B:8D 0C C0 STA $C00C        80-column display OFF
BA7E:8D 0E C0 STA $C00E        Alternate Char Set OFF
BA81:4C 2F FB JMP $FB2F        Exit via monitor INIT

       Called from $B683:

BA84:AD BD B5 LDA $B5BD        Previous file position
BA87:8D E6 B5 STA $B5E6        LSB of fle position
BA8A:8D EA B5 STA $B5EA        Record #
BA8D:BA       TSX
BA8E:8E 9B B3 STX $B39B        Save stack position
BA91:4C 7F B3 JMP $B37F        Leave File Manager
```

Note that this last patch jumps into the file manager exit routine even though the file manager had not been entered.  The purpose is to save a copy of the file manager workarea in the file buffer after patching the file position low-order byte.  Seems to me that jumping directly to $AE7E, without the two lines saving the stack pointer above, would avoid the very dangerous step of jumping into the middle of a subroutine.  But in any case, as Tom Weishaar points out, the code is wrong in that it does not recover the higher bytes of the file position.  Will APPEND ever really be fixed?

A few months back I published a patch for faster LOAD, etc) in these pages which used the space from $BA69 through $BA95.  I suggest you use the older version of DOS 3.3 for the time being.  But eventually you may be forced to find another home for the fast LOAD patch.

```
=============================================================================
DOCUMENT :AAL-8307:Articles:OBriens.BGE.BLT.txt
=============================================================================
```

More opcodes for the S-C MACRO ASSEMBLER.........R.F. O'Brien

While using the assembler I felt that it was a pity that the BGE and BLT instructions had not been incorporated especially as it would only have meant an extra 6 bytes of code. This minimal extra overhead is because of the way opcodes are handled in the assembler.

Take for example the BRANCH opcode table, which resides in locations $EF5B-EF93. ($2E29-2E47 for RAM version.)  [ These addresses are for version 1.0 ]

This table is preceded by a 2-byte descriptor and ends as one would expect with a 00 as end-of-table marker. The descriptor in this case is 0302, i.e. 3-byte entries having a 2-byte name. The table holds the standard 8 6502-opcodes and the 10 Sweet-16 opcodes, interestingly the B of each instruction name has been dropped, saving 18 bytes.

The entries in the table consist of the last 2 letters of the instruction name followed by the hex code. In the case of 2-letter names the entry consists of the second letter plus a space ($20) followed by its hex code.

I decided that I could dispense with the SW-16 codes BM1 and BNM1 without suffering too much if I wanted to write Sweet-16 code in my programs. However, I found that to incorporate the new codes they would have to be placed between the 6502 codes and the SW-16 codes in the table.

It was just a matter of pushing the code for BR to BNZ up in RAM 6 bytes and slotting in the code for BLT and BGE.

To install the code for the two new opcodes just enter the following at any convenient location e.g.$4000 and BSAVE as BLT/BGE.CODE,A$4000,L$1F

```
     :$4000:47 45 B0 4C 54 90     ("G E B0  L T 90")
     :$:52 20 01 4E 43 02 43 20 03 50 20 04
     :$:4D 20 05 5A 20 06 4E 5A 07 53 20 0C 00
```

To install in LC-Version just enter:

```
     :$C083 C083         write enable card.
     :BLOAD BLT/BGE.CODE,A$EF75
     :$C080              write protect card.
```

To install in RAM-Version just enter:

```
:BLOAD BLT/BGE.CODE,A$2E29
```

If you never use Sweet-16 you only need to use the first 6 bytes of the above code. However, this will wipe out the BR and BNC codes in the table.

Now you can use either BCC or its synonym BLT (Branch if Less Than) and BCS or BGE (Branch if Greater than or Equal to) in your programs and have them assembled correctly without using macro definitions.

The load address for the patch file for version 1.1 will vary depending on which of the 8 versions you are patching:

| | 40-col | //e | Videx | STB-80 |
|---|---|---|---|---|
| Motherboard | $31A9 | 318D | 3274 | 329D |
| RAM Card | $F2C3 | F2A7 | F397 | F3C0 |

==============================================================
DOCUMENT :AAL-8307:Articles:Opcodes.txt
==============================================================

| | | Imm | Zp | Abs | Z,X | A,X | Z,Y | A,Y | (X) | ()Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | + | 08 | 04 | 0C | 14 | 1C | -- | 18 | 00 | 10 | Base | |
| ADC | 0 | 69 | 65 | 6D | 75 | 7D | -- | 79 | 61 | 71 | 61 | 097 |
| AND | 0 | 29 | 25 | 2D | 35 | 3D | -- | 39 | 21 | 31 | 21 | 033 |
| CMP | 0 | C9 | C5 | CD | D5 | DD | -- | D9 | C1 | D1 | C1 | 193 |
| EOR | 0 | 49 | 45 | 4D | 55 | 5D | -- | 59 | 41 | 51 | 41 | 065 |
| LDA | 0 | A9 | A5 | AD | B5 | BD | -- | B9 | A1 | B1 | A1 | 161 |
| ORA | 0 | 09 | 05 | 0D | 15 | 1D | -- | 19 | 01 | 11 | 01 | 001 |
| SBC | 0 | E9 | E5 | ED | F5 | FD | -- | F9 | E1 | F1 | E1 | 225 |
| STA | 1 | -- | 85 | 8D | 95 | 9D | -- | 99 | 81 | 91 | 81 | 129 |

| | | Imm | Zp | Abs | Z,X | A,X | Z,Y | A,Y | (X) | ()Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | + | 00 | 04 | 0C | 14 | 1C | 14 | 1C | -- | -- | Base | |
| ASL | 2 | -- | 06 | 0E | 16 | 1E | -- | -- | -- | -- | 02 | 002 |
| LSR | 2 | -- | 46 | 4E | 56 | 5E | -- | -- | -- | -- | 42 | 066 |
| ROL | 2 | -- | 26 | 2E | 36 | 3E | -- | -- | -- | -- | 22 | 034 |
| ROR | 2 | -- | 66 | 6E | 76 | 7E | -- | -- | -- | -- | 62 | 098 |
| BIT | 3 | -- | 24 | 2C | -- | -- | -- | -- | -- | -- | 20 | 032 |
| CPX | 4 | E0 | E4 | EC | -- | -- | -- | -- | -- | -- | E0 | 224 |
| CPY | 4 | C0 | C4 | CC | -- | -- | -- | -- | -- | -- | C0 | 192 |
| DEC | 5 | -- | C6 | CE | D6 | DE | -- | -- | -- | -- | C2 | 194 |
| INC | 5 | -- | E6 | EE | F6 | FE | -- | -- | -- | -- | E2 | 226 |
| LDX | 6 | A2 | A6 | AE | -- | -- | B6 | BE | -- | -- | A2 | 162 |
| LDY | 7 | A0 | A4 | AC | B4 | BC | -- | -- | -- | -- | A0 | 160 |
| STX | 8 | -- | 86 | 8E | -- | -- | 96 | -- | -- | -- | 82 | 130 |
| STY | 9 | -- | 84 | 8C | 94 | -- | -- | -- | -- | -- | 80 | 128 |

==============================================================================
DOCUMENT :AAL-8307:Articles:Othello.txt
==============================================================================

Bobby Deen's Latest Stuff

Bobby Deen is a name you may remember seeing in these pages in several past issues. He will be entering Texas A & M University this fall. Bobby programmed most of the cross assembler modules for the S-C Macro Assembler, some parts of the S-C Word Processor, about half of the yet-to-be-released 18-digit commercial math package (S-C DP18), and parts of the CPR Training System we did for the American Heart Association. A man of many interests, Bobby also has produced some excellent music disks for the ALF music synthesizers (or any Alf compatibles, such as Applied Engineering), and now a fantastic "Othello" game.

His six-voice renditions of the William Tell Overture by Rossini and Tchaikovsky's Nutcracker Suite are outstanding, and the price is only $10. If you have a synthesizer, you ought to have Bobby's music.

Bobby's Othello program is available now for an introductory price of only $20. Of course he wrote it in assembly language, so it is FAST, and has excellent hi-res graphics. You select among six skill levels; Apple can suggest your next move; you can swap sides with the computer; you can modify the board in mid-game (cheat?); you can pit the computer against itself. Whenever two or more moves tie for the machine's best next move, Apple randomizes its choice. This way you never play the same identical game twice.

Order either of these disks from S-C Software.

================================================================
DOCUMENT :AAL-8307:Articles:Short.Subjects.txt
================================================================

Osborne Raises Book Prices

Osborne/McGraw-Hill has raised the prices of most of their books. In particular, all of their books which we carry have new higher prices. They are still bargains when you consider how good they are, and how packed with information:

| Title | Was | Is Now | Our Price |
|---|---|---|---|
| 6502 Assembly Lang Prog | $16.99 | $18.95 | $18.00 |
| 6502 Subroutines | $15.95 | $17.95 | $17.00 |
| Z-80 Assembly Lang Prog | $16.99 | $18.95 | $18.00 |
| Z-80 Subroutines | $15.95 | $17.95 | $17.00 |

Eighty-Column SHOW Command.....................Robert Bragner
Istanbul, Turkey

I make frequent use of the SHOW command for text files (see AAL July 1982), and I wanted to see it in 80-column glory on my shiny new //e. If you've tried it, you will have noticed that the command places a character in every other column on the 80-column screen, so you still only see 40-columns of data per line!

The reason is that the SHOW command code calls COUT1 at $FDF0 for its character output, and COUT1 knows nothing about 80-column output. By calling COUT ($FDED) instead, the text file output will be routed to whatever your current output device happens to be (including printer, 80-column display, etc.).

If you use the Applesoft on page 27 of that issue to load SHOW, all you need to do is change the ninth item on line 100 from 240 to 237.

Here is the modified POKEr, complete with the additions made by Bil Morgan in the June 83 issue, to save you hunting through all those back issues:

```
100 DATA 21,42319,32,163,162,169,141,32,237,253,32,142,
         174,240,5,32,140,166,208,243,76,252,162
110 DATA 23,44686,173,0,192,16,17,141,16,192,201,141,
         240,10,173,0,192,16,251,141,16,192,201,141,96
115 DATA 13,44709,224,0,240,4,162,2,208,2,162,4,76,3,171
116 DATA 3,43773,76,165,174
120 DATA 4,43140,83,72,79,215
130 DATA 2,43273,32,48
140 DATA 0
150 READ N : IF N THEN READ A : FOR I = 1 TO N
    : READ D : POKE A+I-1,D : NEXT : GO TO 150
```

```
==============================================================================
DOCUMENT :AAL-8307:Articles:Show.Poker.txt
==============================================================================

100 DATA
21,42319,32,163,162,169,141,32,237,253,32,142,174,240,5,32,140,166,208
,243,76,252,162
110 DATA
23,44686,173,0,192,16,17,141,16,192,201,141,240,10,173,0,192,16,251,14
1,16,192,201,141,96
115 DATA 13,44709,224,0,240,4,162,2,208,2,162,4,76,3,171
116 DATA 3,43773,76,165,174
120 DATA 4,43140,83,72,79,215
130 DATA 2,43273,32,48
140 DATA 0
150 READ N : IF N THEN READ A : FOR I = 1 TO N : READ D : POKE A+I-1,D
: NEXT : GO TO 150
```

```
==============================================================================
DOCUMENT :AAL-8307:Articles:V3N10.65C02.txt
==============================================================================
```

65C02 Department....................................Bill Morgan

I am holding a brand new NCR65C02A. Now I finally believe that there is such a creature as a 65C02! NCR's version of this processor seems to be the same as GTE's. That is, it has all of the enhancements described in the December '82 issue of AAL, except for the single bit set, reset and branch instructions.

We have tested the chip in the computers here, and there's good news and bad news. As Don Lancaster reported last month, the new chip works perfectly in an Apple //e. You just swap processors and start using new opcodes. However, 65C02 chips do not work in the Apple ][ or Apple ][+.

I am told that the problem lies in the execution of instructions like ASL or INC, which read memory, modify the contents, and write the result back to the same address. The 6502 processor does one read and two writes during such an instruction, which is really incorrect. In the 65C02 this has been changed to the proper combination of two reads and one write.

Unfortunately, the Apple ][s rely on the timing of the read-write-write cycle, and the read-read-write cycle is just different enough to cause the system to fail. Hopefully some of the hardware specialists can come up with a modification to the older Apples to allow the use of the enhanced processors.

Let's talk about programming the 65C02. With the new chip in a //e, Bob and I started tearing into the S-C Word Processor. We just went through the code, looking for places to substitute a new instruction for several old ones. Come to find out, the most useful change is the true Indirect addressing mode, in place of Indexed Indirect. That means replacing

```
     STY YSAVE
     LDY #0
     LDA (POINTER),Y
     LDY YSAVE          with     LDA (POINTER).
```

That's replacing 8 bytes with 2 bytes. BRA (BRanch Always) and STZ (STore Zero) also came in very handy.

All things considered, Bob has decided to wait for the Rockwell version of the 65C02, because he really wants those single bit set, reset, and branch instructions. At last word Rockwell was expecting to start shipping in August, so it will be at least that long before we have any. NCR's chip costs about $10. The Rockwell chip may cost a little more, if and when. We have noticed ads offering 65C02's for $35, which just goes to show how expensive advertising can be.

==================================================================
DOCUMENT :AAL-8307:Articles:WeishaarIIeDOS.txt
==================================================================

Explanation of the new DOS Append Bug..............Tom Weishaar
Overland Park, Kansas

[ Tom is author of ProntoDOS, published by Beagle Bros, an excellent speed-enhanced DOS which happens to be compatible with nearly everything. He also writes the monthly DOStalk column in Softalk Magazine. ]

I was behind on my reading when I wrote, in the April Softalk DOStalk, about the changes Apple made to DOS 3.3 in the new //e release. At that time I noticed the routine used to calculate random access file position at $B331 had been modified, but the change looked insignificant to me.

It turns out this change was supposed to fix another bug in the Append command! The change was very well documented by Art Schumer in the August 1982 Call APPLE, page 57.

In pre-//e DOS, Append called on this random-access file position calculator to reset the position-in-file pointer. As you know, Append simply looks through a file byte-by-byte until it finds the end, which can be indicated either by a zero byte or by a lack of additional sectors.

When Append finds a zero byte in the file, it knows it has reached the end, but by then the position-in-file pointer is one byte beyond the zero and has to be backed up. Somebody once thought a call to the random-access file position calculator would be a good way to do this.

But on sequential files (the only kind you can append to) the File Manager uses a record length of one. Thus files longer than 32767 bytes come to this routine with more than 32767 "records", which is beyond what DOS normally allows. The calculation fails.

Schumer's patch gets it to calculate correctly right up to 65535. At that point it stops working for good. Apple tried to get around this in //e-DOS by throwing out Append's reliance on the random-access calculator. Instead they go back in and change the position-in-file pointer directly, then trick the File Manager into re-saving his workarea.

Problem: they only decrement the low byte of the position-in- file pointer. If the file-ending zero comes in the last byte of a sector, the high byte will have been advanced to point at the next sector. Since they don't decrement it, the position- in-file pointer is 256 bytes beyond where it should be.
Uh Oh...!

I've been trying to get folks at Apple to recognize the problem, but Append doesn't appear to be one of their priorities.  If they don't do something soon I'll publish a patch in Softalk.  I'd do it now, but I fear treading where so many have failed before me.

==============================================================================
DOCUMENT :AAL-8307:DOS3.3:MINI.ASSEMBLER.txt
==============================================================================

( DTC removed -- lots of garbage characters )

```
==========================================================================
DOCUMENT :AAL-8307:DOS3.3:S.FastTextRBSC.txt
==========================================================================

 1000  *SAVE S.FAST TEXT (RBS-C)
 1010  *--------------------------------
 1020  *      PAUL SCHLYTER'S TEXT FILE SPEED-UP
 1030  *      AS MODIFIED BY BOB SANDER-CEDERLOF
 1040  *                      JUNE 8, 1983
 1050  *--------------------------------
 1060  PNTR   .EQ $42,43
 1070  *--------------------------------
 1080  COPY.BUFFER.TO.WORKAREA .EQ $AE6A
 1090  SAVE.WORKAREA           .EQ $AE81
 1100  POINT.TO.WORKAREA       .EQ $AF08
 1110  SETUP.PNTR              .EQ $AF12
 1120  FM.OPCODE               .EQ $B5BB
 1130  PNTR.SAVE               .EQ $B5CF,B5D0
 1140  *--------------------------------
 1150  PATCH.AREA              .EQ $B6B3
 1160  PATCH.LINK1             .EQ $AB0B
 1170  PATCH.LINK2             .EQ $B38F
 1180  *--------------------------------
 1190         .OR $300
 1200  *--------------------------------
 1210  INSTALL.PATCHES
 1220         LDX #PATCH.SIZE-1
 1230  .1     LDA PATCH.CODE,X
 1240         STA PATCH.AREA,X
 1250         DEX
 1260         BPL .1
 1270         LDA #PATCH1
 1280         STA PATCH.LINK1
 1290         LDA /PATCH1
 1300         STA PATCH.LINK1+1
 1310         LDA #PATCH2
 1320         STA PATCH.LINK2
 1330         LDA /PATCH2
 1340         STA PATCH.LINK2+1
 1342         LDA #20
 1344         STA $A1B1
 1350         RTS
 1360  *--------------------------------
 1370  PATCH.CODE
 1380         .OR $B6B3
 1390         .TA PATCH.CODE
 1400  PATCH1 JSR CHECK.OPCODE
 1410         BCC .1       NOT READ/WRITE
 1420         CMP PNTR.SAVE
 1430         BNE .1       NO
 1440         CPX PNTR.SAVE+1
 1450         BEQ PATCH3   YES, RETURN NOW
 1460  .1     BIT FLAG
```

```
1470        BPL .2
1480        LDX #PNTR.SAVE-$B5C7
1490        JSR SETUP.PNTR
1500        JSR PATCH4   CLEAR FLAG, SAVE WORKAREA
1510 .2     JMP COPY.BUFFER.TO.WORKAREA
1520 *---------------------------------
1530 PATCH2 JSR CHECK.OPCODE
1540        BCC PATCH4   NOT READ OR WRITE
1550        ROR FLAG     SET SIGN BIT
1560        STA PNTR.SAVE
1570        STX PNTR.SAVE+1
1580 PATCH3 RTS
1590 *---------------------------------
1600 PATCH4 LSR FLAG     CLEAR SIGN BIT
1610        JMP SAVE.WORKAREA
1620 *---------------------------------
1630 CHECK.OPCODE
1640        JSR POINT.TO.WORKAREA
1650        LDA FM.OPCODE
1660        SEC
1670        EOR #3       READ?
1680        BEQ .1       YES
1690        EOR #7       WRITE?
1700        BEQ .1
1710        CLC
1720 .1     LDA PNTR
1730        LDX PNTR+1
1740        RTS
1750 *---------------------------------
1760 FLAG   .HS 00       MUST START WITH FLAG=0
1770 *---------------------------------
1780 PATCH.SIZE .EQ *-PATCH1
```

==============================================================================
DOCUMENT :AAL-8307:DOS3.3:S.FTSchlyter.txt
==============================================================================

```
*SAVE S.FAST TEXT (SCHLYTER)
*---------------------------------
*  SPEEDUP OF DOS TEXT FILE READ/WRITE
*  BY PAUL SCHLYTER, SWEDEN, MAY 1983.
*---------------------------------
PNTR   .EQ $42,43   DOS VARIABLE
*---------------------------------
GET.WRKAREA.FROM.BUFFER .EQ $AE6A
SAVE.WRKAREA.TO.BUFFER  .EQ $AE7E
SAVE.OLD.WRKAREA        .EQ $AE7E+3
POINT.TO.WRKAREA.BUFFER .EQ $AF08
FM.OPCODE               .EQ $B5BB
*---------------------------------
       .OR $300
       JMP INSTALL
*---------------------------------
*  PATCH EXECUTED UPON ENTRY TO FILE MANAGER:
*      1. IF READ/WRITE AND CORRECT WORK AREA, RETURN
*      2. IF WRONG WORK AREA, SAVE OLD WORK AREA
*      3. LOAD NEW WORK AREA.
*---------------------------------
PATCH1 LDA FM.OPCODE     IF NOT READ OR WRITE,
       CMP #3                DO AS USUAL
       BCC .1            ...LESS THAN READ/WRITE
       CMP #5
       BCS .1            ...HIGHER THAN READ/WRITE
       JSR POINT.TO.WRKAREA.BUFFER
       LDA PNTR     CHECK TO SEE IF CORRECT WORKAREA
       CMP PNTR.S        ALREAD LOADED
       BNE .1            NO
       LDA PNTR+1
       CMP PNTR.S+1
       BNE .1            NO
       RTS               YES, NOTHING ELSE TO DO,
*                        SO EXIT NOW AND SAVE 800 CYCLES!
*
*      OPCODE NOT READ OR WRITE, OR WRONG WORK AREA.
.1     BIT FLG      NEED TO PUT BACK THIS WORK AREA?
       BPL .2       NO, JUST GET NEW ONE
       CLC          YES, CLEAR FLAG
       ROR FLG
       LDA PNTR.S
       STA PNTR
       LDA PNTR.S+1
       STA PNTR+1
       JSR SAVE.OLD.WRKAREA
.2     JMP GET.WRKAREA.FROM.BUFFER
*---------------------------------
*  PATCH EXECUTED WHEN FILE MANAGER IS FINISHED:
```

```
1490  *      1. IF READ/WRITE, SET FLAG AND SAVE PNTR
1500  *      2. IF NOT R/W, CLEAR FLAG AND SAVE WORK AREA
1510  *--------------------------------
1520  PATCH2 LDA FM.OPCODE     R/W?
1530         CMP #3
1540         BCC .1           NO
1550         CMP #5
1560         BCS .1           NO
1570         SEC              YES, SET FLAG
1580         ROR FLG
1590         JSR POINT.TO.WRKAREA.BUFFER
1600         LDA PNTR         AND SAVE POINTER
1610         STA PNTR.S
1620         LDA PNTR+1
1630         STA PNTR.S+1
1640         RTS              SAVE ANOTHER 800 CYCLES
1650  .1     CLC              CLEAR FLAG
1660         ROR FLG
1670         JMP SAVE.WRKAREA.TO.BUFFER
1680  *
1690  PNTR.S .HS 0000
1700  FLG    .HS 00
1710  *--------------------------------
1720  *  TO INSTALL, PATCH DOS LIKE THIS:
1730  *      .OR $AB0A
1740  *      JSR PATCH1
1750  *
1760  *      .OR $B38E
1770  *      JSR PATCH2
1780  *
1790  *  HERE IS ONE WAY TO DO IT:
1800  *--------------------------------
1810  INSTALL
1820         LDA #$20     JSR OPCODE
1830         STA $AB0A
1840         STA $B38E
1850         LDA #PATCH1
1860         STA $AB0B
1870         LDA /PATCH1
1880         STA $AB0C
1890         LDA #PATCH2
1900         STA $B38F
1910         LDA /PATCH2
1920         STA $B390
1930         RTS
1940  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8307:DOS3.3:S.MAD.BOERING.txt
==============================================================================

```
=============================================================================
DOCUMENT :AAL-8307:DOS3.3:S.MAD.FIELD.txt
=============================================================================

 1000  *SAVE S.MON ASCII DISPLAY (FIELD)
 1010  *---------------------------------
 1020  *   PATCHES TO ADD ASCII DUMP TO APPLE MONITOR
 1030  *   ORIGINAL BY PETER BARTLETT
 1040  *   MODIFIED BY BRUCE FIELD
 1050  *---------------------------------
 1060  A1L        .EQ $3C
 1070  COUT       .EQ $FDED
 1080  PRBYTE     .EQ $FDDA
 1090  *---------------------------------
 1100         .OR $FDBD
 1110         .TA $0DBD
 1120         JSR PATCH    CALL MY PATCH CODE
 1130  *---------------------------------
 1140         .OR $FCC9
 1150         .TA $0CC9
 1160  PATCH  PHA          SAVE BYTE
 1170         LDA A1L      LOW BYTE OF DUMP ADDRESS
 1180         AND #7       MASK LINE POSITION
 1190         CLC
 1200         ADC #31      COMPUTE HORIZONTAL OFFSET
 1210         TAY
 1220         PLA          GET BYTE FROM STACK
 1230         PHA          KEEP COPY ON STACK
 1240         ORA #$80     FORCE NORMAL VIDEO
 1250         CMP #$A0     MAKE CONTROL-CHARS INVERSE
 1260         BCS .1       ...NOT CTRL
 1270         AND #$7F     ...CTRL
 1280  .1     STA ($28),Y  STORE IT ON THE SCREEN
 1290         LDY #0       RESTORE Y
 1300         PLA          RECOVER BYTE AGAIN
 1310         JMP PRBYTE
```

```
================================================================================
DOCUMENT :AAL-8307:DOS3.3:TxtFileSpeedup.txt
================================================================================

d≤ TEXT FILE SPEEDUP PATCH-náN:≠N–0ƒÄNxáA:ÅI–
0¡N…1:áX:πA»I,X:Ç:´110÷»É74,46771,32,210,182,144,10,205,207,181,208,5,
236,208,181,240,51,44,252,182,16,8,162,8,32,18,175,32,246,182,76,106,1
74,32,8,175,173a
     “É187,181,56,73,3,240,5,73,7,240,1,24,165,66,166,67,96,32,210,182
,144,10,110,252,182,141,207,181,142,208,181,96,78,252,182,76,129,174,0
v    ‹É2,43787,179,182ã   ÊÉ2,45967,231,182ú   É 1,41393,20£ ˙É0
```

=============================================================================
DOCUMENT :AAL-8308:Articles:Bit.and.Pieces.txt
=============================================================================

Grappler Interfaces

There should be a leaflet included with this issue describing the Grappler printer interfaces. We now have three of them "in the family" here, and have been very pleased with their performance. Check the brochure for features, the ad on page three for our prices, and let us hear from you.

WICO Track Ball

Several of you have inquired about or ordered the WICO Track Ball that I reviewed a couple of months ago, so we've decided to carry them regularly. WICO has since raised their price from $79.95 to $89.95, so we're going from $75 to $80.

Diskettes

There's getting to be a lot more competition in the diskette business, so prices are falling. After seeing so many ads at such attractive prices, Bob called Verbatim and told them that we had to have a better price, or we would have to change brands. That paid off, so we can now offer the same high- quality Verbatim Datalife diskettes at $45.00 for a package of 20. That's $2.25 each for the best diskettes we've found.

Whatever You Want

If you're shopping for a new peripheral, accessory, or program, give us a call and ask for a quote. We can get nearly anything you might want, and we'd love the chance to serve you.

Mailing AAL

Let's review how AAL is mailed, when you should expect to receive it, and what to do about it when you don't. Most of you get your newletter by Bulk Mail, which is a little erratic. You should receive your issue around the third week of each month, but don't start worrying until the end of the month. If you haven't received an issue by the end of the month, call or write and we'll send a replacement. Those of you who have First Class Mail subscriptions should receive your issue around the tenth of the month, and certainly before the twentieth.

The Post Office does not forward Bulk Mail, so make certain to tell us if you move.

The number in the upper right corner of your mailing label is the expiration date of your subscription. If that number is 8308, you're holding your last issue and better renew now. We send out postcards when your subscription is about to expire, and when it has expired. All you have to do is send us a check, or phone with a charge card number, and we'll keep your AAL coming.

65C02

People who have started reading AAL since last December have asked what is all this 65C02 business, anyway? Well the 65C02 is a new CMOS version of the 6502 microprocessor. (CMOS stands for Complementary Metal Oxide Semiconductor. That's a different way of making chips. CMOS circuits are noted for extremely low power consumption and extremely high sensitivity to static electricity.) To us Apple owners, the important thing is that the designers of the new chip corrected the bugs in the 6502 and added several new instructions and addressing modes.

The new instructions include PHX, PLX, PHY, and PLY (push and pull the X and Y registers from the stack), BRA (branch always), STZ (store zero), TSB and TRB (test and set or reset bits), and SMB, RMB, BBR and BBS (set, reset and branch on single bits). The main new addressing mode is true indirect without indexing, LDA ($12). This mode is now available for ORA, AND, EOR, ADC, STA, LDA, CMP, and SBC. There are also new modes for the BIT and JMP instructions. INC and DEC can now work on the A register.

There are some problems, though. Rockwell, GTE, NCR, and Synertek (maybe) are manufacturing 65C02 processors, but they are not all the same. The SMB, RMB, BBS, and BBR instructions are only available in the Rockwell chip. The NCR chip works in the Apple //e, but not in older Apples. The GTE processor does work in all Apples (this is being written on an Apple ][+ with a GTE 65C02). I haven't yet received a sample of the Rockwell processor, so I don't personally know if it works in older Apples. Some people say yes, others no.

That's a summary of what we know so far. The confusion is beginning to clear up, but there are still questions about what will or won't work in which Apples, and why. Stay tuned...

```
==============================================================================
DOCUMENT :AAL-8308:Articles:FasterSpiral.PT.txt
==============================================================================
```

Speeding Up Spirals.......................Bob Sander-Cederlof

Several have written to us about Roger Keating's Spiral Screen Clear (AAL June 1983). Charles Putney, who you may remember as the first one to double the speed of the prime number program in AAL several years ago, has now applied his talent to unwinding the screen.

Roger's program ran in 55 seconds, my table-lookup for BASCALC shortened it to 40 seconds. Charlie wrote the whole thing out as one long string of LDA-STA pairs, and trimmed the time to only 7 seconds!

Let's see...there are 960 characters on the screen. If I write a LDA-STA pair to move each byte ahead one position along the spiral path, I will have 959 such pairs. Each LDA and each STA will take 3 bytes, so the program to shift the whole screen one step around the spiral path will take 2x3x959 = 5754 bytes. Add another 5 bytes to LDA #$A0 and store it in the center of the screen before the first rotation. Then add some code to re- run the 959 steps 959 more times, so that the whole screen clears, and you get Charlie's program, 5777 bytes.

Now try to type it all in! Don't worry, we aren't even going to list it here. It will be on the next Quarterly disk, though.

Charlie decided to use five macros, to decrease the amount of manual labor involved. He defined a macro named MOVE which builds the LDA-STA pair for a pair of arguments:

```
       .MA MOVE
       LDA ]1
       STA ]2
       .EM
```

Then he defined one macro for each leg of the spiral: MOVED, MOVEL, MOVEU, and MOVER for down, left, up, and right respec- tively. With a few comment lines, the macro definitions take a mere 488 lines! The macros are each called with three parameters:

```
       >MOVED col,low.row,high.row
       >MOVEL row,low.col,high.col
       >MOVEU col,low.row,high.row
       >MOVER row,low.col,high.col
```

The definitions out of the way, it only remains to write 12 sets of 4 macro calls, or 48 lines, and a driving loop to do it all 960 times. Here is a condensed listing of the actual code part of Charlie's program:

```
NGE TO ELITE TYPE AND SPACING
6400 *--------------------------------
```

```
6410 *
6420 *       SPIRAL PROGRAM
6430         .OR $6000    OUT OF THE WAY
6440         .TF SPIRAL.OBJ
6450 *
6460 *
6470 SPIRAL LDA #' '+$80 GET A SPACE
6480         STA R12+12   PUT IT IN CENTER
6490         LDX #960     HOW MANY TIMES ?
6500         LDY /960     HIGH ORDER
6510 *
6520 SPI1    >MOVED 0,0,23
6530         >MOVEL R0,0,39
6540         >MOVEU 39,0,23
6550         >MOVER R23,1,39
6560 *
6570         >MOVED 1,1,23
6580         >MOVEL R1,1,38
6590         >MOVEU 38,1,22
6600         >MOVER R22,2,38
6610 *
6620         >MOVED 2,2,22
6630         >MOVEL R2,2,37
6640         >MOVEU 37,2,21
6650         >MOVER R21,3,37
6660 *
6670         >MOVED 3,3,21
6680         >MOVEL R3,3,36
6690         >MOVEU 36,3,20
6700         >MOVER R20,4,36
6710 *
6720         >MOVED 4,4,20
6730         >MOVEL R4,4,35
6740         >MOVEU 35,4,19
6750         >MOVER R19,5,35
6760 *
6770         >MOVED 5,5,19
6780         >MOVEL R5,5,34
6790         >MOVEU 34,5,18
6800         >MOVER R18,6,34
6810 *
6820         >MOVED 6,6,18
6830         >MOVEL R6,6,33
6840         >MOVEU 33,6,17
6850         >MOVER R17,7,33
6860 *
6870         >MOVED 7,7,17
6880         >MOVEL R7,7,32
6890         >MOVEU 32,7,16
6900         >MOVER R16,8,32
6910 *
6920         >MOVED 8,8,16
6930         >MOVEL R8,8,31
6940         >MOVEU 31,8,15
```

```
6950        >MOVER R15,9,31
6960 *
6970        >MOVED 9,9,15
6980        >MOVEL R9,9,30
6990        >MOVEU 30,9,14
7000        >MOVER R14,10,30
7010 *
7020        >MOVED 10,10,14
7030        >MOVEL R10,10,29
7040        >MOVEU 29,10,13
7050        >MOVER R13,11,29
7060 *
7070        >MOVED 11,11,13
7080        >MOVEL R11,11,28
7090        >MOVEU 28,11,12
7100        >MOVER R12,12,28
7110 *
7120        DEX
7130        CPX #$FF
7140        BNE SPI2
7150        DEY
7160        CPY #$FF
7170        BNE SPI2
7180        RTS
7190 SPI2   JMP SPI1
```

NOW CHANGE BACK TO PICA
Remember, the whole source with the full macro definitions will be on the next quarterly disk ($15, for all source code in issues July-August-September 1983).

Because Charlie's program makes such heavy use of macros, it takes considerable time to assemble. He timed it at nearly two minutes. If the program were written out the long way, without macros, it would take only about 20 seconds to assemble.

Charlie pointed out that we are needlessly moving the center of the spiral, which is already blank. As the blanked portion grows, this becomes very significant. In fact, by eliminating moving the cleared portion, the time could be further reduced to only 3 1/2 seconds. Each LDA-STA takes 8 cycles. The long way takes 959*960 pairs, plus some overhead. Ignoring the overhead, we get 7365120 cycles, or about 7.2 seconds. Forgetting the blanked stuff makes it 3.6 seconds. Any takers?

And I was just wondering...how about an Applesoft program which writes the 959 LDA-STA pairs as assembly language source on a text file? Or POKEs the actual object code, by computing the addresses necessary, into a binary buffer area. Again, any takers?

==========================================================================
DOCUMENT :AAL-8308:Articles:Front.Page.txt
==========================================================================

$1.50

Volume 3 -- Issue 11                    August, 1983

In This Issue...



Mailing AAL

Let's review how AAL is mailed, when you should expect to receive it, and what to do about it when you don't.  Most of you get your newletter by Bulk Mail, which is a little erratic.  You should receive your issue around the third week of each month, but don't start worrying until the end of the month.  If you haven't received an issue by the end of the month, call or write and we'll send a replacement. The Post Office does not forward Bulk Mail, so make certain to tell us if you move.  Those of you who have First Class Mail subscriptions should receive your issue around the tenth of the month, and certainly before the twentieth.

The number in the upper right corner of your mailing label is the expiration date of your subscription.  If that number is 8308, you're holding your last issue and better renew now.  We send out postcards when your subscription is about to expire, and when it has expired. All you have to do is send us a check, or phone with a charge card number, and we'll keep your AAL coming.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
================================================================================
DOCUMENT :AAL-8308:Articles:IIe.Auxmem.Bugs.txt
================================================================================
```

Using Auxiliary Memory in the //e.............David C. Johnson
Ridgefield, CT

When I bought my Apple //e (3 days after they became available!), I also got the Extended 80-Column Text Card. I wanted it both to have 80 column text capability and a full complement of Apple Computer Inc. supported memory. However, Apple only supplied two small subroutines in ROM and incomplete (but otherwise excellent) documentation in their manuals, in "support" of the auxiliary memory.

I say "incomplete" because two I/O locations that I used in my program are not mentioned (in English anyway) anywhere in the manuals except in the listings of the 80-column firmware. The two I/O locations are $C011 & $C012 which I call READ.BSR.BANK & READ.BSR.RAM.READ. Apple evidently intends to let software developers determine how the auxiliary memory is to be used.

Well here goes: my program is called "USE.AUXMEM". This program allows you to access the "other" 64K in a manner most Apple users should already be familiar with: monitor commands.

The simplest way to see what I mean is to type in & assemble the program (not so simple), type :"MGO G", :"PR#3" and then :"$^Y" (that is control-Y). You will get a bell and the monitor's prompt. Any monitor commands you type now will "use" the auxiliary memory. Try these now:

```
       *3D0:55
       *3D0     (double nickels, right?)
       *^Y      (back to SCASM!)
       :$3D0    (a $4C!)
```

You should note that control-Y while using the auxiliary memory returns to main memory with everything as it was. Now try these:

```
       :$^Y 3D0
       *3D0 ^Y
```

After the second control-Y returned to main memory, SCASM finished the first command line!

The reason I had you type :"PR#3" before is quite simple: things don't all work right without the 80 column firmware active; specifically, right-arrow & escape functions. You can also type "escape 4" if you don't want 80 columns.

But wait a minute, if you read the 80-column firmware listing (carefully), you know that it does NOT work with the auxiliary memory enabled (as doesn't the regular 40-column firmware), so how is this all working? Well, the I/O hooks in the auxiliary memory zero page point to routines in USE.AUXMEM which switch to main memory, perform the I/O, switch back to auxiliary memory, and return to the monitor. The monitor executes its commands between I/O calls while auxiliary memory is enabled. These switchings also change the bank switched memory state.

The USE.AUXMEM program has two other control-Y commands. They implement the crossbank subroutines AUXMOVE & XFER (supplied in ROM) as monitor commands. See the comments at the top of the source listing for their syntax.

About Some //e Monitor Bugs...

One routine, USE.AUXMEM.CONTROL.Y.HANDLER, deserves a special note. It compensates for a bug in the Apple //e version of the monitor: when parsing a control-Y command the ASCII string "Bryan" at $FEC5 is executed as instructions prior to JMPing to USRADR ($03F8). This bug has a long history.

In the original Apple monitor the CHRSRCH loop ($FF78 - $FF81) scans the CHRTBL ($FFCC - $FFE2) from end to beginning, which matches the $B2 at $FFCD causing TOSUB ($FFBE - $FFCB) to load the $C9 at $FFE4 and RTS to USR ($FECA) which is a JMP USRADR ($03F8).

Things started to go astray when the autostart ROM was created, and the Apple II Plus. To make room for new features, (like printing "APPLE ][" at the top of the screen on power up, and like the escape-IJKM cursor motion), the TRACE and STEP commands were removed. To disable the entries for Trace and Step in CHRTBL, the bytes for "T" and "S" were changed: ($FFCF:B2 & $FFD2:B2, also $FFE9:C4). Locations $FEC5 - $FEC9, immediately prior to USR, were changed to NOPs.

Unfortunately, someone forgot that CHRTBL is searched from end to beginning, causing a control-Y command to be matched with the $B2 at $FFD2, corresponding to the branch address in SUBTBL at $FFE9. So when you type a control-Y command the monitor branches to $FFC5 and executes the 5 NOPs. If $FFE9 had been changed to $C9 instead of $C4, everything would have still been fine.

Executing 5 NOPs is not a bad bug. But when the Apple //e monitor was created those 5 NOPs were replaced by the string "Bryan". In hex it is C2 F2 F9 E1 EE. The 6502 instruction set does not include a definition for $C2, but after a little investigation, or after reading Bob Sander-Cederlof's article in AAL March 1981, you find out that $C2 acts like a two-byte NOP. The "r" is skipped over. The "yan", however, is a SBC $EEE1,Y instruction.

The USE.AUXMEM.CONTROL.Y.HANDLER uses the passed contents of the A & X registers to decide which of the three control-Y commands you've typed. The SBC $EEE1,Y changes the A register so its contents must be reconstructed. The reconstruction is further complicated by the fact that the monitor leaves the carry flag set when it RTS's to $FEC5, while the S-C Assembler and Mini-Assembler leave the carry flag clear.

To restore the A register to its proper value you must set the carry to the complement of the value that it was set to prior to the SBC $EEE1,Y then execute ADC $EEE1,Y.

The Apple //e 80 column firmware also contains a bug. Because of the $11 at $C92A, the key sequence "ESCape ^L" causes a RTS to $4CCE. Location $C92A should contain a $10. This bug can be used to advantage if you feel like adding a secret command to your own software. Just be certain you have the code for your command starting at $4CCE, and that you are running in 80-column mode. Then whenever you type control-L in the escape mode (cursor is an inverse plus) your code will be executed.

I hope all of you enjoy using your auxiliary memory as much as I do.

Last Minute Note: David just called to report yet another oddity in the //e ROMs. In 40-column ESCape mode the (, 5, *, and + keys duplicate the arrow keys. That is, "ESC 5" moves the cursor right one space, just like ESC right arrow. This is a little bit weird, but it doesn't seem to hurt anything. The effect is caused by an unnecessary AND #$DF instruction at $C26E.

==============================================================================
DOCUMENT :AAL-8308:Articles:Kill.LIST.Cmd.txt
==============================================================================

Killing Applesoft's LIST Command...........Bob Sander-Cederlof

1. patch DOS to trap it?
2. patch CHRGET to trap it?
3. preventing access to the monitor
4. preventing use of peek/poke to undo patches.
5. the builtin LOCK at $D6
6. patching the forward link in the first line.
7. Using & followed by tokens above $EA for ordinary keywords.

==============================================================================
DOCUMENT :AAL-8308:Articles:Macro.Patches.txt
==============================================================================

## Some Small Patches..............................Bill Morgan

We've had several calls requesting the patch addresses for a couple of features in the S-C Macro Assemblers.

### Ansert?

In Version 1.1 of the Macro Assembler, Bob changed the CTRL-I (Insert) command in the EDIT mode to CTRL-A (for ADD). This was done because the Apple //e keyboard has the TAB key, which generates a CTRL-I code. It didn't seem to make much sense to have the TAB key do an insert operation, so he added a clear- to-next-tab-stop function for CTRL-I.

Well, a lot of people don't have //e's, or don't much care about the TAB key. A lot of us are used to CTRL-I for Insert, and would like to keep it that way.

The CTRL-A character ($81) is at $1C87 in the $1000 version, and at $DCB7 in the $D000 version. Just change that byte to a $89, and you'll have your good old CTRL-I back. If you want to keep the clear-to-tab-stop function, you can change the $89 at $1CC6 ($DCC6) to a $81. That will make CTRL-A do the clear-to- tab.

### .BS Filler Byte

The directive .BS <expr> skips over <expr> bytes when you are assembling to memory, and sends <expr> zero bytes to the target file when you are assembling to disk. Several people have asked how to change the zero to some other value.

For example, a freshly-erased EPROM contains all $FF bytes. When you burn data into the chip, you actually write in just the zero bits. If you are assembling code to be written into an EPROM, you want any fill bytes to be $FF, so you can add patches later without having to erase and re-write the whole chip.

The following table shows the addresses of the zero byte in the various versions of the Macro Assembler. Just change the indicated byte to the value you want to use for filler.

| Version | 1.0 | 1.1 | | | |
|---|---|---|---|---|---|
| | | 40-col | //e | Videx | STB |
| $1000 | 2D43 | 2D62 | 2D48 | 2E37 | 2E60 |
| $D000 | EE8F | EE86 | EE62 | EF5A | EF83 |

================================================================================
DOCUMENT :AAL-8308:Articles:More.68K.Boards.txt
================================================================================

Some More 68000 Boards

First let me apologize for an erroneous statement in the May '83 issue, in which I juxtaposed two unrelated facts in a cause-effect sentence. Many readers have sent corrections: I am told that grounding the DTACK signal has nothing to do with how much memory you can add. How did I ever get the idea that it did? If you want the straight scoop on this, subscribe to Digital Acoustics' newsletter "DTACK Grounded".

Digital Acoustics has announced a new board, called the "DTACK Grande". Almost sounds like "grounded", but this time it isn't. You get one megabyte of RAM and a 12.5 MHz 68000. RAM refresh is handled by an interrupt routine, with software. The overhead is only 4%, giving an effective speed of 10 MHz. Expansion connectors on the card can connect to another 15.7 megabytes. I'd say Saybrook has been passed by, but Hal Hardenburg beat me to it! (Digital Acoustics, 1415 E. McFadden, Suite F, Santa Ana, CA 92705. (714) 835-4884)

Mike Heckman at Anthro-Digital sent me some literature on another new 68000 board. Enhancement Technology Corporation calls it the "PDQ//". Specs include: 10 MHZ, 256K RAM, UCSD p-system, Applesoft-compatible BASIC. The price will be $1495, available by the end of August. We may be able to make you a deal on one of these. (ETC, P.O.Box 1267, Pittsfield, MA 01202. (413) 445-4219)

```
==================================================================
DOCUMENT :AAL-8308:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50

S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00

S-C Word Processor................................................$50.00
  As is, with fully commented source code.  Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)................... $79.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).......$49.00

The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00
Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00

Blank Diskettes (with hub rings).................package of 20 for $45.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Shift-Key Modifier................................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
     Only Revision level 7 or later Apples.
WICO Track Ball........................................($89.95)   $80.00
STB-80 80-column Display Board (STB Systems)...........($249.00)  $225.00
STB-128 128K RAM Card (STB Systems)....................($399.00)  $350.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
    "THE Book of Apple Software 1983 (with supplement)...($24.90)  $18.00
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
```

```
  "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
  "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
  "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
  "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
  "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
  "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
  "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
  "What's Where in the Apple", Second Edition..........($24.95)  $23.00
  "What's Where Guide" (updates first edition)..........($9.95)   $9.00
  "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
  "6502 Subroutines", Leventhal........................($17.95)  $17.00
Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

     *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
     ***                  (214) 324-2050                  ***
     *** We take Master Charge, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8308:Articles:Pitz.VCR.Patch.txt
==============================================================================

Tinkering with Variable Cross Reference...........Louis Pitz
De Witt, Iowa

I am a tinkerer! Yes I love to take programs and add features to improve them. Sometimes the "improved" version even works! Usually I learn a lot about humility, and occasionally a bit about programming.

A case in point is the program for doing an Applesoft Variable Cross Reference (from the November 1980 issue of Apple Assembly Line). I just recently got Quarterly Disk #1 with its source code, and so it became "tinker-time".

VCR works just fine, and is fast! But it only produces 40- column output, and I wanted both 40-column screen and 80-column printer hardcopy. Here are some patches which will do the job. It makes a good short example of changing output hooks in the middle of a program without goofing up DOS.

```
1060        .TF B.VCRP  "P" FOR PRINTER VERSION

4534         LDA #0     RESET COUNTER TO 0
4538         STA $6     FOR EACH VARIABLE

4821         INC $6     COUNT THE SCREEN LINE
4822         LDA $6
4823         AND #1     LOOK AT ODD-EVEN BIT
4824         BEQ TAB.NEW.LINE   BOTH SCRN AND PRINTER
4825         LDA #$FDF0         ONLY SCRN GETS NEW LINE
4826         STA $36    SO DISCONNECT PRINTER
4827         LDA /$FDF0
4828         STA $37
4829         JSR $3EA           PASS TO DOS
4830         JSR MON.CROUT      SCREEN ONLY
4831         LDA #$C100         REHOOK PRINTER
4832         STA $36
4833         LDA /$C100
4834         STA $37
4835         JSR $3EA           PASS TO DOS
4836         BNE .1             ...ALWAYS
```

To use the printer version of VCR, BRUN B.VCRP. This sets up the ampersand vector. Then LOAD your Applesoft program. Use PR#1 to turn on your printer. Then type "&" and RETURN, and watch the cross reference.

If your printer is in some slot other than 1, change lines 4831 and 4833 to the correct value ($Cs00, where s=slot#).

=============================================================================
DOCUMENT :AAL-8308:Articles:Reverse.Nybbles.txt
=============================================================================

Reversing, Getting, and Putting Nybbles....Bob Sander-Cederlof

In the process of de-crypting a large data base, I needed to reverse the nybbles in each of roughly 32000 bytes. There are probably a lot of ways to do this, but I found one which takes only 12 bytes to reverse the nybbles in the A-register.

Just to be sure we agree on what I am talking about, here is a little diagram:

```
       a b c d   e f g h

       e f g h   a b c d
```

One way, sort of brute force, involves breaking the nybbles out and remerging them:

```
       LDA (PNTR),Y
       ASL             SHIFT EFGH LEFT
       ASL
       ASL
       ASL
       STA TEMP
       LDA (PNTR),Y
       LSR             SHIFT ABCD RIGHT
       LSR
       LSR
       LSR
       ORA TEMP        RE-MERGE NYBBLES
       STA (PNTR),Y
```

From another perspective, I am trying to rotate the data byte half-way around. But if I try to do it with ROL or ROR instructions, one bit gets left in CARRY, and an extra bit gets inserted in the middle. Here is how I finally did it:

```
       LDA (PNTR),Y    abcd efgh
       ASL             bcde fgh0
       ADC #0          bcde fgha
       ASL             cdef gha0
       ADC #0          cdef ghab
       ASL             defg hab0
       ADC #0          defg habc
       ASL             efgh abc0
       ADC #0          efgh abcd
       STA (PNTR),Y
```

Each ASL-ADC pair shifts the byte around one bit. The ASL shifts the leftmost bit into the CARRY bit, and a zero into the right end. The ADC #0 adds CARRY into the rightmost bit.

Naturally, curiosity forces me to look at the possibility of shifting right one bit also. We have LSR and ROR, of course, but both of these leave the shifted out bit in CARRY. I want that bit back in the sign position, like this:

    ABCDEFGH should become HABCDEFG

Two similar methods come to mind, depending on how I might use it. If the byte to be shifted is in A-reg, and needs to remain there, and I don't want to upset any other registers, I can do it like this:

```
PHA     save unshifted value
LSR     get rightmost bit in CARRY
PLA     restore unshifted value
ROR     shift again, putting right bit on left
```

If the byte to be shifted is in memory, and I want the results to be in memory, I might do it like this:

```
LDA FLAG
LSR        RIGHTMOST BIT INTO CARRY
ROR FLAG   SHIFT BYTE, PUTTING RIGHT INTO LEFT
```

Note that I can branch according to the value of the bit which moved around by using BMI or BPL, because that bit is the new sign bit.

The last method above can be useful when you have a program that needs to alternate between two paths. For example, suppose I write a program to pick up the "next nybble" from a data area. The first time I call it, I want to get the left nybble of the first byte. Next time, the right nybble of the same byte. Next time the left nybble of the next byte. And so on.

I might store the value $55 in FLAG initially, and then use LDA FLAG, LSR, ROR FLAG, to shift it around. FLAG will alternate between $55 and $AA. My subroutine can alternate between left and right nybbles.

Not to leave you hanging, I wrote "get next nybble" and "put next nybble" subroutines. By the time I finished polishing, yet another technique had surfaced for rotating the $55/$AA flag. I used this new method so as not disturb the contents of the A-register.

To set up either routine, the address of the beginning of the data area must be put into PNTR and PNTR+1, and $55 must be put into FLAG.

    <<<routines here>>>

=============================================================================
DOCUMENT :AAL-8308:Articles:Wetzels.Patches.txt
=============================================================================

Bringing Some Patches Together.....................Jim Wetzel

Earlier this year I decided to break down and finally buy an 80-column card for my Apple II+. After all, it's cheaper than a IIe. I was just about to type in the Videx patches from AAL Volume 2, No.11, when Bob announced Version 1.1. Well with the Videx patches and all the new features I just couldn't pass up his offer. After a call to Bob and a three day wait I had version 1.1 of the S-C Macro Assembler.

While testing out the new version I soon discovered most of the patches I had applied to version 1.0 would not work properly. The addresses of the routines/tables had all moved. After a few hours work and a lot of dis-assembling I would like to share the new locations with AAL readers and bring some of the patches together.

First I will describe the new addresses and then show how I used them.

The Escape Function Table is now located at $14AB-$14C6 <ESC-@ thru ESC-M>. This is a group of two-byte addresses (minus 1, because they are of the PHA-PHA-RTS variety) of the routines to handle the escape functions.

The Edit Function Table is now located at $1CB4-$1CE3 <ctrl-@ thru ctrl-X>. This table is somewhat different. Each entry is three bytes long and it contains the control character and the address-minus-1 of the routine to handle the function.

Location $14D3 contains the dash count <$26> for the ESC-L function.

Location $13FF contains a JSR to the monitor Bell routine. This is the end of the input checker, the JSR BELL is executed when an invalid character is entered, and a good place to put a JSR to an extended input processor.

These locations are valid for the regular version and the Videx version which load at $1000. For language card users just add $C000 to the address. My hat is off to Bob for adding all the features of Videx and still keeping the assembler looking the same. I have not checked the STB or the IIe versions for compatibility but, with a little bit of work and knowing what to look for it should be an easy process.

Now, what can you do with this information? I have modified Bob's language card loader to show you (figure 1). With the exception of the REM statements, lines 1000-1140 of the file are as Bob supplied; after that the changes begin. I will not spend a lot of time explaining the routines themselves because they are all well documented in the referenced AAL articles.

The first thing I do is load in my extended input processor (figure 2) at $F600. There appears to be about two free pages after the assembler and before monitor in the language card version. For standard version users just move the symbol table up as described AAL Vol. 2, No. 9. My input processor is a combination of Auto Catalog (AAL 2.9) and Toggling Upper/Lower Case (AAL 3.3). Next I modify the JSR BELL to JSR CONTROL.A.

Once you have control you can add any routines you wish (such as R. F. O'Brien's Auto/Manual Toggle AAL 2.11). For now I am only interested in an upper/lower case toggle.

Line 1170 modifies the ESC-C function to JSR to my routine for auto Catalog. Remember this should be the address of the routine - 1. Lines 1180-1190 change the cursor to a blinking underline (as described in AAL 3.5) along with line 1200 which changes the number of "-"'s from 38 to 64 (I found 68 to be too many).

Last but not least is an answer to Steve Mann's request for a upper/lower case toggle in EDIT mode. In version 1.1 Bob changed the ctrl-I key function in EDIT mode and added a ctrl-A key function in its place. He did it so that the //e TAB key, which generates control-I, would really mean TAB.

Well Bob, I like mnemonic commands (like ctrl-I for Insert), and think the older Apples should still take precedence. Line 1210 changes the ctrl-A key to branch to my upper/lower case toggle routine, just past the character check, and line 1220 changes the ctrl-I routine back to its proper function (this was the address found in the ctrl-A area).

I hope these patches will be useful to other AAL readers not only for what they do, but for how they do it.

==============================================================================
DOCUMENT :AAL-8308:Articles:Whisper.VolCtrl.txt
==============================================================================

"One more beep and you're out!"

We have uncovered another neat new Apple accessory:  a volume control for the speaker!  If other people within earshot of your computer are trying to sleep, or just can't take another five minutes of bells, beeps, and buzzes, the WHISPER VOLUME CONTROL is for you.  The Apple version works with II, II Plus, //e, or ///.  All you have to do to install it is take the case off your Apple, unplug the speaker wire from the board, plug in the WVC cable, and plug the speaker wire into the other end of the WVC connector.  You can also get WVC for the IBM/PC.  The retail price is $22.95 for the standard version, or $25.95 with a headphone jack, from Information Dynamics Corp., 1251 Exchange Drive, Richardson, TX 75081.  Phone (214)783-8090.  Or if you like, buy them from us at $21 and $24, respectively.

```
===========================================================================
DOCUMENT :AAL-8308:DOS3.3:S.NybbleGetPut.txt
===========================================================================

 1000  *SAVE NYBBLE GET & PUT
 1010  *--------------------------------
 1020  PNTR   .EQ 0 AND 1
 1030  FLAG   .EQ 2
 1040  *--------------------------------
 1050  *      PUT NEXT NYBBLE AT (PNTR)
 1060  *      IF FLAG = $55, PUT LEFT NYBBLE
 1070  *              = $AA, PUT RIGHT NYBBLE
 1080  *--------------------------------
 1090  PUT.NEXT.NYBBLE
 1100         LDX #0
 1110         LSR FLAG          $55 OR $AA
 1120         BCS .1            ...IT WAS $AA, NOW $54
 1130  *---STORE IN LEFT NYBBLE---------
 1140         ASL          FLAG NOW $AA
 1150         ASL
 1160         ASL
 1170         ASL
 1180         STA (PNTR,X)
 1190         RTS
 1200  *---STORE IN RIGHT NYBBLE--------
 1210  .1     ORA (PNTR,X) MERGE WITH LEFT NYBBLE
 1220         STA (PNTR,X)
 1230         INC FLAG     MAKE $54 INTO $55
 1240         INC PNTR     MOVE PNTR TO NEXT BYTE
 1250         BNE .2
 1260         INC PNTR+1
 1270  .2     RTS
 1280  *--------------------------------
 1290  *      GET NEXT NYBBLE
 1300  *      IF FLAG = $55, GET LEFT NYBBLE
 1310  *              = $AA, GET RIGHT NYBBLE
 1320  *--------------------------------
 1330  GET.NEXT.NYBBLE
 1340         LDX #0
 1350         LSR FLAG          WAS $55 OR $AA
 1360         LDA (PNTR,X)    GET BYTE WITH NYBBLES
 1370         BCS .1            ...WAS $AA, NOW $54
 1380  *---GET LEFT NYBBLE--------------
 1390         LSR
 1400         LSR
 1410         LSR
 1420         LSR
 1430         RTS
 1440  *---GET RIGHT NYBBLE--------------
 1450  .1     INC FLAG     MAKE $54 INTO $55
 1460         INC PNTR     ADVANCE TO NEXT BYTE
 1470         BNE .2
 1480         INC PNTR+1
```

```
1490 .2      AND #$0F      ISOLATE NYBBLE
1500         RTS
1510 *--------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8308:DOS3.3:S.PutneySpiral.txt
=============================================================================

 1000  *SAVE S.PUTNEY'S SPIRAL
 1010  *
 1020  *
 1030  *      FAST SPIRAL SCREEN CLEAR
 1040  *
 1050  *      CHARLES H. PUTNEY
 1060  *      18 QUINNS ROAD
 1070  *      SHANKILL
 1080  *      CO. DUBLIN
 1090  *      IRELAND
 1100  *
 1110  *
 1120  *
 1130  *
 1140  *--------------------------------
 1150  *
 1160  *      TEXT PAGE BASE ADDRESSES
 1170  *
 1180  *
 1190  R0     .EQ $400
 1200  R1     .EQ $480
 1210  R2     .EQ $500
 1220  R3     .EQ $580
 1230  R4     .EQ $600
 1240  R5     .EQ $680
 1250  R6     .EQ $700
 1260  R7     .EQ $780
 1270  R8     .EQ $428
 1280  R9     .EQ $4A8
 1290  R10    .EQ $528
 1300  R11    .EQ $5A8
 1310  R12    .EQ $628
 1320  R13    .EQ $6A8
 1330  R14    .EQ $728
 1340  R15    .EQ $7A8
 1350  R16    .EQ $450
 1360  R17    .EQ $4D0
 1370  R18    .EQ $550
 1380  R19    .EQ $5D0
 1390  R20    .EQ $650
 1400  R21    .EQ $6D0
 1410  R22    .EQ $750
 1420  R23    .EQ $7D0
 1430  *
 1440  *
 1450  *--------------------------------
 1460  *
 1470  *      MACRO DEFINITIONS
 1480  *
```

```
1490  *
1500         .MA MOVE
1510         LDA ]1
1520         STA ]2
1530         .EM
1540  *
1550  *
1560  *
1570         .MA MOVER    MOVE RIGHT ROW,COLLOW,COLHIGH FROM 0-39
1580         .DO ]3>]2
1590         >MOVE ]1+]3-1,]1+]3
1600         .FIN
1610         .DO ]3-1>]2
1620         >MOVE ]1+]3-2,]1+]3-1
1630         .FIN
1640         .DO ]3-2>]2
1650         >MOVE ]1+]3-3,]1+]3-2
1660         .FIN
1670         .DO ]3-3>]2
1680         >MOVE ]1+]3-4,]1+]3-3
1690         .FIN
1700         .DO ]3-4>]2
1710         >MOVE ]1+]3-5,]1+]3-4
1720         .FIN
1730         .DO ]3-5>]2
1740         >MOVE ]1+]3-6,]1+]3-5
1750         .FIN
1760         .DO ]3-6>]2
1770         >MOVE ]1+]3-7,]1+]3-6
1780         .FIN
1790         .DO ]3-7>]2
1800         >MOVE ]1+]3-8,]1+]3-7
1810         .FIN
1820         .DO ]3-8>]2
1830         >MOVE ]1+]3-9,]1+]3-8
1840         .FIN
1850         .DO ]3-9>]2
1860         >MOVE ]1+]3-10,]1+]3-9
1870         .FIN
1880         .DO ]3-10>]2
1890         >MOVE ]1+]3-11,]1+]3-10
1900         .FIN
1910         .DO ]3-11>]2
1920         >MOVE ]1+]3-12,]1+]3-11
1930         .FIN
1940         .DO ]3-12>]2
1950         >MOVE ]1+]3-13,]1+]3-12
1960         .FIN
1970         .DO ]3-13>]2
1980         >MOVE ]1+]3-14,]1+]3-13
1990         .FIN
2000         .DO ]3-14>]2
2010         >MOVE ]1+]3-15,]1+]3-14
2020         .FIN
```

```
2030        .DO ]3-15>]2
2040        >MOVE ]1+]3-16,]1+]3-15
2050        .FIN
2060        .DO ]3-16>]2
2070        >MOVE ]1+]3-17,]1+]3-16
2080        .FIN
2090        .DO ]3-17>]2
2100        >MOVE ]1+]3-18,]1+]3-17
2110        .FIN
2120        .DO ]3-18>]2
2130        >MOVE ]1+]3-19,]1+]3-18
2140        .FIN
2150        .DO ]3-19>]2
2160        >MOVE ]1+]3-20,]1+]3-19
2170        .FIN
2180        .DO ]3-20>]2
2190        >MOVE ]1+]3-21,]1+]3-20
2200        .FIN
2210        .DO ]3-21>]2
2220        >MOVE ]1+]3-22,]1+]3-21
2230        .FIN
2240        .DO ]3-22>]2
2250        >MOVE ]1+]3-23,]1+]3-22
2260        .FIN
2270        .DO ]3-23>]2
2280        >MOVE ]1+]3-24,]1+]3-23
2290        .FIN
2300        .DO ]3-24>]2
2310        >MOVE ]1+]3-25,]1+]3-24
2320        .FIN
2330        .DO ]3-25>]2
2340        >MOVE ]1+]3-26,]1+]3-25
2350        .FIN
2360        .DO ]3-26>]2
2370        >MOVE ]1+]3-27,]1+]3-26
2380        .FIN
2390        .DO ]3-27>]2
2400        >MOVE ]1+]3-28,]1+]3-27
2410        .FIN
2420        .DO ]3-28>]2
2430        >MOVE ]1+]3-29,]1+]3-28
2440        .FIN
2450        .DO ]3-29>]2
2460        >MOVE ]1+]3-30,]1+]3-29
2470        .FIN
2480        .DO ]3-30>]2
2490        >MOVE ]1+]3-31,]1+]3-30
2500        .FIN
2510        .DO ]3-31>]2
2520        >MOVE ]1+]3-32,]1+]3-31
2530        .FIN
2540        .DO ]3-32>]2
2550        >MOVE ]1+]3-33,]1+]3-32
2560        .FIN
```

```
2570        .DO ]3-33>]2
2580        >MOVE ]1+]3-34,]1+]3-33
2590        .FIN
2600        .DO ]3-34>]2
2610        >MOVE ]1+]3-35,]1+]3-34
2620        .FIN
2630        .DO ]3-35>]2
2640        >MOVE ]1+]3-36,]1+]3-35
2650        .FIN
2660        .DO ]3-36>]2
2670        >MOVE ]1+]3-37,]1+]3-36
2680        .FIN
2690        .DO ]3-37>]2
2700        >MOVE ]1+]3-38,]1+]3-37
2710        .FIN
2720        .DO ]3-38>]2
2730        >MOVE ]1+]3-39,]1+]3-38
2740        .FIN
2750        .EM
2760  *
2770  *
2780  *
2790        .MA MOVEL    MOVE LEFT ROW,COLLOW,COLHIGH FROM 0-39
2800        .DO ]3>]2
2810        >MOVE ]1+]2+1,]1+]2
2820        .FIN
2830        .DO ]3-1>]2
2840        >MOVE ]1+]2+2,]1+]2+1
2850        .FIN
2860        .DO ]3-2>]2
2870        >MOVE ]1+]2+3,]1+]2+2
2880        .FIN
2890        .DO ]3-3>]2
2900        >MOVE ]1+]2+4,]1+]2+3
2910        .FIN
2920        .DO ]3-4>]2
2930        >MOVE ]1+]2+5,]1+]2+4
2940        .FIN
2950        .DO ]3-5>]2
2960        >MOVE ]1+]2+6,]1+]2+5
2970        .FIN
2980        .DO ]3-6>]2
2990        >MOVE ]1+]2+7,]1+]2+6
3000        .FIN
3010        .DO ]3-7>]2
3020        >MOVE ]1+]2+8,]1+]2+7
3030        .FIN
3040        .DO ]3-8>]2
3050        >MOVE ]1+]2+9,]1+]2+8
3060        .FIN
3070        .DO ]3-9>]2
3080        >MOVE ]1+]2+10,]1+]2+9
3090        .FIN
3100        .DO ]3-10>]2
```

```
3110          >MOVE ]1+]2+11,]1+]2+10
3120          .FIN
3130          .DO ]3-11>]2
3140          >MOVE ]1+]2+12,]1+]2+11
3150          .FIN
3160          .DO ]3-12>]2
3170          >MOVE ]1+]2+13,]1+]2+12
3180          .FIN
3190          .DO ]3-13>]2
3200          >MOVE ]1+]2+14,]1+]2+13
3210          .FIN
3220          .DO ]3-14>]2
3230          >MOVE ]1+]2+15,]1+]2+14
3240          .FIN
3250          .DO ]3-15>]2
3260          >MOVE ]1+]2+16,]1+]2+15
3270          .FIN
3280          .DO ]3-16>]2
3290          >MOVE ]1+]2+17,]1+]2+16
3300          .FIN
3310          .DO ]3-17>]2
3320          >MOVE ]1+]2+18,]1+]2+17
3330          .FIN
3340          .DO ]3-18>]2
3350          >MOVE ]1+]2+19,]1+]2+18
3360          .FIN
3370          .DO ]3-19>]2
3380          >MOVE ]1+]2+20,]1+]2+19
3390          .FIN
3400          .DO ]3-20>]2
3410          >MOVE ]1+]2+21,]1+]2+20
3420          .FIN
3430          .DO ]3-21>]2
3440          >MOVE ]1+]2+22,]1+]2+21
3450          .FIN
3460          .DO ]3-22>]2
3470          >MOVE ]1+]2+23,]1+]2+22
3480          .FIN
3490          .DO ]3-23>]2
3500          >MOVE ]1+]2+24,]1+]2+23
3510          .FIN
3520          .DO ]3-24>]2
3530          >MOVE ]1+]2+25,]1+]2+24
3540          .FIN
3550          .DO ]3-25>]2
3560          >MOVE ]1+]2+26,]1+]2+25
3570          .FIN
3580          .DO ]3-26>]2
3590          >MOVE ]1+]2+27,]1+]2+26
3600          .FIN
3610          .DO ]3-27>]2
3620          >MOVE ]1+]2+28,]1+]2+27
3630          .FIN
3640          .DO ]3-28>]2
```

```
3650          >MOVE ]1+]2+29,]1+]2+28
3660          .FIN
3670          .DO ]3-29>]2
3680          >MOVE ]1+]2+30,]1+]2+29
3690          .FIN
3700          .DO ]3-30>]2
3710          >MOVE ]1+]2+31,]1+]2+30
3720          .FIN
3730          .DO ]3-31>]2
3740          >MOVE ]1+]2+32,]1+]2+31
3750          .FIN
3760          .DO ]3-32>]2
3770          >MOVE ]1+]2+33,]1+]2+32
3780          .FIN
3790          .DO ]3-33>]2
3800          >MOVE ]1+]2+34,]1+]2+33
3810          .FIN
3820          .DO ]3-34>]2
3830          >MOVE ]1+]2+35,]1+]2+34
3840          .FIN
3850          .DO ]3-35>]2
3860          >MOVE ]1+]2+36,]1+]2+35
3870          .FIN
3880          .DO ]3-36>]2
3890          >MOVE ]1+]2+37,]1+]2+36
3900          .FIN
3910          .DO ]3-37>]2
3920          >MOVE ]1+]2+38,]1+]2+37
3930          .FIN
3940          .DO ]3-38>]2
3950          >MOVE ]1+]2+39,]1+]2+38
3960          .FIN
3970          .EM
3980   *
3990   *
4000   *
4010          .MA MOVEU    MOVE UP COL,ROWLOW,ROWHIGH FROM 0-23
4020          .DO ]2<1
4030          .DO ]3+1>1
4040          >MOVE ]1+R1,]1+R0
4050          .FIN
4060          .FIN
4070          .DO ]2<2
4080          .DO ]3+1>2
4090          >MOVE ]1+R2,]1+R1
4100          .FIN
4110          .FIN
4120          .DO ]2<3
4130          .DO ]3+1>3
4140          >MOVE ]1+R3,]1+R2
4150          .FIN
4160          .FIN
4170          .DO ]2<4
4180          .DO ]3+1>4
```

```
4190          >MOVE ]1+R4,]1+R3
4200          .FIN
4210          .FIN
4220          .DO ]2<5
4230          .DO ]3+1>5
4240          >MOVE ]1+R5,]1+R4
4250          .FIN
4260          .FIN
4270          .DO ]2<6
4280          .DO ]3+1>6
4290          >MOVE ]1+R6,]1+R5
4300          .FIN
4310          .FIN
4320          .DO ]2<7
4330          .DO ]3+1>7
4340          >MOVE ]1+R7,]1+R6
4350          .FIN
4360          .FIN
4370          .DO ]2<8
4380          .DO ]3+1>8
4390          >MOVE ]1+R8,]1+R7
4400          .FIN
4410          .FIN
4420          .DO ]2<9
4430          .DO ]3+1>9
4440          >MOVE ]1+R9,]1+R8
4450          .FIN
4460          .FIN
4470          .DO ]2<10
4480          .DO ]3+1>10
4490          >MOVE ]1+R10,]1+R9
4500          .FIN
4510          .FIN
4520          .DO ]2<11
4530          .DO ]3+1>11
4540          >MOVE ]1+R11,]1+R10
4550          .FIN
4560          .FIN
4570          .DO ]2<12
4580          .DO ]3+1>12
4590          >MOVE ]1+R12,]1+R11
4600          .FIN
4610          .FIN
4620          .DO ]2<13
4630          .DO ]3+1>13
4640          >MOVE ]1+R13,]1+R12
4650          .FIN
4660          .FIN
4670          .DO ]2<14
4680          .DO ]3+1>14
4690          >MOVE ]1+R14,]1+R13
4700          .FIN
4710          .FIN
4720          .DO ]2<15
```

```
4730          .DO ]3+1>15
4740          >MOVE ]1+R15,]1+R14
4750          .FIN
4760          .FIN
4770          .DO ]2<16
4780          .DO ]3+1>16
4790          >MOVE ]1+R16,]1+R15
4800          .FIN
4810          .FIN
4820          .DO ]2<17
4830          .DO ]3+1>17
4840          >MOVE ]1+R17,]1+R16
4850          .FIN
4860          .FIN
4870          .DO ]2<18
4880          .DO ]3+1>18
4890          >MOVE ]1+R18,]1+R17
4900          .FIN
4910          .FIN
4920          .DO ]2<19
4930          .DO ]3+1>19
4940          >MOVE ]1+R19,]1+R18
4950          .FIN
4960          .FIN
4970          .DO ]2<20
4980          .DO ]3+1>20
4990          >MOVE ]1+R20,]1+R19
5000          .FIN
5010          .FIN
5020          .DO ]2<21
5030          .DO ]3+1>21
5040          >MOVE ]1+R21,]1+R20
5050          .FIN
5060          .FIN
5070          .DO ]2<22
5080          .DO ]3+1>22
5090          >MOVE ]1+R22,]1+R21
5100          .FIN
5110          .FIN
5120          .DO ]2<23
5130          .DO ]3+1>23
5140          >MOVE ]1+R23,]1+R22
5150          .FIN
5160          .FIN
5170          .EM
5180   *
5190   *
5200   *
5210          .MA MOVED    MOVE DOWN COL,ROWLOW,ROWHIGH FROM 0-23
5220          .DO ]2<23
5230          .DO ]3+1>23
5240          >MOVE ]1+R22,]1+R23
5250          .FIN
5260          .FIN
```

```
5270        .DO ]2<22
5280        .DO ]3+1>22
5290        >MOVE ]1+R21,]1+R22
5300        .FIN
5310        .FIN
5320        .DO ]2<21
5330        .DO ]3+1>21
5340        >MOVE ]1+R20,]1+R21
5350        .FIN
5360        .FIN
5370        .DO ]2<20
5380        .DO ]3+1>20
5390        >MOVE ]1+R19,]1+R20
5400        .FIN
5410        .FIN
5420        .DO ]2<19
5430        .DO ]3+1>19
5440        >MOVE ]1+R18,]1+R19
5450        .FIN
5460        .FIN
5470        .DO ]2<18
5480        .DO ]3+1>18
5490        >MOVE ]1+R17,]1+R18
5500        .FIN
5510        .FIN
5520        .DO ]2<17
5530        .DO ]3+1>17
5540        >MOVE ]1+R16,]1+R17
5550        .FIN
5560        .FIN
5570        .DO ]2<16
5580        .DO ]3+1>16
5590        >MOVE ]1+R15,]1+R16
5600        .FIN
5610        .FIN
5620        .DO ]2<15
5630        .DO ]3+1>15
5640        >MOVE ]1+R14,]1+R15
5650        .FIN
5660        .FIN
5670        .DO ]2<14
5680        .DO ]3+1>14
5690        >MOVE ]1+R13,]1+R14
5700        .FIN
5710        .FIN
5720        .DO ]2<13
5730        .DO ]3+1>13
5740        >MOVE ]1+R12,]1+R13
5750        .FIN
5760        .FIN
5770        .DO ]2<12
5780        .DO ]3+1>12
5790        >MOVE ]1+R11,]1+R12
5800        .FIN
```

```
5810        .FIN
5820        .DO ]2<11
5830        .DO ]3+1>11
5840        >MOVE ]1+R10,]1+R11
5850        .FIN
5860        .FIN
5870        .DO ]2<10
5880        .DO ]3+1>10
5890        >MOVE ]1+R9,]1+R10
5900        .FIN
5910        .FIN
5920        .DO ]2<9
5930        .DO ]3+1>9
5940        >MOVE ]1+R8,]1+R9
5950        .FIN
5960        .FIN
5970        .DO ]2<8
5980        .DO ]3+1>8
5990        >MOVE ]1+R7,]1+R8
6000        .FIN
6010        .FIN
6020        .DO ]2<7
6030        .DO ]3+1>7
6040        >MOVE ]1+R6,]1+R7
6050        .FIN
6060        .FIN
6070        .DO ]2<6
6080        .DO ]3+1>6
6090        >MOVE ]1+R5,]1+R6
6100        .FIN
6110        .FIN
6120        .DO ]2<5
6130        .DO ]3+1>5
6140        >MOVE ]1+R4,]1+R5
6150        .FIN
6160        .FIN
6170        .DO ]2<4
6180        .DO ]3+1>4
6190        >MOVE ]1+R3,]1+R4
6200        .FIN
6210        .FIN
6220        .DO ]2<3
6230        .DO ]3+1>3
6240        >MOVE ]1+R2,]1+R3
6250        .FIN
6260        .FIN
6270        .DO ]2<2
6280        .DO ]3+1>2
6290        >MOVE ]1+R1,]1+R2
6300        .FIN
6310        .FIN
6320        .DO ]2<1
6330        .DO ]3+1>1
6340        >MOVE ]1+R0,]1+R1
```

```
6350        .FIN
6360        .FIN
6370        .EM
6380 *
6390 *
6400 *--------------------------------
6410 *
6420 *      SPIRAL PROGRAM
6430        .OR $6000    OUT OF THE WAY
6440        .TF SPIRAL.OBJ
6450 *
6460 *
6470 SPIRAL LDA #' '+$80 GET A SPACE
6480        STA R12+12   PUT IT IN CENTER
6490        LDX #960     HOW MANY TIMES ?
6500        LDY /960     HIGH ORDER
6510 *
6520 SPI1   >MOVED 0,0,23
6530        >MOVEL R0,0,39
6540        >MOVEU 39,0,23
6550        >MOVER R23,1,39
6560 *
6570        >MOVED 1,1,23
6580        >MOVEL R1,1,38
6590        >MOVEU 38,1,22
6600        >MOVER R22,2,38
6610 *
6620        >MOVED 2,2,22
6630        >MOVEL R2,2,37
6640        >MOVEU 37,2,21
6650        >MOVER R21,3,37
6660 *
6670        >MOVED 3,3,21
6680        >MOVEL R3,3,36
6690        >MOVEU 36,3,20
6700        >MOVER R20,4,36
6710 *
6720        >MOVED 4,4,20
6730        >MOVEL R4,4,35
6740        >MOVEU 35,4,19
6750        >MOVER R19,5,35
6760 *
6770        >MOVED 5,5,19
6780        >MOVEL R5,5,34
6790        >MOVEU 34,5,18
6800        >MOVER R18,6,34
6810 *
6820        >MOVED 6,6,18
6830        >MOVEL R6,6,33
6840        >MOVEU 33,6,17
6850        >MOVER R17,7,33
6860 *
6870        >MOVED 7,7,17
6880        >MOVEL R7,7,32
```

```
6890        >MOVEU 32,7,16
6900        >MOVER R16,8,32
6910  *
6920        >MOVED 8,8,16
6930        >MOVEL R8,8,31
6940        >MOVEU 31,8,15
6950        >MOVER R15,9,31
6960  *
6970        >MOVED 9,9,15
6980        >MOVEL R9,9,30
6990        >MOVEU 30,9,14
7000        >MOVER R14,10,30
7010  *
7020        >MOVED 10,10,14
7030        >MOVEL R10,10,29
7040        >MOVEU 29,10,13
7050        >MOVER R13,11,29
7060  *
7070        >MOVED 11,11,13
7080        >MOVEL R11,11,28
7090        >MOVEU 28,11,12
7100        >MOVER R12,12,28
7110  *
7120        DEX
7130        CPX #$FF
7140        BNE SPI2
7150        DEY
7160        CPY #$FF
7170        BNE SPI2
7180        RTS
7190  SPI2   JMP SPI1
7200  *
7210  ZZSIZE .EQ *-SPIRAL
```

```
==============================================================
DOCUMENT :AAL-8308:DOS3.3:S.Wetzel11Patch.txt
==============================================================

 1000  *SAVE WETZEL'S PATCHES TO 1.1
 1010         .OR $F600
 1020         .TF SCM.PATCH
 1030  *---------------------------------------------------------
 1040  CH     .EQ $24
 1050  BASL   .EQ $28
 1060  YSAVE  .EQ $40
 1070  WBUF   .EQ $200
 1080  LCPROT .EQ $C080    LC Protect
 1090  LCWRT  .EQ $C083    LC Write enable
 1100  UCFLAG .EQ $D016    UC/LC Flag
 1110  BELL   .EQ $FF3A    Monitor Bell
 1120  *---------------------------------------------------------
 1130  CONTROL.A
 1140         CMP #$81     Was a CNTL-A entered
 1150         BNE ERROR    No - then signal error
 1160         LDA LCWRT    Write enable Language card
 1170         LDA LCWRT
 1180         LDA UCFLAG   Get upper case flag
 1190         EOR #$FF     Reverse it
 1200         STA UCFLAG   Put it back
 1210         LDA LCPROT   Write protect Language card
 1220         RTS
 1230  ERROR
 1240         JSR BELL     Ring bell to signal error
 1250         RTS          Return
 1260  *---------------------------------------------------------
 1270  ESCAPE.C
 1280         CPX #0       Start of line?
 1290         BNE .2       No, rtn
 1300         LDY #0
 1310  .1     LDA MSG,Y    Get message
 1320         STA WBUF,Y   Put in buffer
 1330         STA (BASL),Y Put on screen (40-column)
 1340         INY
 1350         CPY #7       Finished ?
 1360         BNE .1       Not yet
 1370         STY YSAVE
 1380         INY
 1390         STY CH       Tell assembler
 1400         TSX          this was an
 1410         LDA #$CC     ESC-L so it will
 1420         STA $103,X   exec command
 1430         LDX YSAVE
 1440  .2     RTS
 1450  MSG    .AS -/CATALOG/
```

```
==============================================================================
DOCUMENT :AAL-8308:DOS3.3:S.WetzelLoader.txt
==============================================================================

 1000  REM LOAD S-C MACRO ASSEMBLER (VIDEX)
 1010  REM INTO RAM AT $D000
 1020  REM LOAD PATCHES AT $F600
 1030  REM PATCH INPUT TEST TO CHECK FOR MY COMMANDS BEFORE ERROR
 1040  REM PATCH ESCAPE TABLE ($D4AB-) FOR ESCAPE-C
 1050  REM CHANGE CURSOR TO BLINKING UNDERLINE
 1060  REM PATCH ESC-L DASH LINE COUNT
 1070  REM PATCH EDIT CNTL-A TO MY ROUTINE
 1080  REM PATCH EDIT CNTL-I BACK TO INSERT FUNCTION
 1090  CALL-151
 1100  C081 C081
 1110  F800<F800.FFFFM
 1120  BLOAD S-C.ASM.MACRO.D000.VIDEX
 1130  300:A9 4C CD 00 E0 F0 12 8D 00 E0 A9 00 8D 01 E0 A9 D0 8D 02 E0
A9 CB 8D D1 03 60
 1140  300G
 1150  BLOAD SCM.PATCH
 1160  D3FF:20 00 F6
 1170  D4B1:19 F6
 1180  C0B0:0A 68
 1190  C0B0:0B 08
 1200  D4D3:40
 1210  DCB8:03 F6
 1220  DCC7:0B DC
 1230  C080
 1240  3D3G
```

```
==============================================================
DOCUMENT :AAL-8308:DOS3.3:SJohnson.AUXMEM.txt
==============================================================

 1000  *SAVE JOHNSON'S USE AUXMEM
 1010  *--------------------------------
 1020  * SWITCH.MIND Command:  ^Y
 1030  *
 1040  *   When in main bank, enters monitor in
 1050  *   auxmem BSR (hooks I/O through main
 1060  *   and brings USE.AUXMEM to auxmem too)
 1070  *   When in aux bank, returns to main bank
 1080  *   Best used w/80 column firmware active
 1090  *--------------------------------
 1100  * USE.AUXMOVE Command:  DEST<SOURCE.END^Y{CARRY}
 1110  *
 1120  *   DEST      = Destination in one bank
 1130  *   SOURCE    = Start in other bank
 1140  *   END       = End in other bank
 1150  *   CARRY     = Direction of move
 1160  *                (1 = Main Ram-->Card Ram)
 1170  *                (0 = Card Ram-->Main Ram)
 1180  *   DEST, SOURCE, & END must be: >=$0200 & <=$BFFF
 1190  *--------------------------------
 1200  * USE.XFER Command:  ADDRESS^Y{CARRY}{OVERFLOW}
 1210  *
 1220  *   ADDRESS   = Transfer address
 1230  *   CARRY     = Desired 48K Bank ($0200 - $BFFF)
 1240  *                (1 = Use 48K in Card Ram)
 1250  *                (0 = Use 48K in Main Ram)
 1260  *   OVERFLOW  = Desired ZP/STK/BSR
 1270  *                (1 = Use ZP/STK/BSR in Card Ram)
 1280  *                (0 = Use ZP/STK/BSR in Main Ram)
 1290  *   If using USE.XFER from auxmem, routine in main mem
 1300  *   MUST LDX BANK.SP.SAVE, TXS if it uses the stack at all
 1310  *--------------------------------
 1320  MON.BASL         .EQ $28,$29
 1330  MON.YSAV         .EQ $34
 1340  MON.CSWL         .EQ $36,$37
 1350  MON.KSWL         .EQ $38,$39
 1360  MON.A1           .EQ $3C,$3D   Source,Address
 1370  MON.A2           .EQ $3E,$3F   End
 1380  MON.A4           .EQ $42,$43   Dest
 1390  MON.STATUS       .EQ $48
 1400  *--------------------------------
 1410  IN                       .EQ $0200 - $02FF
 1420  BANK.X.SAVE              .EQ $03CC
 1430  BANK.BSR.BANK.SAVE       .EQ $03CD
 1440  BANK.BSR.RAM.READ.SAVE   .EQ $03CE
 1450  BANK.SP.SAVE             .EQ $03CF
 1460  TRANSFER                 .EQ $03ED,$03EE
 1470  MON.BRKV                 .EQ $03F0,$03F1
 1480  USRADR                   .EQ $03F8 - $03FA
```

```
1490  NMI                       .EQ $03FB - $03FD
1500  MON.IRQLOC                .EQ $03FE,$03FF
1510  *--------------------------------
1520  READ.MAIN.RAM             .EQ $C002
1530  READ.AUX.RAM              .EQ $C003
1540  WRITE.MAIN.RAM            .EQ $C004
1550  WRITE.AUX.RAM             .EQ $C005
1560  USE.MAIN.ZP.STK.BSR       .EQ $C008
1570  USE.AUX.ZP.STK.BSR        .EQ $C009
1580  READ.BSR.BANK             .EQ $C011
1590  READ.BSR.RAM.READ         .EQ $C012
1600  READ.RAM.READ.STATUS      .EQ $C013
1610  READ.RAM.WRITE.STATUS     .EQ $C014
1620  READ.ZP.STK.BSR.STATUS    .EQ $C016
1630  BSR.2.RAM.READ.ONLY       .EQ $C080
1640  BSR.2.ROM.READ.RAM.WRITE  .EQ $C081
1650  BSR.2.ROM.READ.ONLY       .EQ $C082
1660  BSR.2.RAM.READ.RAM.WRITE  .EQ $C083
1670  BSR.1.RAM.READ.ONLY       .EQ $C088
1680  BSR.1.ROM.READ.RAM.WRITE  .EQ $C089
1690  BSR.1.ROM.READ.ONLY       .EQ $C08A
1700  BSR.1.RAM.READ.RAM.WRITE  .EQ $C08B
1710  *--------------------------------
1720  AUXMOVE                   .EQ $C311
1730  XFER                      .EQ $C314
1740  MONITOR                   .EQ $F800 - $FFFF
1750  MON.OLDBRK                .EQ $FA59
1760  BEEP                      .EQ $FBDD
1770  MON.RDKEY                 .EQ $FD0C
1780  MON.JSR.CLREOL            .EQ $FD8B - $FD8D
1790  MON.COUT                  .EQ $FDED
1800  MON                       .EQ $FF65
1810  *--------------------------------
1820             .OR $0803
1830  USE.AUXMEM
1840  G
1850             JMP CONNECT.CONTROL.Y
1860  JMP.TO.RETURN.TO.MAIN
1870             JMP RETURN.TO.MAIN
1880  JMP.TO.RETURN.TO.AUX
1890             JMP RETURN.TO.AUX
1900  JMP.TO.SAVE.BSR.STATE
1910             JMP SAVE.BSR.STATE
1920  JMP.TO.RESTORE.BSR.STATE
1930             JMP RESTORE.BSR.STATE
1940  *--------------------------------
1950  CONNECT.CONTROL.Y
1960             LDA /USE.AUXMEM.CONTROL.Y.HANDLER
1970             STA USRADR+2
1980             LDA #USE.AUXMEM.CONTROL.Y.HANDLER
1990             STA USRADR+1
2000             LDA #$4C      JMP
2010             STA USRADR
2020             RTS
```

```
2030  *--------------------------------
2040  USE.AUXMEM.CONTROL.Y.HANDLER
2050  * Reconstruct monitor mode byte
2060  * after "Bryan" messed with it
2070  * ("Br" is NOPish)
2080             PHA
2090             LDA IN
2100             CMP #"$"
2110  * Branch w/Carry set causa S-C or Mini-Asm
2120             BEQ .1
2130             CLC
2140  .1         PLA
2150  * These lines are for you Bryan
2160             .DA #'y'
2170             .AS -'an'     Builds SBC $EEE1,Y
2180  * Check for user specified address
2190             CPX #$01
2200             BNE SWITCH.MIND
2210             TAY
2220  * Lesser complex is USE.XFER
2230             BEQ USE.XFER
2240  * Most complex is USE.AUXMOVE
2250  *--------------------------------
2260  USE.AUXMOVE
2270  * Fetch what should be a "0"
2280  * or "1" to be AUXMOVE's carry
2290             LDY MON.YSAV
2300             LDA IN,Y
2310  * Shift what we fetched to carry
2320             LSR
2330  * Save carry while comparing
2340             PHP
2350  * This is a "0" or "1" after a LSR
2360             CMP #"0"/2
2370             BNE INVALID.CARRY
2380             INC MON.YSAV
2390  * Recover Carry
2400             PLP
2410  CALL.AUXMOVE.WITH.CARRY
2420             JSR AUXMOVE
2430             RTS
2440  *--------------------------------
2450  USE.XFER
2460  * Set XFER Transfer address
2470  * from monitor parameter
2480             LDA MON.A1,X
2490             STA TRANSFER,X
2500             DEX
2510             BPL USE.XFER
2520  * Fetch what should be a "0"
2530  * or "1" to be XFER's carry
2540             LDY MON.YSAV
2550             LDA IN,Y
2560  * Shift what we fetched to carry
```

```
2570            LSR
2580 * Save carry for a while
2590            PHP
2600 * This is a "0" or "1" after a LSR
2610            CMP #"0"/2
2620            BNE INVALID.CARRY
2630            INC MON.YSAV
2640 * Fetch what should be a "0"
2650 * or a "1" to be XFER's overflow
2660            INY
2670            LDA IN,Y
2680 * Shift what we fetched to carry
2690            LSR
2700 * Save this carry too, while we compare
2710            PHP
2720 * This is a "0" or "1" after a LSR
2730            CMP #"0"/2
2740            BNE INVALID.OVERFLOW
2750            INC MON.YSAV
2760 * Recovered carry is valid overflow
2770            PLP
2780 * Move it back to bit 0
2790            ROL
2800 * Recover carry
2810            PLP
2820 * Construct overflow
2830            CLV
2840            AND #%0000.0001
2850            BEQ .1
2860            BIT SEV
2870 * Save BSR bank, BSR ram read, and SP
2880 * for any calls or returns to main/auxmem
2890 .1         JSR SAVE.BSR.STATE
2900            TSX
2910            STX BANK.SP.SAVE
2920 JMP.XFER.WITH.CARRY.AND.OVERFLOW
2930 * Routines in aux/main bank may jmp
2940 * to RETURN.TO.MAIN/AUX when done
2950 SEV        JMP XFER
2960 *--------------------------------
2970 INVALID.OVERFLOW
2980            PLP
2990 INVALID.CARRY
3000            PLP
3010 * Let's not process rest of line
3020            LDY MON.YSAV
3030            LDA #$8D
3040            STA IN,Y
3050            JMP BEEP
3060 *--------------------------------
3070 SWITCH.MIND
3080 * Check in main or aux now
3090            LDA READ.RAM.READ.STATUS
3100            BPL ENTER.AUX.MON
```

```
3110            JMP RETURN.TO.MAIN
3120 ENTER.AUX.MON
3130 * Move USE.AUXMEM to auxmem too
3140            LDA #USE.AUXMEM
3150            STA MON.A1
3160            STA MON.A4
3170            LDA /USE.AUXMEM
3180            STA MON.A1+1
3190            STA MON.A4+1
3200            LDA #USE.AUXMEM.END
3210            STA MON.A2
3220            LDA /USE.AUXMEM.END
3230            STA MON.A2+1
3240            SEC
3250            JSR AUXMOVE
3260 * Save BSR bank, BSR ram read, and SP
3270 * for calls and return to main mem
3280            JSR SAVE.BSR.STATE
3290            TSX
3300            STX BANK.SP.SAVE
3310 * Continue in auxmem w/rom
3320            STA READ.AUX.RAM
3330            STA WRITE.AUX.RAM
3340            STA USE.AUX.ZP.STK.BSR
3350            LDA BSR.2.ROM.READ.RAM.WRITE
3360            LDA BSR.2.ROM.READ.RAM.WRITE
3370 * What else but this too
3380            LDX #$FF
3390            TXS
3400 * Copy rom monitor to auxmem BSR
3410            LDY #MONITOR
3420            STY MON.A1
3430            STY MON.STATUS
3440            LDA /MONITOR
3450            STA MON.A1+1
3460 .1         LDA (MON.A1),Y
3470            STA (MON.A1),Y
3480            INY
3490            BNE .1
3500            INC MON.A1+1
3510            BNE .1
3520 * Now use auxmem BSR
3530            LDA BSR.2.RAM.READ.RAM.WRITE
3540            LDA BSR.2.RAM.READ.RAM.WRITE
3550 * Fix monitor in BSR
3560            LDA /DO.CLREOL
3570            STA MON.JSR.CLREOL+2
3580            LDA #DO.CLREOL
3590            STA MON.JSR.CLREOL+1
3600 * Hook I/O through main
3610            LDA #COUT.TO.MAIN
3620            STA MON.CSWL
3630            LDA #RDKEY.FROM.MAIN
3640            STA MON.KSWL
```

```
3650            LDA /COUT.TO.MAIN
3660            STA MON.CSWL+1
3670 *          LDA /RDKEY.FROM.MAIN
3680            STA MON.KSWL+1
3690 * USE.AUXMEM in auxmem too
3700            JSR CONNECT.CONTROL.Y
3710 * Do page 3 locs
3720            STA NMI
3730            LDA #MON
3740            STA NMI+1
3750            STA MON.IRQLOC
3760            LDA /MON
3770            STA NMI+2
3780            STA MON.IRQLOC+1
3790            LDA #MON.OLDBRK
3800            STA MON.BRKV
3810            LDA /MON.OLDBRK
3820            STA MON.BRKV+1
3830 * Enter monitor in auxmem BSR
3840            JMP MON
3850 *---------------------------------
3860 RETURN.TO.AUX
3870 * Continue in aux ram
3880            STA READ.AUX.RAM
3890            STA WRITE.AUX.RAM
3900            STA USE.AUX.ZP.STK.BSR
3910            JMP RETURN.COMMON
3920 RETURN.TO.MAIN
3930 * Continue in main ram
3940            STA READ.MAIN.RAM
3950            STA WRITE.MAIN.RAM
3960            STA USE.MAIN.ZP.STK.BSR
3970 RETURN.COMMON
3980 * Recover SP
3990            LDX BANK.SP.SAVE
4000            TXS
4010 RESTORE.BSR.STATE
4020            CLV
4030            LDX BANK.BSR.BANK.SAVE
4040            BPL .2
4050            LDX BANK.BSR.RAM.READ.SAVE
4060            BPL .1
4070            LDX BSR.2.RAM.READ.RAM.WRITE
4080            LDX BSR.2.RAM.READ.RAM.WRITE
4090            BVC .4
4100 .1         LDX BSR.2.ROM.READ.RAM.WRITE
4110            LDX BSR.2.ROM.READ.RAM.WRITE
4120            BVC .4
4130 .2         LDX BANK.BSR.RAM.READ.SAVE
4140            BPL .3
4150            LDX BSR.1.RAM.READ.RAM.WRITE
4160            LDX BSR.1.RAM.READ.RAM.WRITE
4170            BVC .4
4180 .3         LDX BSR.1.ROM.READ.RAM.WRITE
```

```
            LDX BSR.1.ROM.READ.RAM.WRITE
.4          RTS
*--------------------------------
SAVE.BSR.STATE
            LDX READ.BSR.BANK
            STX BANK.BSR.BANK.SAVE
            LDX READ.BSR.RAM.READ
            STX BANK.BSR.RAM.READ.SAVE
            RTS
*--------------------------------
DO.CLREOL
            LDA #"]"-'@'
COUT.TO.MAIN
* Save auxmem's X
            STX BANK.X.SAVE
* Save BSR bank, BSR ram read, and SP
* over call to main ram
            JSR SAVE.BSR.STATE
            TSX
            STX BANK.SP.SAVE
* Continue in main ram
            STA READ.MAIN.RAM
            STA WRITE.MAIN.RAM
            STA USE.MAIN.ZP.STK.BSR
* Recover SP
            LDX BANK.SP.SAVE
            TXS
            JSR RESTORE.BSR.STATE
            JSR MON.COUT
            JMP IO.COMMON
*--------------------------------
RDKEY.FROM.MAIN
* Repair monitor's sillier attempt
            STA (MON.BASL),Y
* Save auxmem's X
            STX BANK.X.SAVE
* Save BSR bank, BSR ram read, and SP
* over call to main ram
            JSR SAVE.BSR.STATE
            TSX
            STX BANK.SP.SAVE
* Continue in main ram
            STA READ.MAIN.RAM
            STA WRITE.MAIN.RAM
            STA USE.MAIN.ZP.STK.BSR
            LDX BANK.SP.SAVE       Recover SP
            TXS
            JSR RESTORE.BSR.STATE
            JSR MON.RDKEY
*--------------------------------
IO.COMMON
            STA READ.AUX.RAM       Continue in Aux RAM
            STA WRITE.AUX.RAM
            STA USE.AUX.ZP.STK.BSR
```

```
                LDX BANK.SP.SAVE       Recover SP
                TXS
                JSR RESTORE.BSR.STATE
                LDX BANK.X.SAVE        Recover X
                RTS
      *--------------------------------
      USE.AUXMEM.END           .EQ *-1
```

================================================================
DOCUMENT :AAL-8309:Articles:Amper.Monitor.txt
================================================================

Amper-Monitor..............................Bob Sander-Cederlof

It would be nice to be able to use monitor commands from within Applesoft, both in direct commands and within running Applesoft programs.  At least Kraig Arnett, from Homestead, Florida, thinks so.

I agree, and so I whipped out another handy-dandy &-subroutine for just that purpose.  I call it Amper-Monitor.  You can install it by BRUNning it from a binary file, or by adding some POKEs to your Applesoft program.  My listing shows it residing at the ever popular $300 address, but it can be reassebled to run anywhere.  Just remember to connect it properly to the Ampersand Vector.

Once Amper-Monitor is installed and hooked to the ampersand vector, you call it by typing an ampersand, a quotation mark, and a monitor command.  Here is a sample program showing some uses of the Amper-Monitor.

```
100 FOR I = 768 TO 855
110 READ D : POKE I,D : NEXT
120 CALL 768

130 &"300.357
140 &"380:12 34 56 78 9A BC DE F0
150 &"FBE2G
160 &"300L 380.387

200 DATA 169,11,141,246,3,169,3,141,247,3,96
210 DATA 201,34,208,70,32,177,0,160,0,177,184,201,0
220 DATA 240,8,9,128,153,0,2,200,208,242,169,141
230 DATA 153,0,2,152,24,101,184,133,184,144,2,230
240 DATA 185,32,199,255,32,167,255,132,52,160,23
250 DATA 136,48,23,217,204,255,208,248,192,21,240
260 DATA 8,32,190,255,164,52,76,52,3,32,197,255
270 DATA 76,0,254,76,201,222
```

Why did I choose to require the quotation mark after the ampersand? Because normally Applesoft would parse the line, eliminating blanks, changing DEF to a token instead of three hex digits, using ":" to end a line, and so on.  Using the "-mark prevents all this, leaving the line in raw ASCII form.  Here is a listing of the program in assembly language:

Lines 1200-1240 link in the ampersand vector.  This is the only part that would have to be changed if you move the routine.

When Applesoft sees an "&", it will JSR to AMPER.MONITOR.  The A-register will hold the character following the "&", which we hope is a quotation mark.  Lines 1270 and 1280 do this hoping.

Lines 1290-1380 copy the characters following the quotation mark into the monitor buffer starting at $200. If you typed in the &"... as a direct command, it is already in the monitor buffer but starts at $202, so it gets shifted over two bytes. If the command is in a program, it will be copied out of program space into $200. Applesoft has stripped off the sign bit from every byte, so my loop adds the sign bit back in to satisfy the monitor's requirements. Applesoft ends the line with a $00 byte, and the monitor wants $8D, so I fix that up too. I don't let colon terminate the line, because colon is a valid character in a monitor command line. I use "LDA (TXTPTR),Y" rather than repeated calls to AS.CHRGET because AS.CHRGET would eliminate blanks.

Lines 1390-1440 adjust the Applesoft pointer to the end of the line, so upon returning we won't get false syntax errors and the Applesoft program can continue executing.

Lines 1450-1590 parse the command line one command at a time, call on the monitor to execute each command, and finally return to Applesoft after the last command on the line. (The idea for this code came originally from code Steve Wozniak wrote for the mini-assembler in the old Apple monitor ROM.) Note that an illegal monitor command will result in a syntax error.

I thought it would now be possible to use the Amper-Monitor to write hex dumps on text files...BUT: Unfortunately DOS uses some critical zero page locations which prevent using the Amper-Monitor while writing on a text file. Monitor commands use locations $3D through $42, and so does DOS. I tried using the &"300.357 to do a hex dump into a text file, but DOS went wild and clobbered itself. Sorry, but I see no solution without changing DOS or recoding the entire monitor.

```
===============================================================================
DOCUMENT :AAL-8309:Articles:AmperMon.Poker.txt
===============================================================================
```


d∑I»768∑855-n∑D:∑I,D:∑'x†7685∑∑"300.357W†∑"380:12 34 56 78 9A BC DE
F0c®∑"FBE2Gv∞∑"300L 380.387∞¿∑ 169,11,141,246,3,169,3,141,247,3,96… ∑
201,34,208,70,32,177,0,160,0,177,184,201,0˘’∑
240,8,9,128,153,0,2,200,208,242,169,141. „∑
153,0,2,152,24,101,184,133,184,144,2,230\ Ì∑
185,32,199,255,32,167,255,132,52,160,23∑ ¯∑
136,48,23,217,204,255,208,248,192,21,240∑ ∑
8,32,190,255,164,52,76,52,3,32,197,255 ∑ 76,0,254,76,201,222

```
==============================================================================
DOCUMENT :AAL-8309:Articles:ASCII.80.Cols.txt
==============================================================================
```

80 Column ASCII Monitor Dump...........................Mike Dobe
O'fallon, IL

I have been trying out the monitor patches in the July issue of AAL for adding an ASCII display to the memory dump, and I have two problems with them. Because the routines place the characters directly into the Apple's screen memory, they do not work with my 80 column card. The same problem also arises when I want to send a dump to a printer. As a solution to this problem I present still another monitor patch for an ASCII display. My version is slightly longer than the others, but it still fits in the cassette tape portion of the monitor (just barely, I might add).

In order to take advantage of the 80 column display I first made the following patches to the monitor:

```
FDA6:0F
FDB0:0F
```

These changes allow the dump routine to print 16 values on each line, rather than the usual eight.

Since the characters have to be printed after the current line of the dump is finished, I need a place to buffer up to 16 characters. $BCDF, an unused area in DOS, serves this purpose. My routine buffers each byte before calling PRBYTE to display the hex value. If a particular byte will be the last one on that line of the dump, the patch calls PRBYTE to print the byte, then tabs to column 60 and displays the contents of the buffer. Upper and lower case characters are printed as they are, and control characters are replaced with blanks. (That's my style. As Bob said in July, choose your own favorite!)

Of course the following patch needs to be made to the dump code, to call my routine (this is the same as shown in the July article):

```
FDBE:C9 FC
```

The patch can be used with a 40 column display by ignoring the above patches to $FDA6 and $FDB0, and by making the following changes to my patch routine:

```
1140     AND #7
1200     EOR #7
1300     LDA #30
1420     CPX #8
```

This patch was tested on a Microtek Magnum 80 card, but it should work on other brands as well.

[  It also works fine with the STB80 card, and the Apple //e...Bill  ]

==================================================================
DOCUMENT :AAL-8309:Articles:BaseAddr.Calc.txt
==================================================================

Base Address Calculation...................Bob Sander-Cederlof

I believe that Steve Wozniak was the first to use the tricks in a microcomputer, back in 1976 and 1977. All of the other designs I recall either used the more expensive static RAM, or used a complex circuit to refresh dynamic RAM arrays. Steve's design allowed the use of dynamic RAM without any separate circuitry for refresh.

Dynamic RAM needs refreshing because each bit cell is really only a capacitor, and the charge runs out after a few milliseconds. By reading each bit and re-writing it every few milliseconds, the data in memory is maintained as long as you like. Each 16384-bit RAM-chip is organized in 128 rows by 128 columns of bytes, and the chips are designed so that merely addressing each row often enough will keep the bits fresh as a daisy. Steve hooked up the Apple so that the process of keeping data displayed on the screen also ran through all the row addresses.

His second trick was to keep the screen (and therefore the RAM) happy without stealing any time from the CPU. He did this by using alternate half cycles of the clock. The one-megahertz clock runs the 6502 every other half cycle, and the screen gets its whacks at memory in between.

What has all the above to do with an article titled "Base Address Calculation"? Well, I'm getting to that. In order to address each row often enough, Steve re-arranged the address bits in a rather complicated way. As the screen is refreshed, scan-line by scan-line, bytes are read from RAM in an order that assures every RAM row is accessed about every 2 milliseconds. [ For the exact details of this process, see Winston Gayler's "Apple II Circuit Description", pages 41-57. ]

All this boils down to a need to go through a complicated calculation to convert a display line number into a base address in RAM. The process is implemented for the text screen at $FBC1 in the monitor ROM; for the lo-res graphics screen at $F847 in the monitor ROM; for the hi-res graphics screen at $F417 in the Applesoft ROM.

If we represent the 8-bit value for the line number on the text screen as "000abcde", the base calculation computes the address in RAM for the first character on that line and stores the result in two bytes at $28 and $29 in the form "000001cd eabab000". The two bits "ab" may have values "00", "01", or "10" for lines 0-7, 8-15, and 16-23 respectively. The "abab000" part of the least significant byte of the base address represents "ab" times 40. Remember there are 40 characters on a line?

The hi-res base address calculation is more complicated, but it really the same thing. If we think of a text line as being made up of 8 hi-res lines, both calculations ARE the same. Except that the lo-res RAM starts at $400, and the hi-res starts at $2000. A hi-res line number runs from 0 through 191, or $00 - $BF. If we visualize it as "abcdefgh", the base address calculation merely re-arranges the bits to "001fghcd eabab000". Note that if we multiply the text line number by 8 and run it through the hi-res calculation we will get "001000cd eabab000" which is correct except for starting at $2000 rather than $400.

The hi-res calculation inside Applesoft takes 33 bytes and 61 cycles. Harry Cheung, who lives in Onitsha, Nigeria, wrote a letter to Call APPLE (page 70, July, 1983) to present his shorter, faster version. Harry did it in 25 bytes and only 46 cycles (one more byte and 6 more cycles if you count the RTS, but I didn't count an RTS in the Applesoft version). Here is Harry's code, with my comments.

I need to point out several things here. Harry used page zero locations $00 and $01 for the resulting base address. If you want to use his program with Applesoft, change them to $26 and $27. Harry save the line number temporarily in the Y-register. If the Y-register is already holding something important (it is in the Applesoft case), you can substitute PHA and PLA for the TAY and TYA above. Same number of bytes, but 3 cycles longer.

If you want REAL speed, and can spare a few more bytes, you need to pre-compute all the base addresses and store them in a table. Then you can use the line number as an index into the table and do a base address TRANSLATION in just a few cycles. For example, assume you store all the low-order bytes in a 192-byte table called LO.BASE, and similarly the high-order bytes at HI.BASE. If you get the line number in the Y-register, then you can convert the line number to a base address like this:

```
       LDA LO.BASE,Y
       STA $26
       LDA HI.BASE,Y
       STA $27
```

That takes 10 bytes of program, 384 bytes of table, and only 14 to 16 cycles. I say 14 to 16, because it depends on whether either or both of the two tables cross page boundaries. If they each are entirely within a memory page, 14 cycles.

Now here is a little piece of code I wrote to test out Harry's calculator. It runs through each of the 192 lines and prints out the line number, an equal sign, the base address, and a space for each line (all in hex).

The monitor address $FDD3 is not a labelled entry point, but I think it will probably stay consistent in future editions of the Apple ROMs. It saves whatever is in the A-register, prints "=", restores the A-register, and falls into $FDDA. The routine at $FDDA prints the contents of A in hex.

Just for fun I also wrote some new versions of the text base address calculator. One of them is shorter but takes more time, and the other is longer but takes less time. Oh well, can't win every race! Here are listings of them both, followed by a commented listing of the Applesoft hi-res calculator.

By the way, if you want to see the WHOLE thing...a commented listing of the entire Applesoft ROM, we have it on disk in format for the S-C Macro Assembler.

```
=============================================================================
DOCUMENT :AAL-8309:Articles:Break.Cat.txt
=============================================================================
```

New CATALOG Interrupt......................Col. Paul L. Shetler
                                                Tripler AMC, Hawaii

Most of the routines I've seen to terminate a CATALOG listing involve patching in a routine that checks for a particular key input and adding code to do different actions, like aborting or single-stepping the catalog list.  Here is a modification I came up with that requires only a small change and no additional code.

This is the section of DOS that handles a new line in the CATALOG display:

```
               1000        .OR $AE2C
               1010
AE2C- 4C 7F B3 1020        JMP $B37F    leave File Manager
AE2F- A9 8D    1030 NEWLN  LDA #$8D     carriage return
AE31- 20 ED FD 1040        JSR $FDED    MON.COUT
AE34- CE 9D B3 1050        DEC $B39D    line count
AE37- D0 08    1060        BNE .1
AE39- 20 0C FD 1070        JSR $FD0C    MON.RDKEY
AE3C- A9 15    1080        LDA #$15     count 21 lines
AE3E- 8D 9D B3 1090        STA $B39D    reset line count
AE41- 60       1100 .1     RTS
```

Line 1020 is really the end of the previous routine, but we're going to be borrowing it, so I'll show it here.  NEWLN is called every time the catalog list finishes a file name.

Notice that two bytes are wasted in lines 1030-1040.  Why do LDA #$8D, JSR $FDED, when JSR $FD8E does the same thing?  Two bytes may not sound like much, but in this case it's enough to work some magic!  Try replacing the above piece of DOS with this:

```
               1000        .OR $AE2C
               1010
AE2C- 4C 7F B3 1020 EXIT   JMP $B37F    leave File Manager
AE2F- 20 8E FD 1030 NEWLN  JSR $FD8E    MON.CROUT
AE32- CE 9D B3 1040        DEC $B39D    line count
AE35- D0 0A    1050        BNE .1       return if not done
AE37- 20 0C FD 1060        JSR $FD0C    get a keypress
AE3A- 29 17    1070        AND #$17     the magic number
AE3C- F0 EE    1080        BEQ EXIT     abort CATALOG
AE3E- 8D 9D B3 1090        STA $B39D    new line count
AE41- 60       1100 .1     RTS
```

Slipping in that AND #$17, BEQ EXIT, has several effects:

1. Space Bar or Back Arrow will terminate the listing.
2. Forward Arrow will advance the listing one page (just like normal.)

3. The "A" key will advance the listing one line.

And it all fits into the original space!  The other keys will have different effects, depending on the value left in the accumulator after AND #$17.  Most keys will advance the listing between 1-23 lines.

Try substituting other values for the $17 in line 1070.  Remember that the value of (Keypress AND Value) will be the new line count.  The catalog display will scroll up by that number of lines.  If the result is zero, the catalog display will end.  The maximum result is the same as the mask value, that is, 23 lines for a $17 mask.

[  My favorite mask value is $4F.  With that value SPACE still breaks the display, but now the numeral keys scroll up by the same number of lines, i.e., pressing the "1" key gives one more line, "2" shows two more names, and so on.  Also, the "O" (oh, not zero) key scrolls up by 79 lines, which usually means all the way to the end of the catalog....Bill  ]

```
==============================================================================
DOCUMENT :AAL-8309:Articles:Churchs.Quickie.txt
==============================================================================
```

Saving Source with Apple's Mini-Assembler.........Jim Church
Trumbull, CT

I have discovered a way to store source code, complete with comments, on disk files for the Apple mini-assembler (at $F666 in the Integer BASIC ROM or Language Card load). I use what I call "the world's best word processor", the one you get from S-C Software for $50. I create a text file that looks like this:

```
FP
CALL-151
C080
F666G
300:LDX #C0 ;START WITH "A"-1
 INX        ;LOOP COMES HERE
 TXA        ;CHAR TO PRINT
 JSR FDED   ;PRINT IT
 CPX #DA    ;STOP AFTER "Z"
 BCC 302    ;NOT THERE YET
 RTS        ;FINISHED!
FP
CALL768
```

Assuming I have Integer BASIC in my RAM card, EXECing the above text file assembles the code very nicely and even runs the program once! Note that the Mini-Assembler does allow comments following a ";".

```
===============================================================================
DOCUMENT :AAL-8309:Articles:Front.Page.txt
===============================================================================
```

$1.50

Volume 3 -- Issue 12                   September, 1983

In This Issue...



65C02 Notes

We now have a sample from Rockwell, and it shares the problem of not working in an older Apple.  It's running just fine in the //e, but it doesn't work in the ][+.  Rockwell's distributor says that regular delivery is now scheduled for November.  Sigh....

There's a bug in the 65C02 chips!  Among the new features are several new addressing modes for the BIT instruction, including BIT #immediate.

The BIT instruction actually does two operations:

1) It ANDs together the Accumulator and the specified memory byte, and sets the Zero flag according to the result.

2) It sets the Overflow and Negative flags to the values of bits 6 and 7 of the memory byte.

Well, the BIT #immediate instruction does not do step two; it only modifies the Zero flag.  The other new address modes for BIT behave correctly.  BIT #$40 sure would have come in handy for a SEV (SEt oVerflow flag) instruction.

As always, we'll keep you posted.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

===============================================================================
DOCUMENT :AAL-8309:Articles:Gen.Screen.Dump.txt
===============================================================================

Generic Screen Dump..............................Steve Knouse
Tomball, TX

Some computer terminals have a special key on the keyboard which will dump whatever is on the screen to a printer. The following program will give the same function to an Apple, using the ctrl-P key.

Many different versions of screen dump programs have been written, and published hither and yon. Most of them work with the particular author's printer and interface combination, but not mine or yours. I found the one Bob S-C published in the July 81 issue of AAL to be like that, so I worked it over. Now I believe it can truly be called "generic", or at least general, because it runs on every combination of printers and interfaces I can find.

I tested it on systems using the following interfaces:

    Epson APL
    Orange Micro Grappler, Grappler+, & Buffered Grappler+
    Practical Peripherals Microbuffer II
    SSM AIO II & ASIO
    Tymac Parallel
    Videx PSIO

The screen dump should work with any interface which recognizes the Apple standard method for turning off video output. The standard is to "print" a control-I followed by an "N". Lines 2190 through 2250 perform the output of these two characters.

The only board I found which did not work with this convention was the SSM AIO board, so the program which follows has a special conditional assembly mode to make it assembly slightly different object code for that board. If you have that board, change line 1610 to say "VERSION .EQ AIO" and it will assembly your version. Instead of Lines 2190 through 2250 being assembled, lines 2260 through 2310 will. They do not show up in the listing, so here they are:

```
2260    .DO VERSION=AIO
2270    LDA #$80
2280    JSR COUT
2290    LDX SLOT
2300    STA NOVID,X
2310    .FIN
```

If your assembler does not support conditional assembly, you can merely type in the lines 2270-2300 above in place of lines 2190-2310.

If your printer interface is not plugged into slot 1, change the slot number in line 2030, or at $0319.

Install the program by BRUNning the binary file of the object code, or by BLOADing it and doing a CALL768. Then whenever you type control-P, the screen will be printed. You can also call the screen dump from a running Applesoft program with CALL 794.

```
=============================================================================
DOCUMENT :AAL-8309:Articles:Jump.Vectoring.txt
=============================================================================
```

Jump Vectoring............................Bob Sander-Cederlof

Applesoft has a statement which allows branching according to a computed index:

```
       ON X GO TO 100,200,300,400
```

Integer BASIC has a different method, simply allowing the line number after a GOTO, THEN, or GOSUB to be a computed value:

```
       GO TO X*100
```

Most other languages have some technique for vectoring to one of a series of places based on the value of a variable.  Modern languages like Pascal have a CASE statement, which can combine a comparison step.

```
       case PIECE of
          Pawn : ...;
          Knight : ...;
          Bishop : ...;
          Rook : ...;
          Queen : ...;
          King : ...;
       end
```

I frequently find myself building various schemes to handle the CASE statement in assembly language.  For example, I might accept a character from the keyboard and then compare it to a series of legal inputs, and branch accordingly to process the input.

One common way involves a series of CMP BEQ pairs, like this:

```
       JSR GETCHAR
       CMP #$81        control-A?
       BEQ ...            yes
       CMP #$84        control-D?
       BEQ ...            yes
       CMP #$8D        return?
       BEQ ...            yes
       et cetera
```

If there are not too many cases, and if the processing routines are not too far away for the BEQs to reach, this is a good way to do the job.  If the routines are bigger, and therefore tend to be too far away (causing RANGE ERRORS at assembly time), I might string together CMP BNE pairs instead:

```
       JSR GETCHAR
```

```
       CMP #$81        control-A?
       BNE TRY.D       no, try ctrl-D

      <code to process ctrl-A here>

TRY.D  CMP #$84        control-D?
       BNE TRY.M       no, try return

      <code for ctrl-D here>
TRY.M  CMP #$8D        return?
       BNE ... et cetera

      <code for ctrl-M here>
```

The trouble with the latter way is that programs get strung all over the place, and become very difficult to follow. Unstructured, some would say. The structure is really there, because we are just implementing a CASE statement; however, assembly language code over a sheet of paper long LOOKS unstructured, no matter what it is implementing. And once a programmer gets his CASE statement spread over several sheets of paper, the temptation to begin making a "rat's nest" out of it can be overwhelming.

I prefer to put things into nice neat data tables. Back in the August 1982 issue of AAL I presented a "Search and Perform" subroutine to handle a table like this:

```
       .DA #$81,CTRL.A-1
       .DA #$84,CTRL.D-1
       .DA #$8D,RETURN-1
       etc.
```

The table consists of three bytes per line, the first byte being the CASE value, and the other two being the address of the processing routine.

Another method is handy when the variable has a nice numeric range. For example, what if I have processing routines for every possible control character from ctrl-A through ESC? That is ASCII codes $81 through $9B. If I subtract $81, I get a value from 0 through 26 (decimal). If I then multiply the value by three, and add it to a base address, and store the result into another variable, and JMP indirect, I can access a series of JMPs to each processing routine:

```
       JSR GETCHAR
CASE   SEC
       SBC #$81
       CMP #27
       BCS ...ERROR, NOT IN RANGE
       STA ADDR        TIMES THREE
       ASL
       ADC ADDR
       ADC #TABLE      PLUS TABLE BASE ADDRESS
       STA ADDR
```

```
       LDA #0
       ADC /TABLE
       STA ADDR+1
       JMP (ADDR)
ADDR   .BS 2
TABLE  JMP CTRL.A
       JMP CTRL.B
       .
       .
       JMP ESCAPE
```

Note that if we got to the CASE program by doing a JSR CASE, then each processing routine can do an RTS to return to the main line program. This makes our CASE look like it is doing a series of JSR's instead of JMP's.

We can shave bytes off the above technique by only keeping the address in TABLE, without all the JMP opcodes. Then the variable only needs to be multiplied by two instead of three. We will have to use the doubled variable for an index to pick up the address in the table and put it into ADDR:

```
       JSR GETCHAR
CASE   SEC
       SBC #$81
       CMP #27
       BCS ...ERROR, NOT IN RANGE
       ASL     DOUBLE THE INDEX
       TAX
       LDA TABLE,X
       STA ADDR
       LDA TABLE+1,X
       STA ADDR+1
       JMP (ADDR)
ADDR   .BS 2
TABLE  .DA CTRL.A
       .DA CTRL.B
       .
       .
       .DA ESCAPE
```

I don't recommend self-modifying code, but I still use it sometimes. If you want to save two more bytes above, then you can store the jump address directly into the second and third bytes on a direct JMP instruction:

```
       LDA TABLE,X
       STA ADDR+1
       LDA TABLE+1,X
       STA ADDR+2
ADDR   JMP 0
```

A much better way involves pushing the processing routine address onto the stack, and using an RTS to branch to the pushed address. Since

RTS adds 1 to the address on the stack before branching, we have to push the address-1:

```
       LDA TABLE+1,X
       PHA            HIGH BYTE FIRST
       LDA TABLE,X
       PHA
       RTS
TABLE  .DA CTRL.A-1
       .DA CTRL.B-1
       .
       .
       .DA ESCAPE-1
```

Note that this method not only is not self-modifying, it also is a few bytes shorter and a tad faster.

All this is only necessary because the designers of the 6502 did not give us a JMP (addr,X) instruction.  If they had, we could do it like this:

```
       JSR GETCHAR
CASE   SEC
       SBC #$81
       CMP #27
       BCS ...ERROR
       ASL            DOUBLE FOR INDEX
       TAX
       JMP (TABLE,X)
TABLE  .DA CTRL.A, CTRL.B,...,ESCAPE
```

Then the hardware would add the doubled character offset (0,2,4,...52 for ctrl-A thru ESC) to the base address of the table, pick up the address from the table, and jump to the corresponding processing routine.

Since that would be so nice, and the designers agreed, the new 65C02 chip has it!  So if you know you are writing for a 65C02, and don't EVER intend to run in a plain 6502, you can use the JMP (TABLE,X).

It would also be nice to have JSR (TABLE,X), but you can simulate that by calling CASE with a JSR.  Or in other situations, you might merely do it this way:

```
       JSR CALL
       .
       .
CALL   JMP (TABLE,X)
```

Sometimes it so happens that your program can be arranged so that all the processing routines are in the same memory page.  Then there is no need to store the high byte of the address in the table, right?  Steve Wozniak thought this way, and you can see the result in the Apple monitor at $FFBE and following:

```
TOSUB  LDA #$FE        HIGH BYTE OF ALL ADDRESSES
       PHA
       LDA SUBTBL,Y
       PHA
ZMODE  LDY #0
       STY MODE
       RTS
       .
       .
SUBTBL .DA #BASCONT-1  CTRL-C
       .DA #USR-1      CTRL-Y
       .DA #BEGZ-1     CTRL-E
       .
       .
       .DA #BLANK-1    BLANK
```

Steve also used this technique inside the SWEET-16 interpreter. You can see the code at $F69E through $F6C6 in the Integer BASIC ROM or RAM image.

If the routines are not necessarily all in one page, but are all within one 256-byte range, you can add an offset from the table to a known starting address.

Here is a method I would NEVER use, but it is cute, and short:

```
       LDA TABLE,X     X IS CALCULATED INDEX
       STA BRANCH+1    INTO BCC INSTRUCTION
       CLC             make branch always...
BRANCH BCC BRANCH      2ND BYTE GETS FILLED IN
BASE   .EQ *
       ...
       ...all the routines here
       ...
TABLE  .DA #CTRL.A-BASE
       .DA #CTRL.B-BASE
       etc.
```

The table has pre-computed relative offsets from BASE, so that the values can be plugged directly into the BCC instruction. This is a fast and short technique, but somehow it scares me to think about self-modifying code. If you need it, go ahead and use it!

```
=================================================================
DOCUMENT :AAL-8309:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler (the best there is!)..........................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50

S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00

S-C Word Processor (the one we use!)..............................$50.00
    With fully commented source code.  Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)................... $79.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).......$49.00

The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00

Blank Diskettes.................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Shift-Key Modifier................................................$15.00
Lower-Case Display Encoder ROM....................................$25.00
    Only Revision level 7 or later Apples.
STB-80 80-column Display Board (STB Systems)...........($249.00)  $225.00
STB-128 128K RAM Card (STB Systems)....................($399.00)  $350.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books.........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
```

```
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Micro Cookbook, vol. 2", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00
  Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8309:Articles:New.DOS33.Patch.txt
==============================================================================

Yet Another New Version of DOS 3.3.........Bob Sander-Cederlof

In the July issue of AAL I outlined the changes Apple made to DOS 3.3 early this year.  Today I received a new "Developer's System Master", with a cover letter claiming another correction to the APPEND routine. The letter binds developers to begin using the new version no later than November 1st.

If you like APPEND, or would like to like it, you might want to make these patches in your own system master.  I am going to assume you already have the "early 1983" version, either because you bought a //e or a disk drive this year, or you copied one from a friend, or you made the patches from my July article.  Here are the new changes:

```
"early 1983"                 August, 1983
---------------------       -----------------------
B683:4C 84 BA JMP $BA84     B683:4C B3 B6 JMP $B6B3

$B6B3-B6CE:ALL ZEROES       B6B3:AD BD B5 LDA $B5BD
                            B6B6:8D E6 B5 STA $B5E6
                            B6B9:8D EA B5 STA $B5EA
                            B6BC:AD BE B5 LDA $B5BE
                            B6BF:8D E7 B5 STA $B5E7
                            B6C2:8D EB B5 STA $B5EB
                            B6C5:8D E4 B5 STA $B5E4
                            B6C8:BA       TSX
                            B6C9:8E 9B B3 STX $B39B
                            B6CC:4C 7F B3 JMP $B37F

$BA84-BA93:PATCH            BA84-BA93:ALL ZEROES
```

What Apple has done is move the patch they had put at $BA84 down to $B6B3 and added four extra lines to that patch.  I HOPE IT IS NOW CORRECT!

==============================================================================
DOCUMENT :AAL-8309:Articles:QuickTrace.Load.txt
==============================================================================

Using QUICKTRACE with S-C Assembler..............Bob Urschel
Valparaiso, IN

I wanted to use QUICKTRACE in conjunction with the S-C Assembler without having QUICKTRACE interfere with either my source file or any object code generated. Since I always use the LC version of the assembler, I modified the HELLO program on the S-C assembler disk as follows:

```
10 HOME:PRINT "LOADING QUICKTRACE..."
20 POKE 40192,211:POKE40193,142:CALL42964
30 PRINT CHR$(4)"BLOAD QUICKTRACE,A$8F00"
40 PRINT:PRINT "LOADING S-C ASSEMBLER..."
50 VTAB24:POKE34,23:PRINTCHR$(4)"EXEC LOAD LCASM"
60 END
```

Line 20 in the HELLO program modifies the location of the DOS buffers by $E00 bytes to make room for the QUICKTRACE program. After running the HELLO program, when the S-C prompt appears and BEFORE loading any S-C source files, enter:

```
       :$8F00G  <return>
```

This initializes QUICKTRACE.

I also changed the address at MON$ (from within QUICKTRACE) to MON$=D003 so when I press M from single-step mode, I return to the S-C Assembler with my source file intact.

==============================================================================
DOCUMENT :AAL-8309:Articles:RENEWAL.PLEA.txt
==============================================================================

It is now three years since I began writing and publishing the Apple Assembly Line. Beginning small, and growing gradually, we now send out about 1400 copies each month to over 30 countries.

Many of our readers are also writers, both of text and of software. It is a distinct pleasure each month to recognize the names of so many authors of magazine articles and books as being our subscribers. Names like Roger Wagner, Tom Weishaar, Don Lancaster, Preston Black, Sandy Mossberg, Joe Devine, Jules Gilder, Al Tommervik, Val Golding, Roger Keating, Peter Weiglin, and I could go on and on. They all receive and read AAL, and we enjoy the occasional feedback from them as well.

And of software...our readers have produced Format II, The DOS Enhancer, Font Downloader, ES-CAPE, Cytron Masters, Cartels and Cutthroats, Flash!, The Routine Machine, Amper-Magic, ProntoDOS, The Visible Computer, firmware for numerous interface cards, Data Capture, Nibbles Away, and again I could go on and on. We don't take credit, be we sure enjoy the company!

By the way, we have noticed that your subscription ran out some time ago. We have missed you! If you are one of those who just forgot, perhaps you would like to be reminded that S-C Software and the Apple Assembly Line are still going strong. If you are using assembly language in your Apple, we believe the newsletter will help make you more proficient, and keep you up-to-date with new hardware, software, and books. Regardless of your skill level, it is easy to find at least one item out of twelve issues that pays for the subscription many times over.

Why not sign up again for another year? It is only $15 for twelve monthly issues, or you can have it by First Class Mail for only $3 more. If you want to pick up the back issues you missed (we have them all), they are only $1.50 each. Let us hear from you!

Sincerely,

Bob Sander-Cederlof

==========================================================================
DOCUMENT :AAL-8309:Articles:SAMPLE.txt
==========================================================================


```
d∑I»768∑855-n∑D:∑I,D:∑'x†7685∑∑"300.357W†∑"380:12 34 56 78 9A BC DE
F0c®∑"FBE2Gv∞∑"300L 380.387∞¿∑ 169,11,141,246,3,169,3,141,247,3,96… ∑
201,34,208,70,32,177,0,160,0,177,184,201,0˘’∑
240,8,9,128,153,0,2,200,208,242,169,141. „∑
153,0,2,152,24,101,184,133,184,144,2,230\_     Ì∑
185,32,199,255,32,167,255,132,52,160,23∑ ¯∑
136,48,23,217,204,255,208,248,192,21,240∑     ∑
8,32,190,255,164,52,76,52,3,32,197,255   ∑ 76,0,254,76,201,222
```

-----------------------------------------------------------------------------
DOCUMENT :AAL-8309:Articles:Spiral.Compiler.txt
-----------------------------------------------------------------------------

Generate Machine Code with Applesoft.......Bob Sander-Cederlof

Apparently nobody picked up my challenge at the end of the article about Charlie Putney's faster spiral screen clear program (August 1983 AAL, page 16). I suggested someone write a program in Applesoft which would in turn construct a machine language screen clear.

Nobody else did it, so I did. And whether you are interested in fancy ways to clear the screen or not, the techniques I used may be put to other uses.

The task of building a screen clear program can be divided into two parts. First, generate the memory addresses of the 960 cells on the screen, in the order (or path) that the spiral shift will follow. Second, using that table of addresses, generate the 959 pairs of LDA and STA instructions necessary to move the screen one position along the spiral path. There is really a third part: to generate the necessary prologue and postlogue instructions to make those 959 LDA-STA pairs be executed 960 times, and to clear the vacated byte at the tail end of the spiral path.

After trying various ways to understand the spiral path, I arrived at a table-driven approach. I put the table into data statements (lines 3000-3110 below), and made a simple loop to generate the 960 addresses (lines 100-150).

You might notice that the twelve lines of data correspond very closely to the parameters on Charlie Putney's macro calls. After I typed in the twelve lines, I noticed a definite pattern. I could have used only the first line of data, and computed the others by a simple algorithm: increment each value smaller than 13, and decrement each value 13 or larger. Well, no program is ever finished....

Once the 960 addresses are stored in array A%(0) through A%(959), I proceed to generate machine language code. Line 180 does it all, with the help of four simple subroutines. Then line 190 rings the bell, and line 200 calls the machine language program just generated for a fast two-and-a-half second demonstration.

During the address array building process, I fill up the screen with the letters U, D, L, and R. These show the direction (up, down, left, and right) which a given character will be shifted along the spiral path. The directions are just the opposite from the order in which the letters are displayed, because I generate the address list backwards (from head to tail).

During the generation of the machine language program, which takes about two minutes, I toggle the tail end character between normal to inverse video. This gives you something to watch for those lloooonnggg two minutes.

The generation process is broken into four parts, represented by four subroutines at 5000, 5100, 5200, and 5300.

GOSUB 5000 generates a four byte prologue, starting at memory address $2710, or 10000. The code looks like this:

```
       LDX /-960
       LDY #-960
```

Actually, not -960, but -960/S. S gives a step size. Sidestepping a little from the main discussion, let me tell you about S.

Don Lancaster called last week to talk about a few things with Bill, and passed on the results of his experiments with Charlie's program. He noted that the video refresh rate is 60 times per second, and that a 7.5 second screen clear moves a little more than two steps for each frame time. Therefore you don't really SEE each step. Therefore the screen clear routine could move each character two steps ahead at a time with the same smooth effect on the screen, but clearing the screen in half the time. Or three steps, clearing in one third the time. The variable S in my program lets you experiment with the number of steps each character moves during each pass. As listed, S=3, so the screen clears in 2.5 seconds.

GOSUB 5100 generates the requisite number of LDA-STA pairs to move the screen one step of size S along the spiral path.

GOSUB 5200 generates the instructions to clear the bytes at the tail end of the spiral. If S=3, you will get:

```
       LDA #$A0        BLANK
       STA $636
       STA $635
       STA $634
```

GOSUB 5300 generates the end-of-loop code:

```
       INY
       BNE LP
       INX
       BNE LP
       RTS
  LP   JMP 10004
```

The screen need not necessarily be cleared to all blanks. By changing the value POKEd in the second part of line 5210 you can fill with all stars, or all white, or whatever.

Another interesting option occurs to me. Given a table in the A% array of all the screen addresses, in any arrangement that suits my fancy, I can clear the screen in 2.5 to 7.5 seconds by shifting the screen along that particular path. It could be random, spiral, kaleidoscopic, or whatever.

There are so many other things I could explain about this little program, I hardly know where to stop. I think I'll stop here, and leave the rest for your own rewarding investigation and analysis.

```
==============================================================================
DOCUMENT :AAL-8309:DOS3.3:AmperMtr.Poker.txt
==============================================================================
```


d∑I»768∑855-n∑D:∑I,D:∑'x†7685∑∑"300.357W†∑"380:12 34 56 78 9A BC DE
F0c®∑"FBE2Gv∞∑"300L 380.387∞¿∑ 169,11,141,246,3,169,3,141,247,3,96… ∑
201,34,208,70,32,177,0,160,0,177,184,201,0˘’∑
240,8,9,128,153,0,2,200,208,242,169,141. „∑
153,0,2,152,24,101,184,133,184,144,2,230\ Ì∑
185,32,199,255,32,167,255,132,52,160,23∑ ¯∑
136,48,23,217,204,255,208,248,192,21,240∑ ∑
8,32,190,255,164,52,76,52,3,32,197,255 ∑ 76,0,254,76,201,222

```
==============================================================================
DOCUMENT :AAL-8309:DOS3.3:JOHNSONS.MACROS.txt
==============================================================================

 1000  *SAVE JOHNSON'S MACROS
 1010  *--------------------------------
 1020  *      DAVID JOHNSON'S MACROS
 1030  *      FOR THE FAST SHAPE TABLE
 1040  *      PROGRAM IN LATEST BYTE MAGAZINE
 1050  *--------------------------------
 1060         .MA Q
 1070  R      .SE 0
 1080         >R
 1090  Q      .SE Q+1
 1100         .DO Q<40
 1110         >Q
 1120         .FI
 1130         .EM
 1140         .MA R
 1150         .DA #Q,#R
 1160  R      .SE R+1
 1170         .DO R<7
 1180         >R
 1190         .FI
 1200         .EM
 1210  Q      .SE 0
 1220         >Q
```

=====================================================================
DOCUMENT :AAL-8309:DOS3.3:S.AMPER.MONITOR.txt
=====================================================================

```
 1000  *SAVE S.AMPER.MONITOR
 1010  *--------------------------------
 1020  *      &-MONITOR COMMANDS
 1030  *--------------------------------
 1040  MON.MODE         .EQ $31
 1050  MON.YSAV         .EQ $34
 1060  TXTPTR           .EQ $B8 AND B9
 1070  MON.BUFFER       .EQ $200
 1080  AMPERSAND.VECTOR .EQ $3F5
 1090  *--------------------------------
 1100  AS.CHRGET        .EQ $00B1
 1110  AS.SYNERR        .EQ $DEC9
 1120  MON.BL1          .EQ $FE00
 1130  MON.GETNUM       .EQ $FFA7
 1140  MON.TOSUB        .EQ $FFBE
 1150  MON.ZMODE        .EQ $FFC7
 1160  MON.CHRTBL       .EQ $FFCC
 1170  *--------------------------------
 1180         .OR $300
 1190  *--------------------------------
 1200  SETUP  LDA #AMPER.MONITOR
 1210         STA AMPERSAND.VECTOR+1
 1220         LDA /AMPER.MONITOR
 1230         STA AMPERSAND.VECTOR+2
 1240         RTS
 1250  *--------------------------------
 1260  AMPER.MONITOR
 1270         CMP #$22     MUST BE QUOTATION MARK HERE
 1280         BNE .6       SYNTAX ERROR
 1290         JSR AS.CHRGET
 1300         LDY #0
 1310  .1     LDA (TXTPTR),Y
 1320         BEQ .2
 1330         ORA #$80
 1340         STA MON.BUFFER,Y
 1350         INY
 1360         BNE .1
 1370  .2     LDA #$8D
 1380         STA MON.BUFFER,Y
 1390         TYA
 1400         CLC
 1410         ADC TXTPTR
 1420         STA TXTPTR
 1430         BCC .25
 1440         INC TXTPTR+1
 1450  .25    JSR MON.ZMODE
 1460  .3     JSR MON.GETNUM
 1470         STY MON.YSAV
 1480         LDY #23
```

```
1490  .4     DEY
1500         BMI .6       SYNTAX ERROR
1510         CMP MON.CHRTBL,Y
1520         BNE .4       NOT THIS ENTRY
1530         CPY #21
1540         BEQ .5       <RETURN> ALONE
1550         JSR MON.TOSUB
1560         LDY MON.YSAV
1570         JMP .3
1580  .5     JSR MON.ZMODE-2
1590         JMP MON.BL1
1600  .6     JMP AS.SYNERR
```

=====================================================================
DOCUMENT :AAL-8309:DOS3.3:S.CatalogInt.txt
=====================================================================

```
 1000  *SAVE S.CATALOG INTERRUPT
 1010  *--------------------------------
 1020  FMEXIT .EQ $B37F
 1030  COUNT  .EQ $B39D
 1040  RDKEY  .EQ $FD0C
 1050  CROUT  .EQ $FD8E
 1060  *--------------------------------
 1070         .OR $AE2C
 1080
 1090  EXIT   JMP FMEXIT   leave File Manager
 1100  NEWLN  JSR CROUT    send <CR>
 1110         DEC COUNT    line count
 1120         BNE .1       return if not done
 1130         JSR RDKEY    get a keypress
 1140         AND #$17     the magic number
 1150         BEQ EXIT     abort CATALOG
 1160         STA COUNT    new line count
 1170  .1     RTS
```

==============================================================================
DOCUMENT :AAL-8309:DOS3.3:S.FastShortHBC.txt
==============================================================================

```
 1000  *SAVE FAST & SHORT HBASCALC
 1010  *--------------------------------
 1020  *      DRIVER ROUTINE TO PRINT OUT
 1030  *      CALCULATED BASE ADDRESSES
 1040  *--------------------------------
 1050  TEST   LDX #0
 1060  .1     TXA
 1070         JSR CALC
 1080         TXA
 1090         JSR $FDDA
 1100         LDA 1
 1110         JSR $FDD3
 1120         LDA 0
 1130         JSR $FDDA
 1140         LDA #$A0
 1150         JSR $FDED
 1160         INX
 1170         CPX #192
 1180         BCC .1
 1190         RTS
 1200  *--------------------------------
 1210  *      BASE ADDRESS CALCULATOR
 1220  *      HARRY CHEUNG
 1230  *      PMB 1601, ONITSHA, NIGERIA
 1240  *      CALL APPLE, JULY 1983, PAGE 70
 1250  *--------------------------------
 1260  CALC   TAY          (TAY..TYA COULD BE PHA..PLA)
 1270         AND #$C7     ABCDEFGH
 1280         STA 0        AB000FGH
 1290         ORA #$08     FOR BASE = $2000, $10 FOR $4000
 1300         STA 1        AB001FGH
 1310         TYA          ABCDEFGH
 1320  *                CARRY..A-REG......$00.......$01...
 1330         ASL          A--BCDEFGH0  AB000FGH  AB001FGH
 1340         ASL          B--CDEFGH00     "         "
 1350         ROR 0        H--   "      BAB000FG     "
 1360         ASL          C--DEFGH000     "         "
 1370         ROL 1        A--   "         "      B001FGHC
 1380         ROR 0        G--   "      ABAB000G     "
 1390         ASL          D--EFGH0000
 1400         ROL 1        B--   "         "      001FGHCD
 1410         ASL          E--FGH00000     "         "
 1420         ROR 0        G--   "      EABAB000  001FGHCD
 1430         RTS
 1440  *--------------------------------
 1450  LRCALC.1
 1460         PHA
 1470         AND #$18     000DE000
 1480         ASL          00DE0000
```

```
1490        STA 0
1500        ASL          0DE00000
1510        ASL          DE000000
1520        ORA 0        DEDE0000
1530        STA 0
1540        PLA          000DEFGH
1550        LSR          0000DEFG
1560        ROR 0        HDEDE000
1570        AND #$03     000000FG
1580        ORA #$04     000001FG  (FOR PAGE 1)
1590        STA 1
1600        RTS
1610 *--------------------------------
1620 LRCALC.2
1630        PHA
1640        AND #$18     000DE000
1650        BEQ .1
1660        CMP #$10
1670        LDA #$A0
1680        BCS .1
1690        LSR
1700 .1     STA 0        DEDE0000
1710        PLA          000DEFGH
1720        LSR          0000DEFG
1730        ROR 0        HDEDE000
1740        AND #$03     000000FG
1750        ORA #$04     000001FG  (FOR PAGE 1)
1760        STA 1
1770        RTS
1780 *--------------------------------
1790 *      FROM APPLESOFT ROM AT $F417-$F437
1800 *--------------------------------
1810 MON.GBASL  .EQ $26
1820 MON.GBASH  .EQ $27
1830 HGR.PAGE   .EQ $E6
1840 AS.HRCALC
1850        PHA          Y-POS ALSO ON STACK
1860        AND #$C0     CALCULATE BASE ADDRESS FOR Y-POS
1870        STA MON.GBASL     FOR Y=ABCDEFGH
1880        LSR               GBASL=ABAB0000
1890        LSR
1900        ORA MON.GBASL
1910        STA MON.GBASL
1920        PLA           (C)   (A)     (GBASH)   (GBASL)
1930        STA MON.GBASH  ?-ABCDEFGH  ABCDEFGH  ABAB0000
1940        ASL            A-BCDEFGH0  ABCDEFGH  ABAB0000
1950        ASL            B-CDEFGH00  ABCDEFGH  ABAB0000
1960        ASL            C-DEFGH000  ABCDEFGH  ABAB0000
1970        ROL MON.GBASH  A-DEFGH000  BCDEFGHC  ABAB0000
1980        ASL            D-EFGH0000  BCDEFGHC  ABAB0000
1990        ROL MON.GBASH  B-EFGH0000  CDEFGHCD  ABAB0000
2000        ASL            E-FGH00000  CDEFGHCD  ABAB0000
2010        ROR MON.GBASL  0-FGH00000  CDEFGHCD  EABAB000
2020        LDA MON.GBASH  0-CDEFGHCD  CDEFGHCD  EABAB000
```

```
2030        AND #$1F       0-000FGHCD  CDEFGHCD  EABAB000
2040        ORA HGR.PAGE   0-PPPFGHCD  CDEFGHCD  EABAB000
2050        STA MON.GBASH  0-PPPFGHCD  PPPFGHCD  EABAB000
2060  *--------------------------------
2070        RTS
2080  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8309:DOS3.3:S.GenScreenDump.txt
==============================================================================

 1000  *SAVE GENERIC SCREEN DUMP
 1010  *--------------------------------
 1020  *
 1030  * GENERIC SCREEN DUMP
 1040  *
 1560  *--------------------------------
 1570
 1580  GENERIC    .EQ 1
 1590  AIO        .EQ 2
 1600
 1610  VERSION    .EQ GENERIC
 1620
 1630  CH         .EQ $24
 1640  BASL       .EQ $28
 1650  CSWL       .EQ $36
 1660  CSWH       .EQ CSWL+1
 1670  KSWL       .EQ $38
 1680  KSWH       .EQ KSWL+1
 1690
 1700  DOS.HOOK   .EQ $3EA
 1710
 1720  BASCALC    .EQ $FBC1
 1730  COUT       .EQ $FDED
 1740  KEYIN      .EQ $FD1B
 1750  RDKEY      .EQ $FD0C
 1760  OUTPORT    .EQ $FE95
 1770  VTAB       .EQ $FC22
 1780
 1790  CR         .EQ $8D      CARRIAGE RETURN
 1800  NOVID      .EQ $578
 1810  *--------------------------------
 1820         .OR $300
 1890  *--------------------------------
 1900  START  LDA #ENTRY   HOOK ROUTINE INTO DOS
 1910         STA KSWL
 1920         LDA /ENTRY
 1930         STA KSWH
 1940         JMP DOS.HOOK
 1950  *--------------------------------
 1960  ENTRY  JSR KEYIN    WAIT FOR A KEYPRESS
 1970         CMP #$90     ^P ?
 1980         BNE .1       NO
 1990         JSR DUMP     YES
 2000         JMP RDKEY
 2010  .1     RTS
 2020  *--------------------------------
 2030  SLOT   .DA #1
 2040  *--------------------------------
 2050  DUMP   PHA          SAVE A, X, Y
```

```
2060        TXA
2070        PHA
2080        TAY
2090        PHA
2100        LDA CH       SAVE CH
2110        PHA
2120        LDA CSWL     SAVE OUTPUT HOOKS
2130        PHA
2140        LDA CSWH
2150        PHA
2160 *
2170        LDA SLOT     COLD START BOARD
2180        JSR OUTPORT    IN SLOT 1
2190        .DO VERSION=GENERIC
2200        LDA #$89     KILL VIDEO ECHO
2210        JSR COUT
2220        LDA #"N"
2230        JSR COUT
2240        NOP          PAD TO STAY ALIGNED W/ AIO VERSION
2250        .FIN
2260        .DO VERSION=AIO
2270        LDA #$80     KILL VIDEO ECHO
2280        JSR COUT
2290        LDX SLOT
2300        STA NOVID,X
2310        .FIN
2320 *
2330        LDA #CR      START ON A NEW LINE
2340        JSR COUT
2350 *
2360        LDX #0       START W/ 1ST LINE (0TH)
2370        STX CH     SET CH TO 0 SO PRINTER WON'T INDENT
2380
2390 .1     TXA          LINE LOOP
2400        JSR BASCALC  GET ADDR OF LINE
2410        LDY #0       START W/ 1ST CHARACTER (0TH)
2420 .2     LDA (BASL),Y GET A CHAR
2430 .3     CMP #$A0     CONVERT FLASH/INVERSE CHAR
2440        BCS .4         NON-FLASHING U.C.
2450        ADC #$40
2460        BNE .3       ..ALWAYS
2470 .4     AND #$7F     MASK OFF HI BIT TO AVOID
2480 *                      EPSON BLOCK GRAPHICS
2490        JSR COUT     PRINT IT
2500        INY          LOOP FOR ANOTHER CHAR
2510        CPY #40
2520        BCC .2
2530        LDA #CR      END OF LINE
2540        JSR COUT
2550        INX          LOOP FOR ANOTHER LINE
2560        CPX #24
2570        BCC .1
2580
2590        PLA          RESTORE OUTPUT HOOKS
```

```
2600        STA CSWH
2610        PLA
2620        STA CSWL
2630        PLA          RESTORE CH
2640        STA CH
2650        JSR VTAB       AND LINE
2660        PLA          RESTORE Y, X, A
2670        TAY
2680        PLA
2690        TAX
2700        PLA
2710        RTS          ..THAT'S ALL FOLKS
2720  *
```

==============================================================================
DOCUMENT :AAL-8309:DOS3.3:S.Mon.ASC.DOBE.txt
==============================================================================

```
 1000  *SAVE S.MON ASCII DISPLAY (DOBE)
 1010  *--------------------------------
 1020  CH     .EQ $24
 1030  A1L    .EQ $3C
 1040  A1H    .EQ $3D
 1050  A2L    .EQ $3E
 1060  A2H    .EQ $3F
 1070  BUFFER .EQ $BCDF
 1080  PRBYTE .EQ $FDDA
 1090  COUT   .EQ $FDED
 1100  *--------------------------------
 1110         .OR $FCC9
 1120         .TA $CC9
 1130
 1140  PATCH  PHA           save byte
 1150         LDA A1L       low byte of dump address
 1160         AND #$F         is transformed to
 1170         TAX             offset in buffer
 1180         PLA           get original byte back
 1190         PHA             but keep it on the stack
 1200         STA BUFFER,X buffer the character
 1210         CPX #$F       last byte of line?
 1220         BEQ .0        if so, print the buffer
 1230         LDA A2L
 1240         CMP A1L       done with range?
 1250         BNE .3        return to monitor if not
 1260         LDA A2H
 1270         CMP A1H       check high bytes
 1280         BNE .3        return if more
 1290
 1300  .0     PLA
 1310         JSR PRBYTE    print the last byte
 1320         LDA #60       tab to column 60
 1330         STA CH
 1340         LDX #0
 1350  .1     LDA BUFFER,X display the buffer
 1360         ORA #$80
 1370         CMP #$A0      control character?
 1380         BCS .2
 1390         LDA #$A0      if so, substitute blank
 1400  .2     JSR COUT      print the character
 1410         LDA #$A0
 1420         STA BUFFER,X blank out buffer as we go
 1430         INX
 1440         CPX #$10      done?
 1450         BCC .1        no, go on
 1460         RTS
 1470
 1480  .3     PLA           restore original byte
```

```
1490        JMP PRBYTE   returns to caller
```

==============================================================================
DOCUMENT :AAL-8309:DOS3.3:Spiral.Scr.Addr.txt
==============================================================================

( DTC removed -- lots of garbage characters )

```
===============================================================================
DOCUMENT :AAL-8310:Articles:AAL.AUTHORS.txt
===============================================================================
```

Alphabetical Listing of Authors

Anders, Greg
Barkovitch, Dave
Bartley, David
Bartlett, Peter C. Jr.
Bernard, Robert H.
Black, Preston
Boering, Brooke
Bragner, Robert
Brightwell, Anthony
Broderick, John
Church, Jim
Collins, Bill
Deen, Bobby
Dobe, Mike
Fabbri, Richard
Greenfarb, Sanford
Hatcher, Rick
Hirai, Frank
Johnson, David C.
Kassel, Jim
Keating, Roger
Knouse, Steve
Kriegsmann, Mark
Lancaster, Don
Laumer, Mike
Linn, Bill
Mann, Steve
Marsalis, Allen
Matzinger, Bob
McKinstry, Herbert A.
McKinstry, Herbert L.
Meador, Lee
Meyer, Peter
Morgan, Bill
Mossberg, Sandy
Nacon, Bob
O'Brien, R. F.
Parker, Bill
Perkins, Bob
Pitz, Louis
Pote, Dan
Potts, Bob
Putney, Charles
Sander-Cederlof, Bob
Sanders, Mike
Savoie, William R.
Schlichtman, Ulf

Schlyter, Paul
Schneider, Horst
Schumer, Art
Shafer, Tracy L.
Shetler, Col. Paul L.
Steiner, Robert B.
Taylor, Don
Urschel, Bob
Weishaar, Tom
Welman, Chuck
Wetzel, Jim
Wiggs, Chris

```
================================================================
DOCUMENT :AAL-8310:Articles:Adv.v1.v3.txt
================================================================

E
Index to Advertising

Advanced Peripheral Enterprises, Inc.
  PRAWM Board       83: 4/23,5/7
Anthro-Digital
  Amper-Magic       82: 4/14,5/23,7/17,8/26,9/10,10/10,11/21,12/31
                    83: 1/6,2/24,3/23
  QuickTrace        82: 4/18,5/21,7/14,8/8,9/8,10/26,11/13,12/24
                    83: 1/18,2/6,3/6,4/21,5/11,6/10,7/27,8/25,9/13
Applied Engineering
  A/D System        81: 7/15,8/10,9/15,10/5,11/8
                    82: 1/13,2/10
  General           81: 12/10
                    82: 8/12,9/6,10/3,11/27,11/9
                    83: 4/11,5/27,6/18,7/19,8/18,9/9
  Time II           81: 7/12,8/9,9/2,10/10,11/13
                    82: 3/14,4/3,5/15,6/8,7/27
  Music Syntheszr   81: 8/15,9/7,10/7
                    82: 3/23,4/15,5/14,6/27,7/24
Arrow Micro Software
  DFX               82: 7/29
  Reflexive VC      82: 7/29
Aurora Systems
  QuickTrace        81: 12(insert)
                    82: 3/6
  Amper-Magic       82: 3/2
Axlon (RAM-disk)    82: 7/23,8/17,9/31,10/27,11/15,12/27
Broderick & Assoc.
  John's Debugger   81: 8/8,10/13
                    82: 4/27,5/5
  B.I.S.            82: 5/9
Castle Designs (Bill Goodwill)
  Fastdraw 1.1      83: 4/9,5/8
Church, James O.
  Super Phone       82: 4/20,6/22
Computer Data Services
  DOSSOURCE 3.3     81: 10/17
Computer Micro Works, Inc.
  3 Products        82: 12/11
  Disk Switch       83: 2/15
  PROM Switch       83: 5/26
Crow Ridge Associates
  Apple Flasher     82: 7/26
Cut The Bull Software
  Other Epson Man   82: 7/16
Decision Systems    80: 12/10
                    81:
1/7,2/10,3/6,4/9,5/11,6/9,7/8,8/5,9/10,10/11,11/18,12/15
```

```
                   82:
1/10,2/5,3/16,4/16,5/16,6/15,7/5,8/15,9/4,10/20,11/18,12/12
Douglas Electronics
  Appleseed System 83: 8/15,9/25
ESD Labs Excel-9   81: 12/19
FM Panatronics
  Serial Card      82: 9/22,10/7
Golden Delicious Software
  CIA              83: 7/14,8/7
GSR Associates
  EPSON Graphics   82: 2/14,5/6,6/6
  Clone Kit        82: 2/14,5/6,6/6
Laumer Research
  FLASH!           82: 1/24,2/13,3/8,5/13
  Full Screen Edit 83: 3/9,4/19,5/15,6/31,7/17,8/16
Lee Meador
  Disassembler     81: 2/16
Linn Software
  AED II           82: 4/23,5/19
Martcomm, Inc.
  EPROM Plug       82: 9/19,10/31
Micro Application  81: 12/22
Micro Mart         82: 3/12,5/30,6/10
Micromation
  Hero Robot Stuff 83: 9/7
Missouri Indexing  82: 1/17
Omega Microware, Inc.
  The Inspector    83: 4/25,5/19,6/15
  Watson           83: 4/27,5/21,6/13
RAK-WARE
  Disasm           80: 12/15
                   81: 1/9,3/5,5/2,7/11,8/12,9/13,12/23
                   82: 1/4,2/17
  TAB, XREF, GSR   81: 11/15,12/21
                   82: 1/5,2/12
  XREF             81: 4/7
  XREF & GSR       81: 5/4,6/11,10/3
  MX80 Formatter   82: 3/15,4/4,5/11
  Disasm, Utilites 82: 3/18,4/2,5/25,9/14,10/18,11/7,12/28
                   83: 1/27,2/13,3/8,4/24
  Disasm, Util, Mirror
                   82: 6/23,7/31,8/6
  Performer Board  82: 6/24,7/20,8/10
  RAM/ROM,Performer, and Mirror
                   82: 9/18,10/28,11/20,12/8
                   83: 1/14,2/27,3/25,4/16
  Font Downloader  83: 5/18,6/23,7/29,8/10,9/17
Scientific Software Products
  Amper-Ware       82: 9/16,10/14
S-C Software
  S-C Macro Asm    83: 6/14,7/22
  Cross Assemblers 82: 8/1
                   83: 1/16,2/10,3/18,5/16,6/29,7/30,8/27
  ES-CAPE          82: 7/1
```

```
                   83: 6/28,7/12,8/23,9/24
  General          81: 12/5
                   82: 1/7,2/3,8/3,11/82,12/3
                   83: 1/3,2/3,3/3,4/3,5/3,6/3,7/3,8/3,9/3
  NEC Printers     82: 5/31
  FLASH! special   82: 7/32
S&H Software
  Amper Sort/Merge 82: 7/21,8/30,9/28,10/19,11/5,12/21
  DOS Enhancer     82: 9/28,10/19,11/5,12/21
  UBI 4.0          82: 4/12,7/21,8/30
Softkey Publishing
  Diskedit         82: 6/20
  HARDCORE Mag     82: 6/18,8/19,9/20,10/23,11/23,12/18
                   83: 1/24,3/11,4/15
  HyperDOS         82: 9/26,10/29
Software Systems Support
  Furniture        82: 4/28,5/32,6/28
Soph-Key           83: 1/22,2/20
Southwestern Data Systems
  Write Now        81: 10/4,12/18
                   82: 2/21,4/21,6/11,8/22,10/16
  Routine Machine  82: 9/24,11/10,12/6
Tau Lambda
  SeaFORTH         82: 1/6,2/22
United Computer    83: 4/5,5/5,6/5,7/5
Vagabondo Enterprises
  CEEMAC           82: 1/14,7/8-9
Welman, C. J.      81: 2/13
```

================================================================
DOCUMENT :AAL-8310:Articles:Asm.From.400.txt
================================================================

Assembling in RAM from $400-$9AA6.............Robert F. O'Brien
Dublin, Ireland

I liked the procedure for getting listings into a text file during assembly (AAL July '83). However, it won't work if the file is too large and requires .IN directives. I recently did a large assembly using the following source code:

```
0       .DU
1       .TF LISTING
2       .IN PART1
3       .IN PART2
4       .ED
```

What I expected to get was a 356-sector text file on disk, but all I got was a 2-sector file -- the code for PART1 and PART2 was not sent to the disk (they did list to the screen!) I solved my particular problem by making more of RAM available for the assembly as follows:

a) Issue MAXFILES 1
b) LOAD/MERGE PART1 & PART2 source code in RAM and add .DU, .TF LISTING, and .ED lines
c) $C083 C083 N EAF9:0 N D021:4_ N D003G
d) ASM

By letting the symbol table build from $400 up we make the space between $400-$9AA6 available for assembly use (using the RAM card version of the assembler). I use the VIDEX version with a Peanut 80-column card and I find the screen gets messed up and a re-boot is necessary after assembly, but that's very little bother for the benefit received. It would be better of course to be able to use .IN directives.

[ This is a neat trick, but boy, does it scare me! Use this technique only with a backed-up disk and be very certain that the monitor isn't going to try to scroll the Apple's screen during assembly, because that will scramble the Symbol Table and leave you who-knows-where! The 80-column card (and any other peripherals that aree on) gets scrambled because of the slot-reserved variables just off the edge of the screen memory. Also note that you can't start all the way down at $400 if you are using macros with private labels, since that table grows downward from the base of the Symbol Table. ....Bill ]

```
================================================================================
DOCUMENT :AAL-8310:Articles:Avoid.Extra.Def.txt
================================================================================
```

Avoiding EXTRA DEFINITION ERROR.....................Bill Morgan

No sooner said...

OK, here are some patches to defeat the check for double definitions in the S-C Macro Assemblers.  Just put an RTS ($60) at the appropriate location:

```
Version 1.0 -- Motherboard: $221D
             Language Card: $E369

Version 1.1 -- Motherboard: $210E
             Language Card: $E228
```

Be very certain that any double definitions are intentional and identical.  If you use the same label with two different values (unless it's defined with .SE) the assembler cannot produce correct code.

=============================================================================
DOCUMENT :AAL-8310:Articles:Front.Page.txt
=============================================================================

$1.50

Volume 4 -- Issue 1 October, 1983

In This Issue...



Index to Apple Assembly Line

Why haven't we ever published an index to AAL?, you ask. Now that there are three year's worth of back issues to dig through for that article you know you saw a while back, wouldn't a true index come in handy? Well here it is! The 12 center pages of this issue are a complete index to volumes 1 through 3 of Apple Assembly Line. That's October, 1980 through September, 1983, all at your fingertips. The index is placed in the center of this issue so that, if you wish, you can easily remove those pages and store them separately.

More Applesoft Variable Cross Reference

In this issue Louis Pitz presents us with still more tinkering with the old Applesoft Variable Cross Reference. Now that the program has been modified a couple of times, and since it appeared way back in the second issue of Apple Assembly Line, we'll include the complete source code, including all of Louis' enhancements, on the next Quarterly Disk. Remember that all of the back issues are still available, if you don't have Volume 1, Number 2.

New Basis Version 1.1 Available

If any of you are using the S-C Macro Assembler with a Basis 108 computer, Bob Matzinger has adapted version 1.1 for you. Call us for the upgrade price. (214) 324-2050.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8310:Articles:Generic.Correx.txt
==============================================================================

Corrections to the Generic Screen Dump

Steve Knouse called to thank us for printing his Generic Screen Dump program last month, and to chew us out for garbling it.

It seems that we edited and renumbered the code, but didn't update the line number references in the text.

Here's a table to translate what the article says into what it means:

| Says | Means |
|---|---|
| 1610 | 1100 |
| 2030 | 1460 |
| 2190 | 1620 |
| 2250 | 1680 |
| 2260 | 1690 |
| 2270 | 1700 |
| 2280 | 1710 |
| 2290 | 1720 |
| 2300 | 1730 |
| 2310 | 1740 |

Sorry about that, readers.  Sorry about that, Steve.

[  And another last-minute correction -- the TAY instruction in line 1510 should be a TYA.  ]

```
=============================================================================
DOCUMENT :AAL-8310:Articles:Index.AAAA.GGGG.txt
=============================================================================
```

EIndex to Articles in "Apple Assembly Line", Volumes 1-3

```
================================================================
DOCUMENT :AAL-8310:Articles:Index.HHHH.End.txt
================================================================
```

TTTT
Techniques



Techniques, contd.

```
=========================================================================
DOCUMENT :AAL-8310:Articles:Index.Page.nums.txt
=========================================================================
```

================================================================
DOCUMENT :AAL-8310:Articles:Knouse.Mtr.txt
================================================================

A Compilation of Monitor Modifications.............Steve Knouse

Over the years since I bought my Apple I have been collecting various handy modifications to the Apple Monitor. I wanted a convenient way to load up all my patches so that they would be there when I needed them.

Let me point out right now that the following set of patches will NOT work in an Apple //e. They are only for the Apple II Plus monitor. Anyway, several of my favorite patches are already implemented in the Apple //e; the others may fit, but I haven't tried them.

There are two basic ways to get a modified monitor into an Apple. The first requires burning an EPROM with the new version, modifying the motherboard to accept an EPROM in the F8 ROM socket, and plugging it in. (Rather than cutting and splicing the motherboard, a better way is to use a PROMETTE from Computer Micro Works.) The second way is to run out of a language card (16K RAM Card), with a modified monitor at F800 in the RAM card. Some RAM cards may not allow this, leaving the motherboard F8 ROM always switched on, but all the ones I have tried work. If you want to use Applesoft with the modified monitor, or patch Applesoft as well, you can copy it up into the language card too.

I combined my favorite patches with Bill Morgan's patch program (see "PATCHER: General Purpose Patch Installer", AAL, April, 1983) so that BRUNning the program copies the motherboard monitor into a RAM card and then installs all the patches.

The listing that follows uses the .PH and .EP directives found in Version 1.1 of the S-C Macro Assembler. .PH starts a phase, and .EP ends one. At the start of a phase the current assembler origin is saved and the address from the .PH is substituted. Code continues to be assembled into the target file or at the target address, and the saved origin is incremented along with the phase origin. At the end of the phase the saved origin is restored. This allows me to assemble a series of patches with the correct addresses all into one big target file.

Here is a list of my favorite patches:

1  Allow lowercase input -- nullify conversion of lowercase to uppercase, make cursor over lowercase character to uppercase inverse (since Apple doesn't have inverse or flashing lowercase). (From Videx Keyboard Enhancer II Manual, page 4.)

2  Non-flashing cursor -- Make cursor inverse instead of flashing. (From Videx Keyboard Enhancer II Manual, page 4.)

3  Inverse + cursor when in escape mode -- to indicate IJKM is active. (By Donald W. Miller, Jr., Call-APPLE Mar 83 pp 51-52.)

4  ASCII dump -- display both hex and ASCII values.  (By Peter Bartlett, AAL Dec 81 pp 18-20, and Bruce Field, AAL Jul 83 page 20.)

5  Mask -- XXYY<ADR1.ADR2W masks bytes in memory range, ANDing with XX and ORing with YY.  (By Bob Sander-Cederlof, AAL Dec 82 pp 10-11.)

6  Search -- XXYY<ADR1.ADR2S searches memory range for XXYY, printing addresses of matches.  If XXYY is in the range $00-$FF, only one byte will be compared; otherwise both bytes will be compared during the search.  (By Steve Knouse)

I included several conditional assembly options, using the .DO, .ELSE, and .FIN directives.  These let you select or reject the non-flashing cursor patch and the lowercase display patch.  The third option allows you to copy Applesoft from the motherboard along with the monitor, or just the monitor by itself.

==============================================================================
DOCUMENT :AAL-8310:Articles:Large.Asm.Text.txt
==============================================================================

Large Assembly Listing into Text File.........Robert F. O'Brien
Dublin, Ireland

I liked the procedure for getting listings into a text file during assembly (AAL July '83). However, it won't work if the file is too large and requires .IN directives. I recently did a large assembly using the following source code:

```
0       .DU
1       .TF LISTING
2       .IN PART1
3       .IN PART2
4       .ED
```

What I expected to get was a 356-sector text file on disk, but all I got was a 2-sector file -- the code for PART1 and PART2 was not sent to the disk (they did list to the screen!)

I first tried to solve my particular problem by making more RAM available for the assembly by moving the Symbol Table base down to $400. I thought that should work, since I use an 80-column card and not the Apple's text screen. However, the assembler and the system monitor had other ideas, and promptly destroyed the symbol table by scrolling the screen memory.

However, I did manage to get my large assembly listing to go to disk as a text file -- by doing it in two parts. I used a utility program from the assembler disk to give each part the missing label definitions from the other part.

The steps are as follows:

1) Assemble the code normally with .IN directives.

```
0001        .IN PART1
0002        .IN PART2
```

2) BRUN B.MAKE EQUATE FILE (from the S-C Macro Version 1.1 Disk.) That creates a file of .EQ statements called SYMBOLS which contains all the normal labels and values from the Symbol Table in memory.

3) Merge SYMBOLS with PART1 and delete all duplicate labels from the SYMBOLS section.

4) Assemble PART1 using the .DU-.TF-.ED trick, and using .LIST OFF/ON so that the SYMBOLS section does not write to the text file.

5) Repeat steps 3 and 4 on PART2.

It is a bit laborious deleting all the duplicate labels in the two assemblies.  I hope someone can suggest a patch to the assembler to prevent it from reporting "EXTRA DEFINITION ERROR".  That certainly would make this listing process easier.

```
==============================================================================
DOCUMENT :AAL-8310:Articles:LC.Titles.txt
==============================================================================
```

Lower Case Titles in Version 1.1..................Bob Matzinger

A simple one-byte patch will enable you to use lower-case letters inside .TI titles. There are eight versions of the assembler on the Version 1.1 release disk, and the byte to be changed is in a different place for each version.

The code for the .TI directive looks the same wherever it is located. Here is a hex dump of the code, with a square around the byte to be changed:

```
       A2 00     LDX #0
       20 3E x2  JSR $123E or $D23E
       C9 2C     CMP #$2C
       D0 0D     BNE ...

       20 3E x2  JSR $123E or $D23E

       B0 08     BCS ...
       9D 70 01  STA $170,X
```

The following table shows the address of the byte to be changed:

| File Name | x = 1000 | x = D000 |
|---|---|---|
| S-C.ASM.MACRO.x | $2CE6 | $EE00 |
| S-C.ASM.MACRO.x.E | $2CC2 | $EDDC |
| S-C.ASM.MACRO.x.STB80 | $2DDA | $EEFD |
| S-C.ASM.MACRO.x.VIDEX | $2DB1 | $EED4 |

Once you find the right byte, which contains $3E, change it to $4E. (Remember to change a byte in the RAM card you need to write-enable it first.)

```
==================================================================
DOCUMENT :AAL-8310:Articles:Line.Counter.txt
==================================================================
```

**Counting Lines......................................Bill Morgan**

When Bob and I were first looking at Bruce Love's version of the Spiral Screen Clear, we got to wondering just how many lines actually were being processed by the assembler. With all those nested recursive macros, the total was bound to be in the thousands. Here's a little filter program I threw together to do a count:

```
0000-          1000 COUNT.LO .EQ 0
0001-          1010 COUNT.HI .EQ 1
0036-          1020 OUTHOOK  .EQ $36
03EA-          1030 DOSHOOK  .EQ $3EA
               1040 *---------------------------------
               1050        .OR $300
               1060
0300- A9 00    1070        LDA #0
0302- 85 00    1080        STA COUNT.LO      zero the counters
0304- 85 01    1090        STA COUNT.HI
0306- A9 11    1100        LDA #LINE.COUNTER
0308- 85 36    1110        STA OUTHOOK       direct output
030A- A9 03    1120        LDA /LINE.COUNTER to my routine
030C- 85 37    1130        STA OUTHOOK+1
030E- 4C EA 03 1140        JMP DOSHOOK
               1150 *---------------------------------
               1160 LINE.COUNTER
0311- C9 8D    1170        CMP #$8D          carriage return?
0313- D0 06    1180        BNE .1            no, exit
0315- E6 00    1190        INC COUNT.LO      yes, count it
0317- D0 02    1200        BNE .1
0319- E6 01    1210        INC COUNT.HI
031B- 60       1220 .1     RTS
```

I assembled that code at $300, and then used these commands to set the PRT vector:

```
:$C083 C083 D009:4C 0 3 N C080
```

(For the motherboard versions of the S-C Assemblers, you only need to type :$1009:4C 0 3)

With that in place just load a source file, set .LIST ON, type PRT, and then type ASM. When the assembly is finished, type PR#0 to get the output back to the screen. Now you can type :$0.1 to look at the counters. You might also want to put a .LIST OFF line at the end of your program, so the count won't include the Symbol Table.

By the way, when the macros are expanded those 80 lines of Bruce's program produce 13,593 lines of code, or enough to fill over 200 pages of printout.

==============================================================================
DOCUMENT :AAL-8310:Articles:Loves.Spiral.txt
==============================================================================

Macro-Calculated Spiral Screen Clear..............Bruce V. Love
Hamilton, New Zealand

Here is what I think is a beautiful example of using nested recursive macros with the new .SE directive to calculate the addresses for a Spiral Screen Clear.

The macro SPIRAL calls, in order, LEFT, BOTTOM, RIGHT, and TOP to produce the code to handle each side of the screen. Each of those macros adjusts the appropriate X or Y coordinate and then calls GETADR to calculate the addresses and actually assemble the next instruction pair.

This program won't win any prizes for fast assembly: I timed it at almost 4 minutes. You could speed up the process by rewriting the BOTTOM and TOP macros. They really don't have to call GETADR for all the calculation, they only need to increment or decrement the addresses, but that destroys the symmetry of the original.

I have also produced a faster version of the program. This one uses self-modifying code to avoid shifting the already-cleared bytes on the screen. It's interesting to watch the self-modifying version accelerate as it moves fewer bytes each time through the loop. To produce the faster version, just replace the code from line 1680 on with this new code:

```
==============================================================================
DOCUMENT :AAL-8310:Articles:More.VCR.Tinker.txt
==============================================================================
```

Still More Tinkering with VCR........................Louis Pitz
De Witt, Iowa

I finally figured out how to modify the Applesoft Variable Cross Reference (from the November, 1980 AAL) to distinguish between defined functions and array variables. As Bob mentioned at that time, VCR tags an occurence of FN AB(whatever) as an appearance of the array variable AB().

It turns out that the changes needed aren't many, and are compatible with my tinkering in the August '83 AAL, which added 80-column output to a printer.

As VCR is scanning for variables, in the GET.NEXT.VARIABLE section, add the check for the FN token in lines 2132-2134. If found, go to lines 2222-2228 to set a flag and go back to get the NEXT.CHAR.NOT.QUOTE. Unless the Applesoft program is in error, a variable name immediately follows the FN token.

In PACK.VARIABLE.NAME, the program distinguishes variables by VARNAM+2 having a space, $, or %. Array variables have the high bit set. In lines 2791-2796 I set apart FN variables by placing a dash (-) with the high bit set in VARNAM+2. This will make FN types come after the others alphabetically.

Now we come to the printing stage, in PRINT.LETTER.CHAIN. There the variable name (and dash, in case of FN types) is printed. If the high bit of VARNAM+2 is set, lines 4292-4294 check for the dash value. If so, skip to lines 4511-4515 and print out "FN" also.

This way, FN AB will come out as "AB-FN", which is a bit of a cop-out on my part. But I opted for making minimal changes to VCR to keep things simple.

If you play with long programs also having defined functions, as I have, these additions to VCR should help.

[ Now that the Variable Cross Reference program has been modified a couple of times, and since it appeared way back in the second issue of Apple Assembly Line, we'll include the complete source code, including all of Louis' enhancements, on the next Quarterly Disk. Remember that all of the back issues are still available, if you don't have Volume 1, Number 2. ...Bill ]

```
2132        CMP $#C2
2134        BEQ .4

2222 .4     STA $7       set FLAG2
2224        BEQ .1       ...always
```

```
2226 *  unless syntax error, NEXT.CHAR.NOT.QUOTE
2228 *  will be letter, hence variable!

2791        LDA $7       recall FLAG2
2792        CMP #$C2     FN token?
2793        BNE .5       (to RTS)
2794        LDA #'-+80   "-"
2795        STA VARNAM+2 to indicate FN
2796        STA $7       and reset FLAG2

4292        CMP #$AD     not array, but FN?
4294        BEQ .6

4511 .6     LDA #'F      add 'FN' after
4512        JSR PRINT.CHAR
4513        LDA #'N      variable name
4514        JSR PRINT.CHAR
4515        BNE .4       ...always
```

```
=================================================================
DOCUMENT :AAL-8310:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 1.0...................................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50
Full Screen Editor for S-C Macro Assembler..........(reg. $49.00)  $40.00**
    Includes complete source code.
S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $40.00**

S-C Word Processor (the one we use!)..............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor.....(reg. $60.00)  $40.00**

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research).....(reg. $79.00)  $50.00**

Blank Diskettes.................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Shift-Key Modifier................................................$15.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books.........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Micro Cookbook, vol. 2", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
```

```
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00
  Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***

(** Special price to subscribers only through December 31, 1983.)
```

==========================================================================
DOCUMENT :AAL-8310:Articles:PDos.Disasm.Xp.txt
==========================================================================

Commented Listing of ProDOS $F800-$F90B, $F996-FEBD
.....................Bob Sander-Cederlof

ProDOS boots its bulk into the RAM card, from $D000 thru $FFFF. More is loaded into the alternate $D000-DFFF space, and all but 255 bytes are reserved out of the entire 16K space.

A system global page is maintained from $BF00-BFFF, for various variables and linkage routines. All communication between machine language programs and ProDOS is supposed to be through MLI (Machine Language Interface) calls and the system global page.

One of the first things I did with ProDOS was to start dis-assembling and commenting it. I want to know what is inside and how it works! Apple's 4-inch thick binder tells a lot, but not all.

Right away I ran into a roadblock: to disassemble out of the RAM card it has to be turned on. There is no monitor in the RAM card when ProDOS is loaded. Turning on the RAM card from the motherboard monitor causes a loud crash!

I overcame most of the problem by copying a monitor into the $F800-FFFF region of the RAM card like this:

```
       *C089 C089 F800<F800.FFFFM
       *C083 C083
```

The double C089 write-enables the RAM card, while memory reads are still from the motherboard. The rest of the line copies a monitor up. The two C083's get me into the RAM card monitor, ready to type things like "D000LLLLLLLLLLL"

But what about dis-assemblies of the space between $F800 and $FFFF? For this I had to write a little move program. My program turned on the RAM card and copied $F800-FFFF down to $6800-6FFF. Then I BSAVEd it, and later disassembled it.

The code from $F800-FFFF is the mostly equivalent to what is in DOS 3.3 from $B800-BFFF. First I found a read/write block subroutine, which calls an RWTS-like subroutine twice per block. (All ProDOS works with 512-byte blocks, rather than sectors; this is like Apple Pascal, and the Apple ///.)

The listing which follows shows the RWB and RWTS subroutines, along with the READ.ADDRESS and READ.SECTOR subroutines. Next month I plan to lay out the SEEK.TRACK and WRITE.SECTOR subroutines, as well as the interrupt and reset handling code.

The outstanding difference between ProDOS and DOS 3.3 disk I/O is speed. ProDOS is considerably faster. Most of the speed increase is due to handling the conversion between memory-bytes and disk-bytes on the fly. DOS pre-converted a 256-byte block into 342 bytes in a special buffer, and then wrote the 342 bytes; ProDOS forms the first 86 bytes of the disk data in a special buffer, writes them, and then proceeds to write the rest of the data directly from the caller's buffer. When reading, DOS read the 342 disk-bytes into a buffer for later decoding into the caller's buffer. ProDOS reads and decodes simultaneously directly into the caller's buffer. The result is achieved by extensive use of tables and self-modifying code.

Not only is direct time saved by doing a lot less copying of buffers, but also the sector interleaving can be arranged so that only two revolutions are required to read all 8 blocks on a track.

I believe Apple Pascal uses the same technique, at least for reading.

Whoever coded ProDOS decided to hard-code some parameters which DOS used to keep in tables specified by the user. For example, the number which tells how long to wait for a drive motor to rev up used to be kept in a Device Characteristics Table (DCT). That value is now inside a "LDA #$E8" instruction at $F84F. That means that if you are using a faster drive you have to figure out how to patch and unpatch ProDOS to take advantage of it.

Another hard-coded parameter is the maximum block number. This is no longer part of the data on an initialized disk. It is now locked into the four instructions at $F807-F80D, at a maximum of 279. If you have a 40- or 70-track drive, you can only use 35. Speaking of tracks, the delay tables for track seeking are still used, but they are of course buried in this same almost-unreachable area. If you have a drive with faster track-to-track stepping, the table to change is at $FB73-FB84.

Calls to RWTS in DOS 3.3 involved setting up two tables, an IOB and a DCT. The IOB contained all the data about slot, drive, track, sector, buffer address, etc. The DCT was a 5-byte table with data concerning the drive. ProDOS RWB is called in an entirely different way. A fixed-position table located at $42-47 in page zero is set up with the command, slot, buffer address, and block number.

There are three valid commands, which I call test, read, and write. Test (0) starts up the indicated drive. If it is successful, a normal return occurs; if not, you get an error return (carry set, and (A) non-zero). Read (1) and write (2) are what you expect them to be. RWB has a very simple job: validate the call parameters in $42-47, convert block number to track and sector, and call RWTS twice (once for each sector of the block).

ProDOS RWTS expects the sector number in the A-register, and the track in a variable at $FB56. RWTS handles turning on the drive motor and waiting for it to come up to speed. RWTS then calls SEEK.TRACK to find the desired track, READ.ADDRESS to find the selected sector, and branches READ.SECTOR or WRITE.SECTOR depending on the command.

READ.ADDRESS is virtually the same in ProDOS as it was in DOS 3.3. READ.SECTOR is entirely different. I should point out here that ProDOS diskettes are entirely compatible with Apple /// diskettes. The file structures are exactly the same. Both ProDOS and Apple /// diskettes use the same basic recording techniques on the disk as DOS 3.3, so the diskettes are copyable using standard DOS 3.3 copiers such as the COPYA program on your old System Master Diskette.

READ.SECTOR begins by computing several addresses and plugging them into the code further down. (This enables the use of faster addressing modes, saving enough cycles to leave time for complete decoding of disk data on the fly.) First the disk slot number is merged into the instructions which read bytes from the drive. Next the caller's buffer address is put into the store instructions.

Your buffer is divided into three parts: two 86-byte chunks, and one of 84 bytes. Data coming from the disk is in four chunks: three of 86 bytes, and one of 84.

The first chunk contains the lower two bits from every byte in the original data. READ.SECTOR reads this chunk into TBUF, so that the bits will be available later for merging with the upper six of each byte. ($FC53-FC68)

The second chunk contains the upper six bits from the first 86 bytes of the original data. $FC69-FC83 reads the chunk and merges in the lower two bits from TBUF, storing the completed bytes in the first 85 bytes of the caller's buffer. The last (86th) byte is saved on the stack (I am not sure why), and not stored in the caller's buffer until after all the rest of the data has been read.

A tricky manipulation is necessary to merge in those lower two bits. The data in TBUF has those bits in backward order, packed together with the bits from the other chunks. There was a good diagram of this on page 10 of the June 1981 issue of AAL. DOS merged them with a complex time-consuming shifting process. ProDOS does a swift table lookup, using the TBUF byte as an index to the BIT.PAIR.TABLE.

BIT.PAIR.TABLE has four bytes per row. The first three in each row supply the bit pairs; the fourth is used by SECTOR.WRITE to encode data, and will be covered next month.

When $FC69-FC83 is reading the first chunk, the first byte in each row is used to supply the lower two data bits. The byte in TBUF corresponding to the current position in the chunk selects a byte from BIT.PAIR.TABLE, and the two parts are merged together.

The next two chunks are handled just like the one I just described. After all the data has been read, READ.SECTOR expects to have accumulated a checksum of 00, and expects to find a trailing $EB after the data. Return with carry clear indicates all went well; carry set indicates a read error (bad checksum, missing header, or missing trailer).

I can't help wondering:  can this fast read technique be fit into DOS 3.3?  It takes a little more code and table space, but on the other hand it uses 256 bytes less of intermediate buffer.  If we sacrificed the INIT command, could both the fast read and write be squeezed into DOS 3.3?

==============================================================================
DOCUMENT :AAL-8310:Articles:Price.Changes.txt
==============================================================================

Price Changes...............................Bob Sander-Cederlof

It has been nearly two years since we raised the price of a subscription from $12 to $15 per year, and now we are forced to another increase. Effective January 1, 1984, a year's subscription by bulk mail in the USA will be $18. For First Class Mail in the USA, Canada, and Mexico, add $3. Subscriptions to other countries, including postage, will be $30 per year.

You can beat the price by renewing early. All renewals received before January 1st will be at the old rates.

Now for some good news! We want to reduce our inventory of back issues, so we are offering some special prices. We normally sell them for $1.50 each; between now and January 1st you can buy them for only $1 each!

We want to encourage more of you to save your time and energy by getting the Quarterly Disks, with all the source code from three issues already correctly entered. Each Quarterly Disk costs only $15. To save even more trouble, and some $$$, you can subscribe to the Quarterly Disks. Effectively immediately, prepaid subscriptions for four Quarterly Disks will be only $45. You save 25%!

Continuing in the Christmas spirit, here are some more specials good through the end of this year, only for subscribers to Apple Assembly Line:

| | Regular | Special |
|---|---|---|
| FLASH! Integer BASIC Compiler | $79 | $50 |
| The Visible Computer: 6502 | $50 | $40 |
| ES-CAPE | $60 | $40 |
| S-C Math Disk & Game Disk Set | $35 | $20 |
| Laumer's Full Screen Editor | $49 | $40 |

==============================================================================
DOCUMENT :AAL-8310:Articles:Rates.txt
==============================================================================

Apple Assembly Line is published monthly by S-C Software Corporation, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

Every three months the source code is collected into a Quarterly Disk. The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec. Each Quarterly Disk costs $15.00.

==============================================================================
DOCUMENT :AAL-8310:Articles:Red.Faces.txt
==============================================================================

Duplicated Ideas and Red Faces..............Bob Sander-Cederlof

I suppose it had to happen at least once in three years, but it still came as a shock.

Last June I wrote and published a program and article called Amper-Monitor, and then I did it all over again for the September issue. The programs are slightly different, both in design and implementation, but they still do the same thing.

Maybe now that we have a complete index to the first three volumes I won't make this mistake again.

```
==============================================================================
DOCUMENT :AAL-8310:Articles:ScreenWriter.II.txt
==============================================================================
```

Faster Booting for ScreenWriter II....................Bob Leedom
Glenwood, Maryland

I have found a solution to ScreenWriter II's long boot-up time (which is one of my few complaints with the product). Would you believe a reduction from 46 seconds to just under 14 seconds?

The solution was given in a patch to DOS 3.3 given by Paul Schlyter and Bob Sander-Cederlof in the April 1983 issue of AAL. Since ScreenWriter's DOS is nearly identical to 3.3, I was inspired to try the patch (on ONE of my two copy-protected original disks) -- and it worked!

I installed the patch between lines 50 and 60 of APP2 (ScreenWriter's customizeable startup program). The POKEs will only be performed at startup -- if you look closely at APP2, you'll see that the POKEing lines will be skipped when the program is used to switch between Editor and Runoff in the non-RAMcard version.

To install the patch, do the following:

1. From BASIC, LOAD APP2
2. Type in Lines 51 - 59, carefully!
3. SAVE APP2
4. RUN CUSTOMIZEA

That's it! You will now have a fast-booting ScreenWriter. You may also want to do this to some of your normal DOS 3.3 disks -- the patch is in an unused area of DOS, and seems to coexist happily with everything else I tried (like PLE and GPLE for instance). Exception: in the //e version of DOS 3.3, the patch screws up the infamous APPEND command -- no great loss, in my opinion.

```
51 READ N: IF N=0 THEN 59:REM Make this "THEN 60" (60 is the next
ScreenWriter II line) when line 59 is DELETEd
52 READ A: SUM = SUM + A + N
53 FOR I = 1 TO N: READ P: POKE A,P: A=A+1: SUM=SUM+P: NEXT
54 GOTO 51
55 DATA 44, 47721, 173, 230, 181, 208, 36, 173, 194, 181, 240, 31,
173, 203, 181, 72, 173, 204, 181, 72, 173, 195, 181, 141, 203, 181,
173, 196, 181, 141, 204, 181, 32, 182, 176, 176, 3, 76, 223, 188, 76,
111, 179, 76, 150, 172
56 DATA 33, 48351, 238, 228, 181, 208, 3, 238, 229, 181, 238, 196,
181, 238, 204, 181, 206, 194, 181, 208, 11, 104, 141, 204, 181, 104,
141, 203, 181, 76, 150, 172, 76, 135, 186
57 DATA 2, 44198, 105, 186
58 DATA 0
59 IF SUM <> 153114 THEN PRINT "OOPS! DATA IS OFF BY "153114-SUM:
STOP: REM (Delete this line when you are SURE it works!)
```

```
=============================================================================
DOCUMENT :AAL-8310:Articles:ShapeMaker.txt
=============================================================================
```

If You Like Shapes, Try Shapemaker.........Bob Sander-Cederlof

Frank Belanger sent me a copy of his new Hi-Res utility program, called SHAPEMAKER. I know, there are a lot of these on the market, such as Accu-Shapes and Apple Mechanic. Frank's is priced between those two, at $35, and look at all you get:

* Shape Editor
* Shape Table Editor
* Nine &-routines, including
    * Clear any window in the hi-res page
    * Display string of shapes
    * Input anything
* 44-page manual
* Source Code of &-routines in S-C format
* Unprotected, copyable, modifiable

If these features interest you, write Frank at 4200 Avenue B, Austin, TX 78751. Or call (512) 451-6868.

```
===========================================================================
DOCUMENT :AAL-8310:Articles:Supress.Hex.txt
===========================================================================
```

Suppressing Unwanted Object Bytes

Sometimes we want to get an assembly listing that doesn't use up half a page of paper for each .AS or .HS line, listing three object bytes on each line.  A number of you have asked for a patch to show the source line without listing each and every one of those hex bytes.

Well, David Roberts, a subscriber in Australia, has come up with a simple way to do just that.  He uses macros!  David suggests these definitions:

```
.MA AS
.AS -"]1"
.EM

.MA AT
.AT "]1"
.EM

.MA HS
.HS "]1"
.EM
```

Now you can code text with >AS "THIS IS MY STRING", and use the .LIST MOFF option to suppress the hex listing.  That's really a "why didn't I think of that?"  Thanks, David.

```
==============================================================================
DOCUMENT :AAL-8310:Articles:Where.To.txt
==============================================================================
```

Where To?...........................................Bill Morgan

The word is that the new Mackintosh machine from Apple is going to be 68000-based and affordable.  I know that I am going to want one, and I would like to get a leg up on learning the machine, so I'm starting to study 68000.  It looks like a lot of fun.  With seventeen registers addressing 16 megabytes at 12 megaHertz or thereabouts, we should be able to do just about anything we want.  I'll have a review next month of a new 68000 trainer board for your Apple, at about half the price of the existing 68000 boards.

To get to the point, how many of you good folks out there are interested in 68000?  How many of you already know a little or a lot about it?  Should we start a new newsletter about Mackintosh?  Should we devote a few pages of this one to it?  Let us hear from you.

And another thing, how about C language?  Several of you have mentioned that great August issue of Byte and expressed an interest in learning more about C.  I know that I'm going to study up on it. There is a good C compiler available for the Apple, the Aztec C Compiler System from Manx Software.  I'll have a review of it in the next month or two, and we may start carrying it for sale.  Let me know if you're interested.

==============================================================================
DOCUMENT :AAL-8310:Articles:Writers.Guide.txt
==============================================================================

date

name
address
city, state zip

name,

Thank you for your inquiry about writing for Apple Assembly Line.

We accept articles and programs in almost any form, including paper, but the best thing is to have the article in a standard DOS 3.3 Text file, and the program source code in an S-C Macro Assembler type I file.

We use our own word processor, which reads either standard text files or Applewriter files. If the article file is in some other format, please tell us what the data looks like.

Of course, we use the S-C Macro Assembler. (Doesn't every- body?) We can translate programs from other assemblers, if the source code is in a standard text file, but we do prefer S-C format.

The only payment we can offer for your article is glory: Apple Assembly Line reaches about 1300 readers all around the world. In case you are interested in a stepping-stone to the trade magazines, our subscribers include the editors and/or publishers of SofTalk, Nibble, Apple Orchard, and Call-A.P.P.L.E., as well as many software publishers.

Once again, thanks for writing. We look forward to seeing your articles.

Sincerely,

Bill Morgan

```
==============================================================================
DOCUMENT :AAL-8310:DOS3.3:KnouseMtrPatch.txt
==============================================================================
```

```
 1000  *SAVE S.KNOUSE'S MONITOR PATCHES
 1010  *--------------------------------
 1020  *
 1030  *  A COMPILATION OF MONITOR MODIFICATIONS
 1040  *--------------------------------
 1050  YES         .EQ 1
 1060  NO          .EQ 0
 1070  *
 1080  *  OPTIONS
 1090  *
 1100  NFC         .EQ YES  SET TO YES IF YOU WANT
 1110  *                    A NON-FLASHING CURSOR
 1120  LOWERCASE   .EQ YES  SET TO YES IF YOU CAN
 1130  *                    DISPLAY LOWER CASE
 1140  W.APPLESOFT .EQ YES SET TO YES IF YOU WANT
 1150  *                    TO MOVE APPLESOFT WITH
 1160  *                    THE MONITOR, ELSE SET
 1170  *                    TO NO IF YOU ONLY WANT
 1180  *                    TO MOVE AND MODIFY THE
 1190  *                    MONITOR
 1200  *--------------------------------
 1210  PNTR   .EQ $00,01
 1220  PATCH  .EQ $02,03
 1230  A1L    .EQ $3C
 1240  A1H    .EQ A1L+1
 1250  A2L    .EQ $3E
 1260  A2H    .EQ A2L+1
 1270  A4L    .EQ $42
 1280  A4H    .EQ A4L+1
 1290  BASL   .EQ $28
 1300  CH     .EQ $24
 1310  KSWL   .EQ $38
 1320  *--------------------------------
 1330  COUT        .EQ $FDED
 1340  CRMON       .EQ $FEF6
 1350  CROUT       .EQ $FD8E
 1360  MON.HEADR   .EQ $FCC9
 1370  MON.MOVE    .EQ $FE2C
 1380  NXTA1       .EQ $FCBA
 1390  PRA1        .EQ $FD92
 1400  PRBYTE      .EQ $FDDA
 1410  PRERR       .EQ $FF2D
 1420  RDKEY       .EQ $FD0C
 1430  MON.READ    .EQ $FEFD
 1440  MON.WRITE   .EQ $FECD
 1450  *--------------------------------
 1460  ROMR.RAMW   .EQ $C081
 1470  RAMRW       .EQ $C083
 1480  *--------------------------------
```

```
1490  *
1500  *  GENERAL PURPOSE PATCHER
1510  *
1520  *-------------------------------
1530         .OR $2D0
1540         .DO W.APPLESOFT
1550             .TF PATCH MONITOR & APPLESOFT
1560  MON.START  .EQ $D000
1570         .ELSE
1580             .TF PATCH MONITOR ONLY
1590  MON.START  .EQ $F800
1600         .FIN
1610  MON.END    .EQ $FFFF
1620  *-------------------------------
1630  PATCH.MONITOR
1640         LDA #MON.START    COPY MONITOR TO RAM CARD
1650         STA A4L
1660         STA A1L
1670         LDA /MON.START
1680         STA A4H
1690         STA A1H
1700         LDA #MON.END
1710         STA A2L
1720         LDA /MON.END
1730         STA A2H
1740         LDA ROMR.RAMW     WRITE ENABLE RAM CARD
1750         LDA ROMR.RAMW     BY 2 OF THESE
1760         LDY #0            SET UP MON.MOVE
1770         JSR MON.MOVE      COPY FROM MOTHERBOARD TO RAMCARD
1780  *
1790         LDA #PATCHES-1
1800         STA PNTR
1810         LDA /PATCHES-1
1820         STA PNTR+1
1830         LDY #0
1840  *
1850  .1     JSR GET.BYTE LENGTH OF NEXT PATCH
1860         BEQ .4
1870         TAX          SAVE LENGTH IN X
1880         JSR GET.BYTE GET ADDR OF PATCH
1890         STA PATCH
1900         JSR GET.BYTE
1910         STA PATCH+1
1920  *
1930  .2     JSR GET.BYTE    GET A BYTE
1940         STA (PATCH),Y   STORE AT DESTINATION
1950         INC PATCH       BUMP SOURCE ADDRESS
1960         BNE .3
1970         INC PATCH+1
1980  .3     DEX             DECREMENT NUMBER OF BYTES
1990         BNE .2          LOOP FOR MORE
2000         BEQ .1      ... ALWAYS
2010  *
2020         .DO W.APPLESOFT
```

```
2030 .4     LDA RAMRW
2040        RTS
2050        .ELSE
2060 .4     RTS
2070        .FIN
2080 *--------------------------------
2090 GET.BYTE
2100        INC PNTR
2110        BNE .1
2120        INC PNTR+1
2130 .1     LDA (PNTR),Y
2140        RTS
2150 *--------------------------------
2160        .MA PATCH
2170 ]1.ORG .EQ ]2
2180        .DA #]1.LENGTH
2190        .DA ]1.ORG
2200        .PH ]1.ORG
2210 ]1
2220        .EM
2230 *
2240        .MA ENDP
2250 ]1.END .EQ *-1
2260 ]1.LENGTH  .EQ *-]1
2270        .EP
2280        .EM
2290 *--------------------------------
2300 PATCHES    .EQ *
2310 *--------------------------------
2320 *  MONITOR LOWERCASE INPUT ROUTINE
2330 *--------------------------------
2340 *
2350 *---DON'T STOMP ON LOWERCASE-----
2360     >PATCH NOP.CONVERT,$FD82
2370        AND #$FF     DO NOTHING
2380     >ENDP  NOP.CONVERT
2390 *
2400 *---MAKE SENSIBLE CURSOR---------
2410     >PATCH HANDLE.CURSOR,$FBB3
2420        CMP #$E0     IS IT LOWER CASE?
2430        BCS .1
2440        AND #$3F     NO - MAKE CHAR INVERSE
2450        .DO NFC
2460        ORA #$00
2470        .ELSE
2480        ORA #$40     THEN FLASHING
2490        .FIN
2500        RTS
2510 .1     AND #$1F     CONVERT TO UC INVERSE
2520        RTS
2530     >ENDP  HANDLE.CURSOR
2540 *
2550 *---CALL NEW CURSOR ROUTINE------
2560     >PATCH VEC.HANDLE.CURSOR,$FD11
```

```
       JSR HANDLE.CURSOR  GO TO PATCH
       NOP          FILL BYTE
    >ENDP  VEC.HANDLE.CURSOR
*--------------------------------
*  ASCII DUMP
*--------------------------------
*
*---MODIFIED DUMPER--------------
    >PATCH ASC.DUMP,MON.HEADR
       PHA          SAVE CHAR
       LDA A1L      GET LO ADDR BYTE
       AND #$07     MOD 8
       CLC          ADD DISPLACEMENT
       ADC #30        OF 30 CHAR
       TAY
       PLA          RECOVER CHARACTER
       PHA          SAVE IT AGAIN
       ORA #$80     FORCE NORMAL VIDEO
       CMP #$A0     MAKE CONTROL CHAR INVERSE
       BCS .1       ...NOT CONTROL
       .DO LOWERCASE
       AND #$7F     ...CONTROL
.1     STA (BASL),Y PUT ON SCREEN
       NOP          TO STAY ALIGNED W/
       NOP          NON-LOWERCASE CODE
       NOP
       NOP
       NOP
       NOP
       .ELSE
       LDA #$DF     MAKE CTRL-CHARS INVERSE
.1     CMP #$E0     IN LOWER CASE RANGE?
       BCC .2       ..NO, DISPLAY NORMAL VIDEO
       AND #$1F     ..YES, FORCE INVERSE VIDEO
.2     STA (BASL),Y STORE IT ON SCREEN
       .FIN
       LDY #0       RESTORE Y REG
       PLA          RECOVER BYTE AGAIN
       JMP PRBYTE
    >ENDP  ASC.DUMP
*
*---CALL ASCII DUMP--------------
    >PATCH VEC.ASC.DUMP,$FDBD
       JSR ASC.DUMP
    >ENDP  VEC.ASC.DUMP
*--------------------------------
*   + CURSOR IN ESCAPE MODE
*--------------------------------
*
*---SAVE SCREEN, SPOT + ---------
    >PATCH RDKEY2,ASC.DUMP.END+1
       LDY CH       SAVE CHARACTER
       LDA (BASL),Y
       PHA
```

```
       LDA #'+      PUT AN INVERSE + ON SCREEN
       STA (BASL),Y
       PLA          GET THE CHARACTER BACK
       JMP (KSWL)
       .BS RDKEY-*  FILL W/ 0'S TO RDKEY
    >ENDP  RDKEY2
*
*---CALL + CURSOR----------------
    >PATCH VEC.RDKEY2.1,$FBA2
       JSR RDKEY2
    >ENDP  VEC.RDKEY2.1
*
*---CALL + CURSOR----------------
    >PATCH VEC.RDKEY2.2,$FD2F
       JSR RDKEY2
    >ENDP  VEC.RDKEY2.2
*-------------------------------
*   MASK BIT CONTROL OVER MEMORY RANGE
*   XXYY<ADR1.ADR2W   FORMS M=(M.AND.XX).OR.YY
*-------------------------------
*
    >PATCH WRITE,MON.WRITE
       LDA (A1L),Y  GET A BYTE
       AND A4H      AND IT WITH XX
       ORA A4L      OR  IT WITH YY
       STA (A1L),Y  PUT IT BACK
       JSR NXTA1    INCR ADDRESS
       BCC WRITE    LOOP FOR MORE
       RTS
       .BS CRMON-*  FILL W/ 0'S TO CRMON
    >ENDP  WRITE
*-------------------------------
*   SEARCH
*   XXYY<ADR1.ADR2S
*-------------------------------
*
*---SEARCH PROCESSOR-------------
    >PATCH SEARCH,MON.READ
       LDA A4H      IS THIS A 1 OR 2 BYTE COMPARE
       BEQ .2       ..ONE BYTE
       LDA A2L      ..TWO BYTE
       BNE .1       DECREMENT ENDING ADDR
       DEC A2H
.1     DEC A2L
*
.2     LDA A4H      GET FIRST BYTE TO COMPARE
       BEQ .3       IF ZERO DO A ONE BYTE SEARCH
       CMP (A1L),Y  COMPARE WITH MEMORY
       BNE .4       NOT EQUAL - GO TO NEXT BYTE
       INY          GET NEXT BYTE
.3     LDA (A1L),Y
       LDY #0       RESTORE Y REG
       CMP A4L      COMPARE
       BNE .4       NOT EQUAL - DRIVE ON
```

```
3650         JSR PRA1
3660  .4     JSR NXTA1    GET NEXT BYTE
3670         BCC .2       LOOP FOR MORE
3680         RTS
3690         .BS PRERR-*  FILL W/ 0'S TO PRERR
3700      >ENDP  SEARCH
3710  *
3720  *---PATCH COMMAND TABLE----------
3730      >PATCH VEC.SEARCH,$FFDE
3740         .DA #$EC     'S' EOR $B0 + $89
3750      >ENDP  VEC.SEARCH
3760  *--------------------------------
3770         .DA #0       END OF PATCHES
3780  *--------------------------------
3790  END    .EQ *-1
3800  LENGTH .EQ END-PATCH.MONITOR+1
3810         .LIST OFF
```

```
===============================================================================
DOCUMENT :AAL-8310:DOS3.3:S.LINE.COUNTER.txt
===============================================================================

 1000  COUNT.LO .EQ 0
 1010  COUNT.HI .EQ 1
 1020  OUTHOOK  .EQ $36
 1030  DOSHOOK  .EQ $3EA
 1040  *---------------------------------
 1050         .OR $300
 1060
 1070         LDA #0
 1080         STA COUNT.LO      zero the counters
 1090         STA COUNT.HI
 1100         LDA #LINE.COUNTER
 1110         STA OUTHOOK       direct output
 1120         LDA /LINE.COUNTER to my routine
 1130         STA OUTHOOK+1
 1140         JMP DOSHOOK
 1150  *---------------------------------
 1160  LINE.COUNTER
 1170         CMP #$8D          carriage return?
 1180         BNE .1            no, exit
 1190         INC COUNT.LO      yes, count it
 1200         BNE .1
 1210         INC COUNT.HI
 1220  .1     RTS
```

```
==============================================================================
DOCUMENT :AAL-8310:DOS3.3:S.LOVES.SPIRAL.txt
==============================================================================

 1000        .TF CLEAR
 1010        .LIST OFF
 1020  *--------------------------------
 1030        .MA SPIRAL
 1040        >LEFT              move left side up
 1050  BOTLFT .SE BOTLFT-1        and move corner up
 1060        >BOTTOM            move bottom left
 1070  BOTRGT .SE BOTRGT-1        and move corner left
 1080        >RIGHT             move right side down
 1090  TOPRGT .SE TOPRGT+1        and move corner down
 1100        >TOP               move top right
 1110  TOPLFT .SE TOPLFT+1        and move corner right
 1120        .DO TOPLFT<13      done?
 1130        >SPIRAL            no, do it again
 1140        .FIN
 1150        .EM
 1160  *--------------------------------
 1170        .MA GETADR
 1180  ADRTO  .SE ADRFRM
 1190  BLOCK  .SE Y.CORD/8      hi, mid, or low, 0-2
 1200  BLK.AD .SE BLOCK*$28     block offset
 1210  TEMP   .SE BLOCK*8
 1220  LINE   .SE Y.CORD-TEMP   line within block, 0-7
 1230  LIN.AD .SE LINE*$80      line offset
 1240  ADRFRM .SE $400+BLK.AD+LIN.AD+X.CORD
 1250        LDA ADRFRM
 1260        STA ADRTO
 1270        .EM
 1280  *--------------------------------
 1290        .MA LEFT
 1300  Y.CORD .SE Y.CORD+1      down one step
 1310        >GETADR
 1320        .DO Y.CORD<BOTLFT done?
 1330        >LEFT              no, again
 1340        .FIN
 1350        .EM
 1360  *--------------------------------
 1370        .MA BOTTOM
 1380  X.CORD .SE X.CORD+1      right one step
 1390        >GETADR
 1400        .DO X.CORD<BOTRGT done?
 1410        >BOTTOM            no, again
 1420        .FIN
 1430        .EM
 1440  *--------------------------------
 1450        .MA RIGHT
 1460  Y.CORD .SE Y.CORD-1      up one step
 1470        >GETADR
 1480        .DO Y.CORD>TOPRGT done?
```

```
1490        >RIGHT           no, again
1500        .FIN
1510        .EM
1520  *--------------------------------
1530        .MA TOP
1540  X.CORD .SE X.CORD-1      left one step
1550        >GETADR
1560        .DO X.CORD>TOPLFT done?
1570        >TOP             no, again
1580        .FIN
1590        .EM
1600  *--------------------------------
1610  BOTLFT .SE 23       bottom left Y coord
1620  BOTRGT .SE 39       bottom right X coord
1630  TOPRGT .SE 0        top right Y coord
1640  TOPLFT .SE 1        top left X coord
1650  X.CORD .SE 0        start with upper
1660  Y.CORD .SE 0          left corner
1670  ADRFRM .SE $400
1680  *--------------------------------
1690        LDX #960     do the loop 960 times
1700        LDY /960
1710        LDA #$A0     put space in center
1720        STA $5B4
1730  LOOP   >SPIRAL      do one spiral
1740  END    DEX
1750        BNE .1       branch if not done
1760        DEY
1770        BPL .1
1780        JMP $3D0     exit to DOS
1790  .1     JMP LOOP     go spiral again
```

```
==============================================================================
DOCUMENT :AAL-8310:DOS3.3:S.LoveSpiralFst.txt
==============================================================================

 1000        .TF CLEAR
 1010        .LIST OFF
 1020  *--------------------------------
 1030        .MA SPIRAL
 1040        >LEFT             move left side up
 1050  BOTLFT .SE BOTLFT-1       and move corner up
 1060        >BOTTOM           move bottom left
 1070  BOTRGT .SE BOTRGT-1       and move corner left
 1080        >RIGHT            move right side down
 1090  TOPRGT .SE TOPRGT+1       and move corner down
 1100        >TOP              move top right
 1110  TOPLFT .SE TOPLFT+1       and move corner right
 1120        .DO TOPLFT<13     done?
 1130        >SPIRAL           no, do it again
 1140        .FIN
 1150        .EM
 1160  *--------------------------------
 1170        .MA GETADR
 1180  ADRTO  .SE ADRFRM
 1190  BLOCK  .SE Y.CORD/8      hi, mid, or low, 0-2
 1200  BLK.AD .SE BLOCK*$28     block offset
 1210  TEMP   .SE BLOCK*8
 1220  LINE   .SE Y.CORD-TEMP   line within block, 0-7
 1230  LIN.AD .SE LINE*$80      line offset
 1240  ADRFRM .SE $400+BLK.AD+LIN.AD+X.CORD
 1250        LDA ADRFRM
 1260        STA ADRTO
 1270        .EM
 1280  *--------------------------------
 1290        .MA LEFT
 1300  Y.CORD .SE Y.CORD+1      down one step
 1310        >GETADR
 1320        .DO Y.CORD<BOTLFT done?
 1330        >LEFT             no, again
 1340        .FIN
 1350        .EM
 1360  *--------------------------------
 1370        .MA BOTTOM
 1380  X.CORD .SE X.CORD+1      right one step
 1390        >GETADR
 1400        .DO X.CORD<BOTRGT done?
 1410        >BOTTOM           no, again
 1420        .FIN
 1430        .EM
 1440  *--------------------------------
 1450        .MA RIGHT
 1460  Y.CORD .SE Y.CORD-1      up one step
 1470        >GETADR
 1480        .DO Y.CORD>TOPRGT done?
```

```
1490        >RIGHT            no, again
1500        .FIN
1510        .EM
1520 *--------------------------------
1530        .MA TOP
1540 X.CORD .SE X.CORD-1      left one step
1550        >GETADR
1560        .DO X.CORD>TOPLFT done?
1570        >TOP              no, again
1580        .FIN
1590        .EM
1600 *--------------------------------
1610 BOTLFT .SE 23       bottom left Y coord
1620 BOTRGT .SE 39       bottom right X coord
1630 TOPRGT .SE 0        top right Y coord
1640 TOPLFT .SE 1        top left X coord
1650 X.CORD .SE 0        start with upper
1660 Y.CORD .SE 0          left corner
1670 ADRFRM .SE $400
1680 POINTER .EQ 0
1690 *--------------------------------
1700        LDY #0       no indexing
1710        LDA #END     start pointer at end of code
1720        STA POINTER
1730        LDA /END
1740        STA POINTER+1
1750 .2     JSR LOOP     do one step
1760        LDA #$AD     restore LDA code
1770        STA (POINTER),Y
1780 *--------------------------------
1790        LDA POINTER  decrement pointer
1800        SEC          by 6
1810        SBC #6
1820        STA POINTER
1830        BCS .1
1840        DEC POINTER+1
1850 .1     LDA #$60     insert RTS code
1860        STA (POINTER),Y
1870 *--------------------------------
1880        LDA POINTER  compare pointer
1890        CMP #LOOP    to beginning of code
1900        BNE .2
1910        LDA POINTER+1
1920        SBC /LOOP
1930        BNE .2       branch if not yet done
1940 *--------------------------------
1950 FIXUP  LDA #$AD     restore LDA
1960        STA LOOP      at beginning
1970        LDA #$60     and RTS
1980        STA END       at end
1990        JMP $3D0     and reenter DOS
2000 *--------------------------------
2010 SAVE   .DA #$A0     <space> to fill screen
2020 *--------------------------------
```

```
2030  LOOP   >SPIRAL
2040         LDA SAVE
2050         STA $5B4
2060  END    RTS
```

```
===================================================================
DOCUMENT :AAL-8310:DOS3.3:S.VCR.REVISED.txt
===================================================================

 1000  *--------------------------------
 1010  *      VARIABLE CROSS REFERENCE
 1020  *       FOR APPLESOFT PROGRAMS
 1030  *--------------------------------
 1040  ZZ.BEG .EQ $8800
 1050         .OR ZZ.BEG
 1060         .TF B.VCRP
 1070  *--------------------------------
 1080         LDA #$4C     AMPERSAND VECTOR
 1090         STA $3F5
 1100         LDA #VCR
 1110         STA $3F6
 1120         LDA /VCR
 1130         STA $3F7
 1140         RTS
 1150  *--------------------------------
 1160  PNTR   .EQ $18,19   POINTER INTO PROGRAM
 1170  DATA   .EQ $1A THRU $1D
 1180  LZFLAG .EQ $1A      LEADING ZERO FLAG
 1190  NEXTLN .EQ $1A,1B   ADDRESS OF NEXT LINE
 1200  LINNUM .EQ $1C,1D   CURRENT LINE NUMBER
 1210  STPNTR .EQ $1E,1F   POINTER INTO VARIABLE TABLE
 1220  TPTR   .EQ $9B,9C   TEMP POINTER
 1230  SYMBOL .EQ $9D THRU $A4  8 BYTES
 1240  VARNAM .EQ SYMBOL+1
 1250  HSHTBL .EQ $280
 1260  ENTRY.SIZE .EQ $A5,A6
 1270  *--------------------------------
 1280  PRGBOT .EQ $67,68   BEGINNING OF PROGRAM
 1290  LOMEM  .EQ $69,6A   BEGINNING OF VARIABLE SPACE
 1300  EOT    .EQ $6B,6C   END OF VARIABLE TABLE
 1310  *--------------------------------
 1320  TKN.REM    .EQ 178
 1330  TKN.DATA   .EQ 131
 1340  *--------------------------------
 1350  MON.CH     .EQ $24
 1360  MON.PRBL2  .EQ $F94A
 1370  MON.COUT   .EQ $FDED
 1380  MON.CROUT  .EQ $FD8E
 1390  *--------------------------------
 1400  VCR
 1410         JSR INITIALIZATION
 1420  .1     JSR PROCESS.LINE
 1430         BNE .1       UNTIL END OF PROGRAM
 1440         JSR PRINT.REPORT
 1450         JSR INITIALIZATION  ERASE VARIABLE TABLE
 1452         LDA #0       CLEAR $A4 SO APPLESOFT WILL
 1454         STA $A4      WORK CORRECTLY
 1460         RTS
```

```
1470  *--------------------------------
1480  INITIALIZATION
1490         LDA LOMEM
1500         STA EOT
1510         LDA LOMEM+1
1520         STA EOT+1
1530         LDX #52       # OF BYTES FOR HASH POINTERS
1540         LDA #0
1550  .1     STA HSHTBL-1,X
1560         DEX
1570         BNE .1
1580         LDA PRGBOT
1590         STA PNTR
1600         LDA PRGBOT+1
1610         STA PNTR+1
1620         RTS
1630  *--------------------------------
1640  PROCESS.LINE
1650         LDY #3        CAPTURE POINTER AND LINE #
1660  .1     LDA (PNTR),Y
1670         STA DATA,Y
1680         DEY
1690         BPL .1
1692         LDA DATA+1    CHECK IF END
1694         BEQ .3        YES
1700         CLC           SKIP OVER DATA
1710         LDA PNTR
1720         ADC #4
1730         STA PNTR
1740         BCC .2
1750         INC PNTR+1
1760  .2     JSR SCAN.FOR.VARIABLES
1770         LDA DATA
1780         STA PNTR
1790         LDA DATA+1
1800         STA PNTR+1
1810  *      BNE .3
1820  .3     RTS
1830  *--------------------------------
1840  SCAN.FOR.VARIABLES
1850  .1     JSR GET.NEXT.VARIABLE
1860         BEQ .3        END OF LINE
1870         JSR PACK.VARIABLE.NAME
1880         JSR SEARCH.VARIABLE.TABLE
1890         BCC .2        FOUND SAME VARIABLE
1900         LDA #0
1910         STA SYMBOL+4 START OF LINE NUMBER CHAIN
1920         STA SYMBOL+5
1930         LDA LINNUM+1 MSB FIRST
1940         STA SYMBOL+6
1950         LDA LINNUM
1960         STA SYMBOL+7
1970         LDA #8        ADD 8 BYTE ENTRY
1980         JSR ADD.NEW.ENTRY
```

```
1990        JMP .1
2000 .2     JSR SEARCH.LINE.CHAIN
2010        BCC .1       FOUND SAME LINE NUMBER
2020        LDA #4       ADD 4 BYTE ENTRY
2030        JSR ADD.NEW.ENTRY
2040        JMP .1
2050 .3     RTS
2060 *---------------------------------
2070 GET.NEXT.VARIABLE
2080 .1     JSR NEXT.CHAR.NOT.QUOTE
2090        BEQ .2       END OF LINE
2100        CMP #TKN.DATA
2110        BEQ .3
2120        CMP #TKN.REM
2130        BEQ .2       SKIP TO NEXT LINE
2132        CMP #$C2     FN token?
2134        BEQ .4
2140        JSR LETTER   LETTER?
2150        BCC .1       NO, KEEP LOOKING
2160 .2     RTS
2170 *   DATA, SO SKIP TO NEXT STATEMENT
2180 .3     JSR NEXT.CHAR.NOT.QUOTE
2190        BEQ .2       EOL, RETURN
2200        CMP #':      COLON?
2210        BNE .3       NOT END YET
2220        BEQ .1       ...ALWAYS
2222 .4     STA $7       set FLAG2
2224        BEQ .1       ...always
2226 *  unless syntax error, NEXT.CHAR.NOT.QUOTE
2228 *  will be letter, hence variable!
2230 *---------------------------------
2240 NEXT.CHAR.NOT.QUOTE
2250 .1     JSR NEXT.CHAR
2260        BEQ .2       EOL, RETURN
2270        CMP #'"      QUOTE?
2280        BEQ .3       YES, SCAN OVER QUOTATION
2290 .2     RTS          RETURN
2300 .3     JSR NEXT.CHAR
2310        BEQ .2       EOL, RETURN
2320        CMP #'"      TERMINAL QUOTE?
2330        BNE .3       NOT YET
2340        BEQ .1       ...ALWAYS
2350 *---------------------------------
2360 *      NEXT CHARACTER FROM LINE
2370 *        CALL:  JSR NEXT.CHAR
2380 *      RETURN:  (A)=CHAR FROM LINE
2390 *               IF CHAR .NE. EOL,
2400 *                   INCREMENT PNTR AND
2410 *                   STATUS Z=0
2420 *               IF CHAR .EQ. EOL,
2430 *                   STATUS Z=1
2440 *---------------------------------
2450 NEXT.CHAR
2460        LDY #0
```

```
2470        LDA (PNTR),Y
2480        BEQ .1       EOL
2490        INC PNTR     BUMP POINTER
2500        BNE .1
2510        INC PNTR+1
2520 .1     RTS
2530 *---------------------------------
2540 PACK.VARIABLE.NAME
2550        STA VARNAM   FIRST CHAR OF NAME
2560        LDA #'       BLANKS FOR OTHER TWO CHARS
2570        STA VARNAM+1
2580        STA VARNAM+2
2590        JSR NEXT.CHAR
2600        BEQ .5       END OF LINE
2610        JSR LTRDIG
2620        BCC .2       NOT LETTER OR DIGIT
2630        STA VARNAM+1
2640 .1     JSR NEXT.CHAR IGNORE EXCESS NAME
2650        BEQ .5       END OF LINE
2660        JSR LTRDIG
2670        BCS .1       LETTER OR DIGIT
2680 .2     CMP #'$      DOLLAR SIGN?
2690        BEQ .3       YES
2700        CMP #'%      PER CENT?
2710        BNE .4       NO
2720 .3     STA VARNAM+2
2730        JSR NEXT.CHAR
2740        BEQ .5       END OF LINE
2750 .4     CMP #'(      LEFT PAREN?
2752        BEQ .6       YES
2754        CMP #'"      QUOTE?
2760        BNE .5       NO
2762        LDA PNTR     YES, BACK UP POINTER
2763        BNE .7
2764        DEC PNTR+1
2765 .7     DEC PNTR
2766        RTS
2770 .6     LDA VARNAM+2 SET HIGH BIT
2780        ORA #$80     TO FLAG ARRAY
2790        STA VARNAM+2 REFERENCE
2791        LDA $7       recall FLAG2
2792        CMP #$C2     FN token?
2793        BNE .5       (to RTS)
2794        LDA #'-+$80  "-"
2795        STA VARNAM+2 to indicate FN
2796        STA $7       and reset FLAG2
2800 .5     RTS
2810 *---------------------------------
2820 SEARCH.VARIABLE.TABLE
2830        SEC          CONVERT 1ST CHAR TO
2840        LDA VARNAM   HASH TABLE INDEX
2850        SBC #'A
2860        ASL
2870        ADC #HSHTBL
```

```
2880        STA STPNTR
2890        LDA /HSHTBL
2900        ADC #0
2910        STA STPNTR+1
2920  *---   FALL INTO CHAIN SEARCH ROUTINE
2930  *--------------------------------
2940  CHAIN.SEARCH
2950  .1     LDY #0       POINT AT POINTER IN ENTRY
2960         LDA (STPNTR),Y
2970         STA TPTR
2980         INY
2990         LDA (STPNTR),Y
3000         BEQ .4       END OF CHAIN, NOT IN TABLE
3010         STA TPTR+1
3020         LDX #2       2 MORE CHARS IN SYMBOL
3030         LDY #2       POINT AT NAME IN ENTRY
3040  .2     LDA (TPTR),Y COMPARE NAMES
3050         CMP SYMBOL,Y
3060         BCC .3       NOT THIS ONE, BUT KEEP LOOKING
3070         BNE .4       NOT IN THIS CHAIN
3080         DEX
3090         BEQ .5       NAME IS THE SAME
3100         INY          NEXT BYTE PAIR
3110         BNE .2       ...ALWAYS
3120  *--------------------------------
3130  .3     JSR .5       UPDATE POINTER, CLEAR CARRY
3140         BCC .1       ...ALWAYS
3150  *--------------------------------
3160  .4     SEC          DID NOT FIND
3170         RTS
3180  *--------------------------------
3190  .5     LDA TPTR
3200         STA STPNTR
3210         LDA TPTR+1
3220         STA STPNTR+1
3230         CLC
3240         RTS
3250  *--------------------------------
3260  ADD.NEW.ENTRY
3270         STA ENTRY.SIZE
3280         CLC          SEE IF ROOM
3290         LDX #1
3300         LDY #0
3310         STY ENTRY.SIZE+1
3320  .1     LDA (STPNTR),Y  GET CURRENT POINTER
3330         STA SYMBOL,Y
3340         LDA EOT,Y
3350         STA (STPNTR),Y
3360         STA TPTR,Y
3370         ADC ENTRY.SIZE,Y
3380         STA EOT,Y
3390         INY
3400         DEX
3410         BPL .1
```

```
3420  *---   SEE IF GOING TO BE ENOUGH ROOM
3430         LDA EOT
3440         CMP #ZZ.BEG
3450         LDA EOT+1
3460         SBC /ZZ.BEG
3470         BCS .3       MEM FULL ERR
3480  *---   MOVE ENTRY INTO VARIABLE TABLE
3490         LDY ENTRY.SIZE
3500         DEY
3510  .2     LDA SYMBOL,Y
3520         STA (TPTR),Y
3530         DEY
3540         BPL .2
3550         LDA TPTR
3560         STA STPNTR
3570         LDA TPTR+1
3580         STA STPNTR+1
3590         RTS
3600  .3     JMP MEM.FULL.ERR
3610  MEM.FULL.ERR
3620         BRK
3630  *---------------------------------
3640  SEARCH.LINE.CHAIN
3650         CLC          ADJUST POINTER TO START
3660         LDA STPNTR   OF LINE # CHAIN
3670         ADC #4
3680         STA SYMBOL
3690         LDA STPNTR+1
3700         ADC #0
3710         STA SYMBOL+1
3720         LDA #SYMBOL
3730         STA STPNTR
3740         LDA /SYMBOL
3750         STA STPNTR+1
3760         LDA LINNUM   PUT LINE NUMBER INTO SYMBOL
3770         STA SYMBOL+3
3780         LDA LINNUM+1
3790         STA SYMBOL+2
3800         JMP CHAIN.SEARCH
3810  *---------------------------------
3820  PRINT.REPORT
3830         LDA #'A      START WITH A'S
3840  .1     STA VARNAM
3850         SEC
3860         SBC #'A      CONVERT TO HSHTBL INDEX
3870         ASL
3880         TAY
3890         LDA HSHTBL+1,Y
3900         BEQ .2       NO ENTRY FOR THIS LETTER
3910         STA PNTR+1
3920         LDA HSHTBL,Y
3930         STA PNTR
3940         JSR PRINT.LETTER.CHAIN
3950  .2     INC VARNAM   NEXT LETTER
```

```
3960        LDA VARNAM
3970        CMP #'Z+1
3980        BCC .1        STILL MORE LETTERS
3990        RTS           FINISHED
4000 *---------------------------------
4010 LTRDIG
4020        CMP #'0       DIGIT?
4030        BCC LD1       NO
4040        CMP #'9+1
4050        BCC LD2       YES
4060 LETTER
4070        CMP #'A       LETTER?
4080        BCC LD1       NO
4090        CMP #'Z+1
4100        BCC LD2       YES
4110        CLC           NO
4120 LD1    RTS
4130 LD2    SEC
4140        RTS
4150 *---------------------------------
4160 PRINT.LETTER.CHAIN
4170 .1     LDA VARNAM    FIRST LETTER
4180        JSR PRINT.CHAR
4190        LDY #1
4200 .2     INY
4210        LDA (PNTR),Y REST OF NAME
4220        AND #$7F
4230        CMP #'        BLANK?
4240        BEQ .3
4250        JSR PRINT.CHAR
4260 .3     CPY #3
4270        BCC .2
4280        LDA (PNTR),Y CHECK IF ARRAY
4290        BPL .4
4292        CMP #$AD      not array, but FN?
4294        BEQ .6
4300        LDA #'(
4310        JSR PRINT.CHAR
4320 .4     CLC           POINT AT LINE # CHAIN
4330        LDA PNTR
4340        ADC #4
4350        STA TPTR
4360        LDA PNTR+1
4370        ADC #0
4380        STA TPTR+1
4390        JSR PRINT.LINNUM.CHAIN
4400        JSR MON.CROUT
4410        LDY #1
4420        LDA (PNTR),Y POINTER TO NEXT VARIABLE
4430        BEQ .5        NO MORE
4440        PHA
4450        DEY
4460        LDA (PNTR),Y
4470        STA PNTR
```

```
4480         PLA
4490         STA PNTR+1
4500         BNE .1       ...ALWAYS
4510  .5     RTS
4511  .6     LDA #'F      add 'FN' after
4512         JSR PRINT.CHAR
4513         LDA #'N      variable name
4514         JSR PRINT.CHAR
4515         BNE .4       ...always
4520  *--------------------------------
4530  PRINT.LINNUM.CHAIN
4534         LDA #0       reset counter to 0
4538         STA $6       for each variable
4540  .1     JSR TAB.NEXT.COLUMN
4550         LDY #2       POINT AT LINE #
4560         LDA (TPTR),Y
4570         STA LINNUM+1
4580         INY
4590         LDA (TPTR),Y
4600         STA LINNUM
4610         JSR PRINT.LINE.NUMBER
4620         LDY #1       SET UP NEXT POINTER
4630         LDA (TPTR),Y
4640         BEQ .2
4650         PHA
4660         DEY
4670         LDA (TPTR),Y
4680         STA TPTR
4690         PLA
4700         STA TPTR+1
4710         BNE .1       ...ALWAYS
4720  .2     RTS
4730  *--------------------------------
4740  TAB.NEW.LINE
4750         JSR MON.CROUT
4760  TAB.NEXT.COLUMN
4770  .1     LDA #7       FIRST TAB STOP
4780  .2     CMP MON.CH   CURSOR POSITION
4790         BCS .3       PERFORM TAB
4800         ADC #6       NEXT TAB STOP
4810         CMP #33      END OF LINE?
4820         BCC .2
4821         INC $6       count the screen line
4822         LDA $6
4823         AND #1       look at odd-even bit
4824         BEQ TAB.NEW.LINE  both scrn and printer
4834         LDA #$8D
4835         JSR $FDF0    <CR> to screen only
4836         JMP .1       ...always
4840  .3     BEQ .4       ALREADY THERE
4850         SBC MON.CH   CALCULATE # OF BLANKS
4860         TAX
4870         JSR MON.PRBL2
4880  .4     RTS
```

```
4890  *--------------------------------
4900  PRINT.LINE.NUMBER
4910         LDX #4       PRINT 5 DIGITS
4920         STX LZFLAG   TURN ON LEADING ZERO FLAG
4930  .1     LDA #'0      DIGIT=0
4940  .2     PHA
4950         SEC
4960         LDA LINNUM
4970         SBC PLNTBL,X
4980         PHA
4990         LDA LINNUM+1
5000         SBC PLNTBH,X
5010         BCC .3       LESS THAN DIVISOR
5020         STA LINNUM+1
5030         PLA
5040         STA LINNUM
5050         PLA
5060         ADC #0       INCREMENT DIGIT
5070         BNE .2       ...ALWAYS
5080  .3     PLA
5090         PLA
5100         CMP #'0
5110         BEQ .5       ZERO, MIGHT BE LEADING
5120         SEC          TURN OFF LZFLAG
5130         ROR LZFLAG
5140  .4     JSR PRINT.CHAR
5150         DEX
5160         BPL .1
5170         RTS
5180  .5     BIT LZFLAG   LEADING ZERO FLAG
5190         BMI .4       NO
5200         LDA #'       BLANK
5210         BNE .4       ...ALWAYS
5220  PLNTBL .DA #1
5230         .DA #10
5240         .DA #100
5250         .DA #1000
5260         .DA #10000
5270  PLNTBH .DA /1
5280         .DA /10
5290         .DA /100
5300         .DA /1000
5310         .DA /10000
5320  *--------------------------------
5330  PRINT.CHAR
5340         ORA #$80
5350         JSR MON.COUT
5360         RTS
5370  *--------------------------------
5380  ZZ.END .EQ *
5390  ZZ.SIZ .EQ ZZ.END-ZZ.BEG
```

==============================================================================
DOCUMENT :AAL-8311:Articles:Aztec.C.txt
==============================================================================

A Look at the Aztec C Compiler for Apple DOS........Bill Morgan

As I mentioned last month, I'm getting very interested in the C language. That August issue of Byte definitely turned me on, so I've started to look at ways to get C into my Apple.

Byte featured a comparative review of several C compilers for CP/M. One of the highest-rated was the Aztec C Compiler System, which is also available for Apple DOS 3.3. The Aztec compiler was given especially high marks for being truly complete and compatible with the standard for C, the book "The C Programming Language", by Kernigan and Ritchie.

I haven't had a chance to actually do any programming with the Aztec system yet, but, thanks to Donna Lamb, a subscriber in New York City, I was able to spend an afternoon looking over the manual. Here are some of my impressions.

Manual

The manual is 135 pages long in 5 chapters and 2 appendices:

Tutorial Intro - 15pp - Getting started, configuring and using the SHELL, compiling, assembling, linking and executing. A get-your-toes-damp intro to the system.

Shell - 22pp - The SHELL program resides in the language card, at $D000-$F7FF. It replaces the Command Interpreter portion of DOS 3.3 and provides a UNIX-like user interface, including I/O redirection and command parsing with argument passing.

Programs - 23pp - Using the editor, compilers, assemblers, linker, and utilities.

Libraries - 33pp - Discussion of the Standard I/O, System I/O, Utility, and Math Routines supplied with the system.

Technical Info - 28pp - Miscellaneous information on the internals of the system and the assembly-language interface. Manx promises continuing additions to this chapter, as part of the updates.

Appendices - 12pp - Error messages and examples of the compiler and assembler outputs for a simple program.

DOS 3.3 Interface

The disks you receive from Manx do not include DOS, so to enter the system you must first boot DOS, then BRUN SHELL.

SHELL overlays the DOS Command Interpreter and patches at least two (unspecified) points inside the File Manager. All the documentation has to say about non-standard (i.e., fast) DOS's is "try it and see." I am told that Diversi-DOS does not work; I don't know about others.

Two Compilers for the Price of One

The Aztec system includes two separate compilers and two assemblers. There is a compiler/assembler pair for generating native 6502 code, and another compiler/assembler for an interpreted pseudo-code. The native code is fast but large, while the pseudo-code is slower but smaller. You can compile most of your program to pseudo-code, compile the time-critical parts to machine code, and write any extremely critical sections directly in assembly language. You can then link all these different object modules into one executable program.

Updates

The copy I saw was Version 1.05b of the Aztec system. Updates are available for an unspecified "nominal" fee, or an automatic update service is available for $50 per year.

Drawbacks

The people I have talked to who use the Aztec system regularly mention two drawbacks: compilation time and program size. Much of the compile time problem seems to be a matter of the Apple's disk speed, which can be improved.

The program size is related to the size of the run-time routines and the libraries included in a program. Experienced C programmers say that it is usually possible to manipulate the libraries to minimize the size of included code, but that is a fairly advanced technique.

ProDOS Version

There is supposed to be a ProDOS version of the Aztec system, which should be significantly faster, coming sometime. It's too soon to tell when that is likely to appear, so we'll just have to wait. The ProDOS version will be marketed as a completely separate version, rather than as an update to the DOS 3.3 version.

Conclusions

The Aztec C Compiler System is a full C compiler that runs in an Apple ][, and that makes it unique. Since my interest is in learning C and starting to develop programs that will be used on other, more powerful computers, I plan to place my order as soon as the ProDOS version is available.

All things considered, the Aztec system is not a great approach for developing applications intended only for use on Apple ][ computers. The Apple is simply too limited for full C.

Available from:  Manx Software Systems, Box 55, Shrewsbury, NJ 07701.
(201) 780-4004.

========================================================================
DOCUMENT :AAL-8311:Articles:Front.Page.txt
========================================================================

$1.50

Volume 4 -- Issue 2 November, 1983

In This Issue...



Tearing into ProDOS

Have we got a treat for you! You've heard about ProDOS, the new operating system for the Apple II's. Its main advantage over DOS 3.3 is speed, and on the next page of this issue you'll start to see what makes it so fast. ProDOS uses a completely different technique for translating between memory bytes and nibble-coded disk data, and here it is! Start reading Bob's completely commented disassembly.

Holiday Special Prices

Remember that we are offering special prices on several popular products from our list. Check the ad on page two for details. We are also having a sale on back issues of Apple Assembly Line: now only $1.00 each, rather than the usual $1.50. This is the time to complete your set! Subscription rates will be going up as of the first of the year, but you can still renew at the current prices. Let us hear from you.

Non-volatile RAM

Rodney Jacks, a Mostek engineer, tells us of a very interesting new chip: a 2K-byte static RAM, plug compatible with a 2716 EPROM, with a built-in lithium battery. Call your distributor and ask for Mostek MK48Z02. I can hardly wait to get some.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

=============================================================================
DOCUMENT :AAL-8311:Articles:Ideas....txt
=============================================================================

Possible articles for v4n2

Timemaster review ###

ProDOS class, features, prognosis ###

News about Apple II mouse, lack of sockets on future motherboards

Possiblity of extending Timemaster firmware by using PortA outputs to select EPROM page ###

Review of CHEAP Assembler

Notice about Dataphile Digest ###

Labelled GOTOs and GOSUBs ###

Visit with Tom Weishaar

Visit with Jack Lewis, Micromation
   look for exciting new stuff
   robot conference in spring

Anyone else run into trouble reading huge text files like Bob Nacon did?

=====================================================================
DOCUMENT :AAL-8311:Articles:Killing.Exec.txt
=====================================================================

Killing the EXEC....................................Bob Bragner
Istanbul, Turkey

Have you ever been at the beginning of the execution of a l-o-n-g EXEC file and realized you didn't really want to go through with it? There's not really much you can do. Control-C and RESET are ineffective even if you have an old Apple ][ without the Autostart ROM. On a //e you can hit Control-Open Apple-RESET, but at the expense of anything you may have in the Apple's memory -- a rather drastic solution.

As it turns out, there is a very easy way to terminate an EXEC file in progress. Apple DOS contains a single byte ($AAB3 when DOS is at its normal location) which is called "EXEC.STATUS". If the value of this byte is not 0 DOS thinks an EXEC file is in charge. If it is 0 then as far as DOS is concerned, no EXEC file is active. So we have the following little routine:

This routine can be reassembled to run anywhere. the INIT portion simply directs the RESET vector to the KILL.EXEC part of the routine and must be called before the EXEC command is issued. KILL.EXEC stores a 0 in the EXEC.STATUS flag and jumps to the DOS warm start at $3D0. Now if you hit RESET during an EXEC file's operation, the file will terminate politely.

Here is a series of POKES and a CALL that could be placed at the beginning of any EXEC program:

```
POKE 1010,13 : POKE 1011,3 : CALL 64367
POKE 781,169 : POKE 782,0 : POKE 783,141 : POKE 784,179
POKE 785,170 : POKE 786,76 : POKE 787,208 : POKE 788,3
(the rest of your program goes here)
```

This works from machine language, Integer BASIC, Applesoft, AND the S-C RAMcard Macro Assembler. The latter is a big help when you discover you're EXEC'ing the wrong 2000-line text file into the assembler, or you've forgotten to turn AUTO on!

[ Just a couple of comments: this trick won't work with an old non-Autostart ROM Apple, since you can't redirect RESET; and be sure to type the CLOSE command after the RESET, to free up the file buffer that the EXEC file was using. Bill ]

==============================================================================
DOCUMENT :AAL-8311:Articles:Lower.Case.Sq.txt
==============================================================================

Lower Case Titles Revisited.........................Bill Morgan

Last month we published Bob Matzinger's patch to Version 1.1 of the Macro Assembler to allow lower-case characters in a .TItle line. The article contained this sentence: "Here is a hex dump of the code, with a square around the byte to be changed:" But I forgot to draw the square on the page!

Here is that section of code again, this time with the square drawn in:

```
       A2 00     LDX #0
       20 3E x2  JSR $123E or $D23E
       C9 2C     CMP #$2C
       D0 0D     BNE ...

       20 3E x2  JSR $123E or $D23E

       B0 08     BCS ...
       9D 70 01  STA $170,X
```

```
=================================================================
DOCUMENT :AAL-8311:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 1.0...................................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50
Full Screen Editor for S-C Macro Assembler..........(reg. $49.00)  $40.00**
    Includes complete source code.
S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $40.00**

S-C Word Processor (the one we use!)..............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor.....(reg. $60.00)  $40.00**

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research).....(reg. $79.00)  $50.00**

Blank Diskettes..................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender..........................................$20.00
Shift-Key Modifier................................................$15.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Micro Cookbook, vol. 2", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
```

```
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00
  Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***

(** Special price to subscribers only through December 31, 1983.)
```

==========================================================================
DOCUMENT :AAL-8311:Articles:PDOs.Clk.Drvr.txt
==========================================================================

ProDOS and Clock Drivers, with a...........Bob Sander-Cederlof
Commented Listing of ProDOS $F142-$F1BE

ProDOS is a new operating system which Apple expects to release to the public during the first quarter of 1984. I am told that new computers and disk drives will be shipped with ProDOS rather than DOS 3.3. Version 1.0 is already available to licensed developers (I have it).

Apple has released massive amounts of documentation to licensed developers, and has even been offering a full day class at $225 per seat in various cities around the country. I attended the Dallas class on October 21st. Even with all the help they are giving, there are still a lot of unclear details that can only be illuminated by well-commented assembly listings of the actual ProDOS code. Apple will never publish these, so we will do it ourselves.

My first serious foray into ProDOS began at the request of Dan Pote, Applied Engineering. Dan wanted me to modify the firmware of his Timemaster clock card so that it automatically had full compatibility with ProDOS. Dan wanted all programs, even protected ones, which boot under ProDOS, to be able to read the date and time from his card. Also, he wanted ProDOS to time/date stamp the files in the directory with his card, just as it does with Thunderclock. (No small task, it turned out.)

ProDOS, when booting, searches the slots for a Thunderclock. If it finds one, it marks a bit in the machine ID byte (MACHID, bit 0 of $BF98 = 1); it plugs two bytes at $F14D and F150 with $CN, where N is the slot number; and it stores a JMP opcode ($4C) at $BF06.

$BF06 is a standard vector to whatever clock routine is installed. If no Thunderclock was found, an RTS opcode will be stored there.

The ProDOS boot slot search looks for these Thunderclock ID bytes:

```
       $CN00 = $08
       $CN02 = $28
       $CN04 = $58
       $CN08 = $70
```

After booting, ProDOS loads and executes the program called STARTUP. The standard STARTUP program searches the slots for various cards and displays a list of what it finds. Unfortunately this list seldom agrees with the true configuration in any of my computers. For one thing, STARTUP examines different bytes than the boot search does. In looking for a clock card, STARTUP wants:

```
       $CN00 = $08
       $CN01 = $78
```

```
       $CN02 = $28
```

If you do not have a Thunderclock, but do have some other clock, you have several options. What I did for Dan was change the firmware of Timemaster so that it emulates Thunderclock. ProDOS is convinced it has a Thunderclock, but you are saved the extra expense, and you gain extra features.

Another approach is to write a program which installs your own clock driver inside ProDOS. Mike Owen, of Austin, Texas, did this for Dan. After ProDOS boots it loads the first type SYS file it can find in the directory whose name ends with ".SYSTEM". Normally this is "BASIC.SYSTEM", which then proceeds to execute STARTUP. However, you can set up your disk with CLOCK.SYSTEM before BASIC.SYSTEM in the directory.

Write CLOCK.SYSTEM so that it begins at $2000, because all type SYS files load there. The program should mark the clock ID bit in MACHID, punch a JMP opcode at $BF06, and look at the address in $BF07,BF08. That address is the beginning of the clock driver inside the language card. Right now that address is $F142, but it could change.

Your program should write enable the language card by two "LDA $C081" instructions in a row, and then copy your clock driver into the space starting at that address. You can use up to 124 bytes. If your driver has references to the clock slot, be sure to modify them to the actual slot you are using. If your driver has internal references, be sure to modify them to point to the actual addresses inside the new physical location.

It is standard practice in peripheral firmware to use the following code to find out which slot the card is in:

```
       JSR $FF58     A Guaranteed $60 (RTS opcode)
       TSX           Stack pointer
       LDA $100,X    Get $CN off stack
```

Many cards also use "BIT $FF58" as a means for setting the V-bit in the status register. BE AWARE THAT ProDOS DOES NOT HAVE $60 AT $FF58 in the language card!!!!

The Thunderclock has two entries, at $CN08 and $CN0B, which assume that $CN is already in the X-register. $CN0B allows setting the clock mode, and $CN08 reads the clock in the current mode. The ProDOS driver calls on these two entries, as the following listing shows.

ProDOS maintains a full page at $BF00 called the System Global Page. The definition of this page should not change, ever. They say. Locations $BF90-BF93 contain the current date and time in a packed format. A system call will read the clock, if a driver is installed, and format the year-month-day-hour-minute into these four bytes.

Now here is a listing of the current Thunderclock driver, as labelled and commented by me.

================================================================================
DOCUMENT :AAL-8311:Articles:PDos.Disasm.Ex.txt
================================================================================

Commented Listing of ProDOS $F800-$F90B, $F996-FEBD
.....................Bob Sander-Cederlof

ProDOS boots its bulk into the RAM card, from $D000 thru $FFFF. More is loaded into the alternate $D000-DFFF space, and all but 255 bytes are reserved out of the entire 16K space.

A system global page is maintained from $BF00-BFFF, for various variables and linkage routines. All communication between machine language programs and ProDOS is supposed to be through MLI (Machine Language Interface) calls and the system global page.

One of the first things I did with ProDOS was to start dis-assembling and commenting it. I want to know what is inside and how it works! Apple's 4-inch thick binder tells a lot, but not all.

Right away I ran into a roadblock: to disassemble out of the RAM card it has to be turned on. There is no monitor in the RAM card when ProDOS is loaded. Turning on the RAM card from the motherboard monitor causes a loud crash!

I overcame most of the problem by copying a monitor into the $F800-FFFF region of the RAM card like this:

```
       *C089 C089 F800<F800.FFFFM
       *C083 C083
```

The double C089 write-enables the RAM card, while memory reads are still from the motherboard. The rest of the line copies a monitor up. The two C083's get me into the RAM card monitor, ready to type things like "D000LLLLLLLLLLL"

But what about dis-assemblies of the space between $F800 and $FFFF? For this I had to write a little move program. My program turned on the RAM card and copied $F800-FFFF down to $6800-6FFF. Then I BSAVEd it, and later disassembled it.

The code from $F800-FFFF is mostly equivalent to what is in DOS 3.3 from $B800-BFFF. First I found a read/write block subroutine, which calls an RWTS-like subroutine twice per block. (All ProDOS works with 512-byte blocks, rather than sectors; this is like Apple Pascal, and the Apple ///.)

The listing which follows shows the RWB and RWTS subroutines, along with the READ.ADDRESS and READ.SECTOR subroutines. Next month I plan to lay out the SEEK.TRACK and WRITE.SECTOR subroutines, as well as the interrupt and reset handling code.

The outstanding difference between ProDOS and DOS 3.3 disk I/O is speed. ProDOS is considerably faster. Most of the speed increase is due to handling the conversion between memory-bytes and disk-bytes on the fly. DOS pre-converted a 256-byte block into 342 bytes in a special buffer, and then wrote the 342 bytes; ProDOS forms the first 86 bytes of the disk data in a special buffer, writes them, and then proceeds to write the rest of the data directly from the caller's buffer. When reading, DOS read the 342 disk-bytes into a buffer for later decoding into the caller's buffer. ProDOS reads and decodes simultaneously directly into the caller's buffer. This is achieved by extensive use of tables and self-modifying code.

Not only is direct time saved by doing a lot less copying of buffers, but also the sector interleaving can be arranged so that only two revolutions are required to read all 8 blocks on a track.

I believe Apple Pascal uses the same technique, at least for reading.

Whoever coded ProDOS decided to hard-code some parameters which DOS used to keep in tables specified by the user. For example, the number which tells how long to wait for a drive motor to rev up used to be kept in a Device Characteristics Table (DCT). That value is now inside a "LDA #$E8" instruction at $F84F. That means that if you are using a faster drive you have to figure out how to patch and unpatch ProDOS to take advantage of it.

Another hard-coded parameter is the maximum block number. This is no longer part of the data on an initialized disk. It is now locked into the four instructions at $F807-F80D, at a maximum of 279. If you have a 40- or 70-track drive, you can only use 35. Speaking of tracks, the delay tables for track seeking are still used, but they are of course buried in this same almost-unreachable area. If you have a drive with faster track-to-track stepping, the table to change is at $FB73-FB84.

Calls to RWTS in DOS 3.3 involved setting up two tables, an IOB and a DCT. The IOB contained all the data about slot, drive, track, sector, buffer address, etc. The DCT was a 5-byte table with data concerning the drive. ProDOS RWB is called in an entirely different way. A fixed-position table located at $42-47 in page zero is set up with the command, slot, buffer address, and block number.

There are three valid commands, which I call test, read, and write. Test (0) starts up the indicated drive. If it is successful, a normal return occurs; if not, you get an error return (carry set, and (A) non-zero). Read (1) and write (2) are what you expect them to be. RWB has a very simple job: validate the call parameters in $42-47, convert block number to track and sector, and call RWTS twice (once for each sector of the block).

ProDOS RWTS expects the sector number in the A-register, and the track in a variable at $FB56. RWTS handles turning on the drive motor and waiting for it to come up to speed. RWTS then calls SEEK.TRACK to find the desired track, READ.ADDRESS to find the selected sector, and branches to READ.SECTOR or WRITE.SECTOR depending on the command.

READ.ADDRESS is virtually the same in ProDOS as it was in DOS 3.3. READ.SECTOR is entirely different. I should point out here that ProDOS diskettes are entirely compatible with Apple /// diskettes. The file structures are exactly the same. Both ProDOS and Apple /// diskettes use the same basic recording techniques on the disk as DOS 3.3, so the diskettes are copyable using standard DOS 3.3 copiers such as the COPYA program on your old System Master Diskette.

READ.SECTOR begins by computing several addresses and plugging them into the code further down. (This enables the use of faster addressing modes, saving enough cycles to leave time for complete decoding of disk data on the fly.) First the disk slot number is merged into the instructions which read bytes from the drive. Next the caller's buffer address is put into the store instructions.

Note that the byte from the disk is loaded into the X-register, then used to index into BYTE.TABLE, at $F996, to get the equivalent 6-bit data value. Since a disk byte may only have certain values, there is some space within BYTE.TABLE that will never be accessed. Most of this unused space contains $FF bytes, but some of it is used for other small tables: BIT.PAIR.LEFT, .MIDDLE, and .RIGHT, and DATA.TRAILER. These are used by WRITE.SECTOR, which we'll look at next month.

Your buffer is divided into three parts: two 86-byte chunks, and one of 84 bytes. Data coming from the disk is in four chunks: three of 86 bytes, and one of 84.

The first chunk contains the lower two bits from every byte in the original data. READ.SECTOR reads this chunk into TBUF, so that the bits will be available later for merging with the upper six of each byte. ($FC53-FC68)

The second chunk contains the upper six bits from the first 86 bytes of the original data. $FC69-FC83 reads the chunk and merges in the lower two bits from TBUF, storing the completed bytes in the first 85 bytes of the caller's buffer. The last (86th) byte is saved on the stack (I am not sure why), and not stored in the caller's buffer until after all the rest of the data has been read.

A tricky manipulation is necessary to merge in those lower two bits. The data in TBUF has those bits in backward order, packed together with the bits from the other chunks. There was a good diagram of this on page 10 of the June 1981 issue of AAL. DOS merged them with a complex time-consuming shifting process. ProDOS does a swift table lookup, using the TBUF byte as an index to the BIT.PAIR.TABLE.

BIT.PAIR.TABLE has four bytes per row. The first three in each row supply the bit pairs; the fourth is used by SECTOR.WRITE to encode data, and will be covered next month.

When $FC69-FC83 is reading the first chunk, the first byte in each row is used to supply the lower two data bits. The byte in TBUF corresponding to the current position in the chunk selects a byte from BIT.PAIR.TABLE, and the two parts are merged together.

The next two chunks are handled just like the one I just described. After all the data has been read, READ.SECTOR expects to have accumulated a checksum of 00, and expects to find a trailing $EB after the data.  Return with carry clear indicates all went well; carry set indicates a read error (bad checksum, missing header, or missing trailer).

I can't help wondering:  can this fast read technique be fit into DOS 3.3?  It takes a little more code and table space, but on the other hand it uses 256 bytes less of intermediate buffer.  If we sacrificed the INIT command, could both the fast read and write be squeezed into DOS 3.3?

For more good information on ProDOS, be sure to take a look at Tom Weishaar's DOStalk column in the current issue of Softalk.

================================================================================
DOCUMENT :AAL-8311:Articles:Qwerty.Review.txt
================================================================================

Qwerty 68000 Training/Development System...Bob Sander-Cederlof

There is now a plethora of 68000 boards designed to fit inside, or nearly inside, your Apple. Names like DTACK Grounded, PDQ, Saybrook, and Acorn.

Most of these are aimed at hot-rodding your Apple. Some come with the UCSD p-System, including Pascal and an Applesoft- compatible BASIC and much more. Others have a more limited selection. Most are too costly for most of us, around $1500.

Motorola and others sell development systems based on the 68000 for $10K-30K. The Apple Lisa makes an excellent development system, at $6995 plus the developer's software kit (when it becomes available).

"Wait a minute! I don't even have a spare $1500, let alone $10K! And I want to get my feet wet first, before diving in over my head!"

"In fact, I want to try my hand at learning 68000 assembly language first. I need an assembler, some books, and a monitor with step and trace commands. I would like a hands-on tutorial I can work though at my own pace."

"I can't afford to lay out more than $750 right now. But I want an expandable system, that can grow with my knowledge and needs."

Guess what...somebody overheard our thoughts! Jerry Hansen and Lane Hauck, of Qwerty Inc., have put together a package deal too good to resist: a complete integrated training and software development package for only $695.

The package includes a card to plug in any slot of your Apple II, II Plus, or //e; a reference manual which leads you through the details of the card, their firmware, and the assembler; a full-fledged macro assembler; the best three reference books, with other booklets and reference cards. You can use the books in a hands-on tutorial fashion, mastering the 68000 assembly language as you go.

The Q-68 card is the heart of the package. It is a compact, well-crafted design, with a 68008 microprocessor, 2K bytes of RAM, and 8K bytes of EPROM. The full Apple address-space can be addressed by the 68008 as well, including any memory expansion cards you may have. RAM can be expanded on-board to 8K, and EPROM to 32K. A 50-pin expansion connector allows connection of additional memory, to a total of 1 megabyte.

You don't need any external power supply or chassis. The card draws a maximum of 400 milliamps. (While this is more than Apple will recommend, it seems to be well within the capability of the Apple power supply.)  If you don't already have a cooling fan, you will probably want one after installing this card.  The 68008 is the main power user, which fact makes me ever-so-hungry for a CMOS version. The 68008 is a trimmed-down version of the 68000, with an 8-bit data bus.  The instruction set is unchanged, but it comes in a smaller package:  fewer pins, fewer milliamps, fewer dollars.  On the Q-68 board, the 68008 is clocked at 7.16 MHz.

The Apple 6502 keeps running while the 68008 is executing code; when the 68008 refers to Apple memory, the 68008 slows down to wait for the Apple bus, and the Apple slows to half speed during that cycle.  True multiprocessing is possible.

The Q-68 EPROM is loaded with good things.  You get a comprehensive self-test facility, and an easy-to-use debugging monitor.  The debugging monitor allows you to step and trace through your programs, and set breakpoints.  There are five different display windows you can cycle through with a single keystroke:  Register, Memory, Disassembly, and Breakpoint displays, and a helpful Command Summary.

Qwerty is aiming primarily at the those of us who want to learn 68000 programming and/or develop 68000 software without investing in an expensive complete 68000 system.  However, there are many other exciting possibilities for this board.  Those of you who really do want to speed up your Apple can certainly write code for the purpose. (Or maybe adapt public domain code already written for other 68K boards.)  The Q-68 card may be used as a powerful controller or co-processor with your yet-to-be-written software.  You can connect the Q-68 to the outside world directly, as well as through the Apple bus.

Now for something truly unique:  the package comes with a special version of the S-C 68000 Cross Assembler.  The S-C manual has been re-written to give 68000 code examples throughout.  New commands have been added to start the Q-68 card, either in debug mode or at full speed.  Three versions are included to provide different memory usage options.

What you get is a near optimum environment both for learning and for serious software development.  Gone are the "load the editor, load-edit-save the source program, load the assembler, assemble, load the loader, load the object program, run into a bug, load the editor...." blues.  With this package you simply edit, assemble, and run directly from RAM.

Programs too large for RAM can be assembled and loaded using multiple source and object files when necessary, but you still never need to reload the editor/assembler or monitor/debugger.

Current users of the S-C Assembler family already know the commands and editing techniques.  You can concentrate on learning the 68000 itself, and the Qwerty debugger, without being distracted by a whole new operating system.  (Later, when you can afford a Lisa or MacIntosh, you will already know the language and can concentrate on learning the operating system.)

Here is another new twist: Qwerty offers a free 30-day trial period. If you're not happy with the package for any reason, you can return it within 30 days in salable condition for a full refund. Qwerty, Inc. Phone (619) 569-5283.

```
==============================================================================
DOCUMENT :AAL-8311:Articles:Shapemaker.Enh.txt
==============================================================================
```

Shapemaker Enhancements....................Bob Sander-Cederlof

Frank Belanger sent me a new updated version of his Shapemaker Utility.  He says it is now the best program of its type on the market, and he is really proud of it.  Here are the new features:

```
*   Clearer, more accessible HELP screens.
*   RENUMBER command in the Shape Editor.
*   Two grid sizes:  18x30 and 24x40.
*   Hi-Res Dump for Epson printer, accessible
    both in Shapemaker and with an &-command.
*   Four new typefaces (total now 9).
*   Manual now 55 pages long.
```

Shapemaker is still just $35, from Frank at 4200 Avenue B, Austin, TX 78751.

==============================================================================
DOCUMENT :AAL-8311:Articles:Shorts.txt
==============================================================================

The Computer Hacker and Dataphile Digest

I received Vol 1 No 2 of the "Computer Hacker", and I think it will be a useful newsletter.  As the magazines become more and more general, filled with reviews of packaged systems and software, we will have to look elsewhere for articles that get down to the nitty-gritty.  Even local club newsletters are steering away from the hobbyist's or technician's needs.

The issue I have includes listings of a pair of programs to transfer data from one computer to another in the CP/M environment; part two of a detailed explanation of the RS-232 "standard"; part one of directions for building a hardware print spooler; a review of floppy disk formats; an Apple (6502) assembly language program for sending Morse code; and a beginner's introduction to electronics.

The Computer Hacker, 12 issues per year for $24, P. O. Box 1697, Kalispell, MT 59903.

Dataphile Digest is a monthly survey of Apple related periodicals. Bill & Shannon Bailey scan more than a dozen magazines each month, and write brief descriptions of each article relating to Apple computers. They organize the descriptions into categories that make it easy to find any topic you like.  The second issue covered one or two issues of 14 different magazines, and included 840 entries organized into 38 categories.

Dataphile Digest is typeset, and printed the same size as Apple Assembly Line.  The current issue is 78 pages (plus cover and contents pages), and bears a cover price of $3.50.  No subscription price is given, so I would suggest writing to them at P. O. Box 2806, Del Mar, CA 92014.  Or call at (619) 436-9382.

```
==============================================================================
DOCUMENT :AAL-8311:Articles:XAsm.6301.txt
==============================================================================
```

Hitachi 6301 Cross Support.................Bob Sander-Cederlof

As you probably know, we have a growing line of cross assemblers available. You can use your Apple as a development system without ever learning another editor/assembler/operating-system, on any of ten or more different chips.

It all started back in 1980 when Nigel Nathan paid me to create a 6801 cross assembler based on version 4.0 of the S-C Assembler II. Later Bob Urschel bought a copy. Back then we thought $300 a copy was a pretty good price.

All our competition in this field seems to agree. Avocet charges $200 or more per cross assembler. Byte magazine carries several ads showing prices for cross assemblers between $395 and $1000 apiece. Our assemblers are just as good, and many of you tell us ours are easier to use and more powerful. But we charge either $32.50 or $50 apiece, after you own the $80 S-C Macro Assembler.

Until very recently, the 6800/1/2 Macro Cross Assembler came with only one version on the disk. This one version assembled all of the opcodes of the 6801 chip. If you were programming for a 6800, which did not support all of those opcodes and addressing modes, it was a little dangerous. Last month we upgraded this disk by making two versions: one for 6800 only, and one for 6801.

Now I have added a third version for the Hitachi 6301. The 6301 is a CMOS chip, includes all the opcodes of the 6801, and adds six more:

```
       XGDX    Exchange D and X
       SLP     Sleep (reduced power mode)
       AIM     And Immediate into Memory
       OIM     Or Immediate into Memory
       EIM     Exclusive Or Immediate into Memory
       TIM     Test Memory Immediate
```

The last four each have two addressing modes. You can write "AIM #val,addr" or "AIM #val,addr,X". In both modes the address is only 8 bits. You can see that AIM lets you clear any bits in a memory byte; OIM lets you set any bits in a byte; EIM lets you toggle any bits; and TIM lets you test any bits. TIM forms the logical product (AND) of the memory byte and the immediate value, and tests for sign and zero.

The 6301 includes extensive memory mapped I/O on the chip, mapped into the zero page. With these "xIM" opcodes you have an extremely powerful I/O capability.

If you have the older disk of the 6800/1/2 cross assembler, and want to upgrade to get the 6301 version, send $5.

```
==============================================================================
DOCUMENT :AAL-8311:DOS3.3:PDOS.F142.F1Be.txt
==============================================================================

 1000  *SAVE S.PRODOS $F142...$F1BE
 1010  *---------------------------------
 1020  *  IF THE PRODOS BOOT RECOGNIZES A THUNDERCLOCK,
 1030  *  A "JMP $F142" IS INSTALLED AT $BF06 AND
 1040  *  THE SLOT ADDRESS IS PATCHED INTO THE FOLLOWING
 1050  *  CODE AT SLOT.A AND SLOT.B BELOW.
 1060  *---------------------------------
 1070  DATE   .EQ $BF90    $BF91 = YYYYYYYM
 1080  *                   $BF90 = MMMDDDDD
 1090  TIME   .EQ $BF92    $BF93 = 000HHHHH
 1100  *                   $BF92 = 00MMMMMM
 1110  MODE   .EQ $5F8-$C0 THUNDERCLOCK MODE IN SCREEN HOLE
 1120  *---------------------------------
 1130         .OR $F142
 1140         .TA $800
 1150  *---------------------------------
 1160  PRODOS.THUNDERCLOCK.DRIVER
 1170         LDX SLOT.B   $CN
 1180         LDA MODE,X   SAVE CURRENT THUNDERCLOCK MODE
 1190         PHA
 1200         LDA #$A3     SEND "#" TO THUNDERCLOCK TO
 1210         JSR $C20B       SELECT INTEGER MODE
 1220  SLOT.A .EQ *-1
 1230  *---------------------------------
 1240  *      READ TIME & DATE INTO $200...$211 IN FORMAT:
 1250  *---------------------------------
 1260         JSR $C208
 1270  SLOT.B .EQ *-1
 1280  *---------------------------------
 1290  *      CONVERT ASCII VALUES TO BINARY
 1300  *      $3E -- MINUTE
 1310  *      $3D -- HOUR
 1320  *      $3C -- DAY OF MONTH
 1330  *      $3B -- DAY OF WEEK
 1340  *      $3A -- MONTH
 1350  *---------------------------------
 1360         CLC
 1370         LDX #4
 1380         LDY #12      POINT AT MINUTE
 1390  .1     LDA $200,Y   TEN'S DIGIT
 1400         AND #$07     IGNORE TOP BIT
 1410         STA $3A      MULTIPLY DIGIT BY TEN
 1420         ASL          *2
 1430         ASL          *4
 1440         ADC $3A      *5
 1450         ASL          *10
 1460         ADC $201,Y   ADD UNIT'S DIGIT
 1470         SEC
 1480         SBC #$B0     SUBTRACT ASCII ZERO
```

```
1490        STA $3A,X    STORE VALUE
1500        DEY          BACK UP TO PREVIOUS FIELD
1510        DEY
1520        DEY
1530        DEX          BACK UP TO PREVIOUS VALUE
1540        BPL .1       ...UNTIL ALL 5 FIELDS CONVERTED
1550 *--------------------------------
1560 *      PACK MONTH AND DAY OF MONTH,
1570 *--------------------------------
1580        TAY          MONTH (1...12)
1590        LSR          00000ABC--D
1600        ROR          D00000AB--C
1610        ROR          CD00000A--B
1620        ROR          BCD00000--A
1630        ORA $3C      MERGE DAY OF MONTH
1640        STA DATE     SAVE PACKED DAY AND MONTH
1650        PHP          SAVE TOP BIT OF MONTH
1660 *--------------------------------
1670 *      CONVERT MONTH, DAY OF MONTH,
1680 *      AND DAY OF WEEK INTO YEAR.
1690 *--------------------------------
1700        AND #$1F     ISOLATE DAY OF MONTH (1...31)
1710 *      CARRY SET FOR MONTHS 8...12
1720        ADC YEAR.DAY,Y    COMPUTE DAY OF YEAR
1730        BCC .2
1740        ADC #3    ADJUST REMAINDER FOR YEARDAY > 255
1750 .2     SEC          GET REMAINDER MODULO 7
1760 .3     SBC #7
1770        BCS .3       ...UNTIL ALL 7'S REMOVED
1780        ADC #7       RESTORE TO POSITIVE VALUE
1790        SBC $3B      SUBTRACT KNOWN DAY OF WEEK
1800        BCS .4       NO BORROW
1810        ADC #7       BORROWED, SO ADD 7 BACK
1820 .4     TAY          ADJUSTED DAY OW WEEK AS INDEX
1830        LDA YRTBL,Y  GET YEAR (82...87)
1840        PLP          GET HIGH BIT OF MONTH IN CARRY
1850        ROL          FORM YYYYYYYM
1860        STA DATE+1
1870        LDA $3D      GET HOUR
1880        STA TIME+1
1890        LDA $3E      GET MINUTE
1900        STA TIME
1910        PLA          RESTORE THUNDERCLOCK MODE
1920        LDX SLOT.B   GET $CN FOR INDEX
1930        STA MODE,X
1940        RTS
1950 *--------------------------------
1960 YEAR.DAY   .EQ *-1  OFFSET BECAUSE INDEX 1...12
1970    .DA #0,#31,#59,#90      JAN,FEB,MAR,APR
1980    .DA #120,#151,#181,#211 MAY,JUN,JUL,AUG
1990    .DA #242,#20,#51,#81    SEP,OCT,NOV,DEC
2000 *--------------------------------
2010 YRTBL  .DA #84,#84,#83,#82,#87,#86,#85
2020 *--------------------------------
```

```
===============================================================================
DOCUMENT :AAL-8311:DOS3.3:PDos.F800.FFFF.txt
===============================================================================

 1000        .TI 76,PRODOS F800-FFFF.....COMMENTED BY RBS-C  11-8-
83............
 1010  *SAVE S.PRODOS F800-FFFF
 1020  *--------------------------------
 1030  RUNNING.SUM        .EQ $3A
 1040  TBUF.0             .EQ $3A
 1050  BYTE.AT.BUF00      .EQ $3B
 1060  BYTE.AT.BUF01      .EQ $3C
 1070  LAST.BYTE          .EQ $3D
 1080  SLOT.X16           .EQ $3E
 1090  INDEX.OF.LAST.BYTE .EQ $3F
 1100  *--------------------------------
 1110  RWB.COMMAND .EQ $42
 1120  RWB.SLOT    .EQ $43   DSSSXXXX
 1130  RWB.BUFFER  .EQ $44,45
 1140  RWB.BLOCK   .EQ $46,47   0...279
 1150  *--------------------------------
 1160  BUFF.BASE .EQ $4700 DUMMY ADDRESS FOR ASSEMBLY ONLY
 1170  *--------------------------------
 1180  SAVE.LOC45 .EQ $BF56
 1190  SAVE.D000  .EQ $BF57
 1200  INTAREG    .EQ $BF88
 1210  INTBANKID  .EQ $BF8D
 1220  IRQXIT.3   .EQ $BFD3
 1230  *--------------------------------
 1240  DRV.PHASE  .EQ $C080
 1250  DRV.MTROFF .EQ $C088
 1260  DRV.MTRON  .EQ $C089
 1270  DRV.ENBL.0 .EQ $C08A
 1280  DRV.Q6L    .EQ $C08C
 1290  DRV.Q6H    .EQ $C08D
 1300  DRV.Q7L    .EQ $C08E
 1310  DRV.Q7H    .EQ $C08F
 1320  *--------------------------------
 1330  *                 <<<COMPUTED >>>
 1340  MODIFIER .EQ $60  <<<SLOT * 16>>>
 1350  *--------------------------------
 1360        .OR $F800
 1370        .TA $800
 1380  *--------------------------------
 1390  *      READ/WRITE A BLOCK
 1400  *
 1410  *      1.  ASSURE VALID BLOCK NUMBER (0...279)
 1420  *      2.  CONVERT BLOCK NUMBER TO TRACK/SECTOR
 1430  *          TRACK = INT(BLOCK/8)
 1440  *          BLOCK   SECTORS
 1450  *          -----  ---------
 1460  *            0     0 AND 2
 1470  *            1     4 AND 6
```

```
*            2     8 AND 10
*            3    12 AND 14
*            4     1 AND 3
*            5     5 AND 7
*            6     9 AND 11
*            7    13 AND 15
*      3.  CALL RWTS TWICE
*      4.  RETURN WITH ERROR STATUS
*---------------------------------
RWB
       LDA RWB.BLOCK     BLOCK MUST BE 0...279
       LDX RWB.BLOCK+1
       STX RWTS.TRACK
       BEQ .1       ...BLOCK # LESS THAN 256
       DEX
       BNE .5       ...BLOCK # MORE THAN 511
       CMP #$18
       BCS .5       ...BLOCK # MORE THAN 279
.1     LDY #5       SHIFT 5 BITS OF TRACK #
.2     ASL              RWTS.TRACK   A-REG
       ROL RWTS.TRACK   ----------   --------
       DEY               00TTTTTT    ABC00000
       BNE .2
       ASL          TRANSFORM BLOCK # INTO SECTOR #
       BCC .3       ABC00000 --> 0000BC0A
       ORA #$10
.3     LSR
       LSR
       LSR
       LSR
       PHA
       JSR RWTS     R/W FIRST SECTOR OF BLOCK
       PLA
       BCS .4       ...ERROR
       INC RWB.BUFFER+1
       ADC #2
       JSR RWTS     R/W SECOND SECTOR OF BLOCK
       DEC RWB.BUFFER+1
.4     LDA RWTS.ERROR
       RTS
*---BLOCK NUMBER > 279-----------
.5     LDA #$27     I/O ERROR
       SEC
       RTS
*---------------------------------
*      READ/WRITE A GIVEN SECTOR
*---------------------------------
RWTS
       LDY #1       TRY SEEKING TWICE
       STY SEEK.COUNT
       STA RWTS.SECTOR
       LDA RWB.SLOT
       AND #$70     0SSS0000
       STA SLOT.X16
```

```
2020        JSR WAIT.FOR.OLD.MOTOR.TO.STOP
2030        JSR CHECK.IF.MOTOR.RUNNING
2040        PHP          SAVE ANSWER (.NE. IF RUNNING)
2050        LDA #$E8     MOTOR STARTING TIME
2060        STA MOTOR.TIME+1  ONLY HI-BYTE NECESSARY
2070        LDA RWB.SLOT      SAME SLOT AND DRIVE?
2080        CMP OLD.SLOT
2090        STA OLD.SLOT
2100        PHP               SAVE ANSWER
2110        ASL               DRIVE # TO C-BIT
2120        LDA DRV.MTRON,X   START MOTOR
2130        BCC .1            ...DRIVE 0
2140        INX               ...DRIVE 1
2150  .1    LDA DRV.ENBL.0,X  ENABLE DRIVE X
2160        PLP               SAME SLOT/DRIVE?
2170        BEQ .3            ...YES
2180        PLP          DISCARD ANSWER ABOUT MOTOR GOING
2190        LDY #7       DELAY 150-175 MILLISECS
2200  .2    JSR DELAY.100     DELAY 25 MILLISECS
2210        DEY
2220        BNE .2
2230        PHP          SAY MOTOR NOT ALREADY GOING
2240  .3    LDA RWB.COMMAND   0=TEST, 1=READ, 2=WRITE
2250        BEQ .4            ...0, MERELY TEST
2260        LDA RWTS.TRACK
2270        JSR SEEK.TRACK
2280  .4    PLP          WAS MOTOR ALREADY GOING?
2290        BNE .6       ...YES
2300  .5    LDA #1       DELAY 100 USECS
2310        JSR DELAY.100
2320        LDA MOTOR.TIME+1
2330        BMI .5       ...WAIT TILL IT OUGHT TO BE
2340        JSR CHECK.IF.MOTOR.RUNNING
2350        BEQ .14      ...NOT RUNNING YET, ERROR
2360  .6    LDA RWB.COMMAND
2370        BEQ .17      CHECK IF WRITE PROTECTED
2380        LSR          .CS. IF READ, .CC. IF WRITE
2390        BCS .7       ...READ
2400        JSR PRE.NYBBLE   ...WRITE
2410  .7    LDY #64      TRY 64 TIMES TO FIND THE SECTOR
2420        STY SEARCH.COUNT
2430  .8    LDX SLOT.X16
2440        JSR READ.ADDRESS
2450        BCC .10      ...FOUND IT
2460  .9    DEC SEARCH.COUNT
2470        BPL .8       ...KEEP LOOKING
2480        LDA #$27     I/O ERROR CODE
2490        DEC SEEK.COUNT   ANY TRIES LEFT?
2500        BNE .14      ...NO, I/O ERROR
2510        LDA CURRENT.TRACK
2520        PHA
2530        ASL          SLIGHT RE-CALIBRATION
2540        ADC #$10
2550        LDY #64      ANOTHER 64 TRIES
```

```
2560        STY SEARCH.COUNT
2570        BNE .11       ...ALWAYS
2580 .10    LDY HDR.TRACK      ACTUAL TRACK FOUND
2590        CPY CURRENT.TRACK
2600        BEQ .12            FOUND THE RIGHT ONE
2610        LDA CURRENT.TRACK WRONG ONE, TRY AGAIN
2620        PHA
2630        TYA           STARTING FROM TRACK FOUND
2640        ASL
2650 .11    JSR UPDATE.TRACK.TABLE
2660        PLA
2670        JSR SEEK.TRACK
2680        BCC .8        ...ALWAYS
2690 .12    LDA HDR.SECTOR
2700        CMP RWTS.SECTOR
2710        BNE .9
2720        LDA RWB.COMMAND
2730        LSR
2740        BCC .15            ...WRITE
2750        JSR READ.SECTOR    ...READ
2760        BCS .9             ...READ ERROR
2770 .13    LDA #0        NO ERROR
2780        .HS D0        "BNE"...NEVER, JUST SKIPS "SEC"
2790 .14    SEC           ERROR
2800        STA RWTS.ERROR     SAVE ERROR CODE
2810        LDA DRV.MTROFF,X   STOP MOTOR
2820        RTS                RETURN
2830 *--------------------------------
2840 .15    JSR WRITE.SECTOR
2850 .16    BCC .13       ...NO ERROR
2860        LDA #$2B      WRITE PROTECTED ERROR CODE
2870        BNE .14       ...ALWAYS
2880 .17    LDX SLOT.X16 CHECK IF WRITE PROTECTED
2890        LDA DRV.Q6H,X
2900        LDA DRV.Q7L,X
2910        ROL
2920        LDA DRV.Q6L,X
2930        JMP .16       GIVE ERROR IF PROTECTED
2940 *--------------------------------
2950 SEEK.TRACK
2960        ASL           GET PHYSICAL TRACK #
2970        STA HDR.TRACK      ...SAVE HERE
2980        JSR CLEAR.PHASES   (CARRY WAS CLEAR)
2990        JSR GET.SSSD.IN.X
3000        LDA OLD.TRACK.TABLE,X
3010        STA CURRENT.TRACK
3020        LDA HDR.TRACK
3030        STA OLD.TRACK.TABLE,X
3040        JSR SEEK.TRACK.ABSOLUTE
3050 *--------------------------------
3060 CLEAR.PHASES
3070        LDY #3
3080 .1     TYA
3090        JSR PHASE.COMMANDER
```

```
3100        DEY
3110        BPL .1
3120        LSR CURRENT.TRACK    BACK TO LOGICAL TRACK #
3130        CLC          SIGNAL NO ERROR
3140        RTS
3150 *--------------------------------
3160 SEEK.TRACK.ABSOLUTE
3170        STA TARGET.TRACK  SAVE ACTUAL TRACK #
3180        CMP CURRENT.TRACK ALREADY THERE?
3190        BEQ .7            ...YES
3200        LDA #0
3210        STA STEP.CNT      # STEPS SO FAR
3220 .1     LDA CURRENT.TRACK
3230        STA CURRENT.TRACK.OLD
3240        SEC
3250        SBC TARGET.TRACK
3260        BEQ .6            ...WE HAVE ARRIVED
3270        BCS .2            CURRENT > DESIRED
3280        EOR #$FF          CURRENT < DESIRED
3290        INC CURRENT.TRACK
3300        BCC .3            ...ALWAYS
3310 .2     ADC #$FE          .CS., SO A=A-1
3320        DEC CURRENT.TRACK
3330 .3     CMP STEP.CNT GET MINIMUM OF:
3340        BCC .4         1. # OF TRACKS TO MOVE LESS 1
3350        LDA STEP.CNT   2. # OF STEPS SO FAR
3360 .4     CMP #9         3. EIGHT
3370        BCS .5
3380        TAY
3390        SEC          TURN NEW PHASE ON
3400 .5     JSR .7
3410        LDA ONTBL,Y  DELAY
3420        JSR DELAY.100
3430        LDA CURRENT.TRACK.OLD
3440        CLC          TURN OLD PHASE OFF
3450        JSR PHASE.COMMANDER
3460        LDA OFFTBL,Y DELAY
3470        JSR DELAY.100
3480        INC STEP.CNT # OF STEPS SO FAR
3490        BNE .1       ...ALWAYS
3500 .6     JSR DELAY.100
3510        CLC          TURN PHASE OFF
3520 .7     LDA CURRENT.TRACK
3530 *--------------------------------
3540 *      (A) = TRACK #
3550 *      .CC. THEN PHASE OFF
3560 *      .CS. THEN PHASE ON
3570 *--------------------------------
3580 PHASE.COMMANDER
3590        AND #3       ONLY KEEP LOWER TWO BITS
3600        ROL               00000XXC
3610        ORA SLOT.X16      0SSS0XXC
3620        TAX
3630        LDA DRV.PHASE,X
```

```
3640        LDX SLOT.X16      RESTORE SLOT*16
3650        RTS
3660 *--------------------------------
3670 *      VALUE READ FROM DISK IS INDEX INTO THIS TABLE
3680 *      TABLE ENTRY GIVES TOP 6 BITS OF ACTUAL DATA
3690 *
3700 *      OTHER DATA TABLES ARE IMBEDDED IN THE UNUSED
3710 *      PORTIONS OF THE BYTE.TABLE
3720 *--------------------------------
3730 BYTE.TABLE .EQ *-$96
3740        .HS 0004FFFF080CFF101418
3750 BIT.PAIR.LEFT
3760        .HS 008040C0
3770        .HS FFFF1C20FFFFFF24282C
3780        .HS 3034FFFF383C4044
3790        .HS 484CFF5054585C606468
3800 BIT.PAIR.MIDDLE
3810        .HS 00201030
3820 DATA.TRAILER
3830        .HS DEAAEBFF
3840        .HS FFFFFF6CFF70
3850        .HS 7478FFFFFF7CFFFF
3860        .HS 8084FF888C9094989CA0
3870 BIT.PAIR.RIGHT
3880        .HS 0008040C
3890        .HS FFA4A8ACFFB0B4B8BCC0
3900        .HS C4C8FFFFCCD0D4D8
3910        .HS DCE0FFE4E8ECF0F4
3920        .HS F8FC
3930 *--------------------------------
3940 BIT.PAIR.TABLE
3950        .HS 00000096
3960        .HS 02000097
3970        .HS 0100009A
3980        .HS 0300009B
3990        .HS 0002009D
4000        .HS 0202009E
4010        .HS 0102009F
4020        .HS 030200A6
4030        .HS 000100A7
4040        .HS 020100AB
4050        .HS 010100AC
4060        .HS 030100AD
4070        .HS 000300AE
4080        .HS 020300AF
4090        .HS 010300B2
4100        .HS 030300B3
4110        .HS 000002B4
4120        .HS 020002B5
4130        .HS 010002B6
4140        .HS 030002B7
4150        .HS 000202B9
4160        .HS 020202BA
4170        .HS 010202BB
```

```
        .HS 030202BC
        .HS 000102BD
        .HS 020102BE
        .HS 010102BF
        .HS 030102CB
        .HS 000302CD
        .HS 020302CE
        .HS 010302CF
        .HS 030302D3
        .HS 000001D6
        .HS 020001D7
        .HS 010001D9
        .HS 030001DA
        .HS 000201DB
        .HS 020201DC
        .HS 010201DD
        .HS 030201DE
        .HS 000101DF
        .HS 020101E5
        .HS 010101E6
        .HS 030101E7
        .HS 000301E9
        .HS 020301EA
        .HS 010301EB
        .HS 030301EC
        .HS 000003ED
        .HS 020003EE
        .HS 010003EF
        .HS 030003F2
        .HS 000203F3
        .HS 020203F4
        .HS 010203F5
        .HS 030203F6
        .HS 000103F7
        .HS 020103F9
        .HS 010103FA
        .HS 030103FB
        .HS 000303FC
        .HS 020303FD
        .HS 010303FE
        .HS 030303FF
*--------------------------------
TBUF   .BS 86
*--------------------------------
RWTS.TRACK    .HS 07
RWTS.SECTOR   .HS 0F
RWTS.ERROR    .HS 00
OLD.SLOT      .HS 60
CURRENT.TRACK .HS 07
              .HS 00
*--------------------------------
OLD.TRACK.TABLE .EQ *-4
       .HS 0000     SLOT 2, DRIVE 0--DRIVE 1
       .HS 0000     SLOT 3
```

```
        .HS 0000     SLOT 4
        .HS 0000     SLOT 5
        .HS 0E00     SLOT 6
        .HS 0000     SLOT 7
*--------------------------------
        .HS 00
*--------------------------------
SEARCH.COUNT      .BS 1
SEEK.COUNT        .BS 1
STEP.CNT      .EQ *
SEEK.D5.CNT   .EQ *
X1X1X1X1          .BS 1   ALSO STEP.CNT & SEEK.D5.CNT
CHECK.SUM         .BS 1
HDR.CHKSUM        .BS 1
HDR.SECTOR        .BS 1
HDR.TRACK     .EQ *
MOTOR.TIME        .BS 2   ALSO HDR.TRACK & HDR.VOLUME
CURRENT.TRACK.OLD .BS 1
TARGET.TRACK      .BS 1
*--------------------------------
*       DELAY TIMES FOR ACCELERATION & DECELERATION
*       OF TRACK STEPPING MOTOR
*--------------------------------
ONTBL  .HS 01302824201E1D1C1C
OFFTBL .HS 702C26221F1E1D1C1C
*--------------------------------
*       DELAY ABOUT 100*A MICROSECONDS
*       RUN DOWN MOTOR.TIME WHILE DELAYING
*--------------------------------
DELAY.100
.1      LDX #17
.2      DEX
        BNE .2
        INC MOTOR.TIME
        BNE .3
        INC MOTOR.TIME+1
.3      SEC
        SBC #1
        BNE .1
        RTS
*--------------------------------
READ.ADDRESS
        LDY #$FC     TRY 772 TIMES TO FIND $D5
        STY SEEK.D5.CNT        (FROM $FCFC TO $10000)
.1      INY
        BNE .2       ...KEEP TRYING
        INC SEEK.D5.CNT
        BEQ .11      ...THAT IS ENUF!
.2      LDA DRV.Q6L,X     GET NEXT BYTE
        BPL .2
.3      CMP #$D5     IS IT $D5?
        BNE .1       ...NO, TRY AGAIN
        NOP          ...YES, DELAY
.4      LDA DRV.Q6L,X     GET NEXT BYTE
```

```
                BPL .4
                CMP #$AA     NOW NEED $AA AND $96
                BNE .3       ...NO, BACK TO $D5 SEARCH
                LDY #3       (READ 3 BYTES LATER)
       .5       LDA DRV.Q6L,X     GET NEXT BYTE
                BPL .5
                CMP #$96     BETTER BE...
                BNE .3       ...IT IS NOT
                SEI          ...NO INTERRUPTS NOW
                LDA #0       START CHECK SUM
       .6       STA CHECK.SUM
       .7       LDA DRV.Q6L,X     GET NEXT BYTE
                BPL .7       1X1X1X1X
                ROL          X1X1X1X1
                STA X1X1X1X1
       .8       LDA DRV.Q6L,X     GET NEXT BYTE
                BPL .8       1Y1Y1Y1Y
                AND X1X1X1X1   XYXYXYXY
                STA HDR.CHKSUM,Y
                EOR CHECK.SUM
                DEY
                BPL .6
                TAY          CHECK CHECKSUM
                BNE .11      NON-ZERO, ERROR
       .9       LDA DRV.Q6L,X     GET NEXT BYTE
                BPL .9
                CMP #$DE     TRAILER EXPECTED $DE.AA.EB
                BNE .11      NO, ERROR
                NOP
       .10      LDA DRV.Q6L,X
                BPL .10
                CMP #$AA
                BNE .11      NO, ERROR
                CLC
                RTS
       .11      SEC
                RTS
       *--------------------------------
       READ.SECTOR
                TXA          SLOT*16 ($60 FOR SLOT 6)
                ORA #$8C     BUILD Q6L ADDRESS FOR SLOT
                STA .9+1     STORE INTO READ-DISK OPS
                STA .12+1
                STA .13+1
                STA .15+1
                STA .18+1
                LDA RWB.BUFFER    PLUG CALLER'S BUFFER
                LDY RWB.BUFFER+1  ADDRESS INTO STORE'S
                STA .17+1    PNTR FOR LAST THIRD
                STY .17+2
                SEC          PNTR FOR MIDDLE THIRD
                SBC #84
                BCS .1
                DEY
```

```
5800  .1     STA .14+1
5810         STY .14+2
5820         SEC          PNTR FOR BOTTOM THIRD
5830         SBC #87
5840         BCS .2
5850         DEY
5860  .2     STA .11+1
5870         STY .11+2
5880  *---FIND $D5.AA.AD HEADER--------
5890         LDY #32      MUST FIND $D5 WITHIN 32 BYTES
5900  .3     DEY
5910         BEQ .10      ERROR RETURN
5920  .4     LDA DRV.Q6L,X
5930         BPL .4
5940  .5     EOR #$D5
5950         BNE .3
5960         NOP
5970  .6     LDA DRV.Q6L,X
5980         BPL .6
5990         CMP #$AA
6000         BNE .5
6010         NOP
6020  .7     LDA DRV.Q6L,X
6030         BPL .7
6040         CMP #$AD
6050         BNE .5
6060  *---READ 86 BYTES INTO TBUF...TBUF+85----------
6070  *---THESE ARE THE PACKED LOWER TWO BITS--------
6080  *---FROM EACH BYTE OF THE CALLER'S BUFFER.-----
6090         LDY #170
6100         LDA #0       INIT RUNNING EOR-SUM
6110  .8     STA RUNNING.SUM
6120  .9     LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6130         BPL .9
6140         LDA BYTE.TABLE,X      DECODE DATA
6150         STA TBUF-170,Y
6160         EOR RUNNING.SUM
6170         INY
6180         BNE .8
6190  *---READ NEXT 86 BYTES------------------------
6200  *---STORE 1ST 85 IN BUFFER...BUFFER+84---------
6210  *---SAVE THE 86TH BYTE ON THE STACK------------
6220         LDY #170
6230         BNE .12      ...ALWAYS
6240  *--
6250  .10    SEC          I/O ERROR EXIT
6260         RTS
6270  *--
6280  .11    STA BUFF.BASE-171,Y
6290  .12    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6300         BPL .12
6310         EOR BYTE.TABLE,X      DECODE DATA
6320         LDX TBUF-170,Y        MERGE LOWER 2 BITS
6330         EOR BIT.PAIR.TABLE,X
```

```
6340        INY
6350        BNE .11
6360        PHA          SAVE LAST BYTE (LATER BUFFER+85)
6370 *---READ NEXT 86 BYTES-----------
6380 *---STORE AT BUFFER+86...BUFFER+171------------
6390        AND #$FC              MASK FOR RUNNING EOR.SUM
6400        LDY #170
6410 .13    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6420        BPL .13
6430        EOR BYTE.TABLE,X      DECODE DATA
6440        LDX TBUF-170,Y        MERGE LOWER 2 BITS
6450        EOR BIT.PAIR.TABLE+1,X
6460 .14    STA BUFF.BASE-84,Y
6470        INY
6480        BNE .13
6490 *---READ NEXT 84 BYTES------------------------
6500 *---INTO BUFFER+172...BUFFER+255---------------
6510 .15    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6520        BPL .15
6530        AND #$FC
6540        LDY #172
6550 .16    EOR BYTE.TABLE,X      DECODE DATA
6560        LDX TBUF-172,Y        MERGE LOWER 2 BITS
6570        EOR BIT.PAIR.TABLE+2,X
6580 .17    STA BUFF.BASE,Y
6590 .18    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6600        BPL .18
6610        INY
6620        BNE .16
6630        AND #$FC
6640 *---END OF DATA------------------
6650        EOR BYTE.TABLE,X      DECODE DATA
6660        BNE .20           ...BAD CHECKSUM
6670        LDX SLOT.X16      CHECK FOR TRAILER $DE
6680 .19    LDA DRV.Q6L,X
6690        BPL .19
6700        CMP #$DE
6710        CLC
6720        BEQ .21           ...GOOD READ!
6730 .20    SEC               ...SIGNAL BAD READ
6740 .21    PLA               STORE BYTE AT BUFFER+85
6750        LDY #85
6760        STA (RWB.BUFFER),Y
6770        RTS
6780 *--------------------------------
6790 UPDATE.TRACK.TABLE
6800        JSR GET.SSSD.IN.X
6810        STA OLD.TRACK.TABLE,X
6820        RTS
6830 *--------------------------------
6840 CHECK.IF.MOTOR.RUNNING
6850        LDX SLOT.X16
6860 CHECK.IF.MOTOR.RUNNING.X
6870        LDY #0
```

```
6880 .1      LDA DRV.Q6L,X     READ CURRENT INPUT REGISTER
6890         JSR .2            ...12 CYCLES...
6900         PHA               ...7 MORE CYCLES...
6910         PLA
6920         CMP DRV.Q6L,X     BY NOW INPUT REGISTER
6930         BNE .2            SHOULD HAVE CHANGED
6940         LDA #$28     ERROR CODE: NO DEVICE CONNECTED
6950         DEY          BUT TRY 255 MORE TIMES
6960         BNE .1       ...RETURN .NE. IF MOVING...
6970 .2      RTS          ...RETURN .EQ. IF NOT MOVING...
6980 *--------------------------------
6990 GET.SSSD.IN.X
7000         PHA          SAVE A-REG
7010         LDA RWB.SLOT     DSSSXXXX
7020         LSR
7030         LSR
7040         LSR
7050         LSR          0000DSSS
7060         CMP #8       SET CARRY IF DRIVE 2
7070         AND #7       00000SSS
7080         ROL          0000SSSD
7090         TAX          INTO X-REG
7100         PLA          RESTORE A-REG
7110         RTS
7120 *--------------------------------
7130 WRITE.SECTOR
7140         SEC          IN CASE WRITE-PROTECTED
7150         LDA DRV.Q6H,X
7160         LDA DRV.Q7L,X
7170         BPL .1       ...NOT WRITE PROTECTED
7180         JMP WS.RET   ...PROTECTED, ERROR
7190 *--------------------------------
7200 .1      LDA TBUF
7210         STA TBUF.0
7220 *---WRITE 5 SYNC BYTES-----------
7230         LDA #$FF
7240         STA DRV.Q7H,X
7250         ORA DRV.Q6L,X
7260         LDY #4
7270         NOP          $FF AT 40-CYCLE INTERVALS LEAVES
7280         PHA          TWO ZERO-BITS AFTER EACH $FF
7290         PLA
7300 .2      PHA
7310         PLA
7320         JSR WRITE2
7330         DEY
7340         BNE .2
7350 *---WRITE $D5 AA AD HEADER-------
7360         LDA #$D5
7370         JSR WRITE1
7380         LDA #$AA
7390         JSR WRITE1
7400         LDA #$AD
7410         JSR WRITE1
```

```
7420  *---WRITE 86 BYTES FROM TBUF------------------
7430  *---BACKWARDS:   TBUF+85...TBUF+1, TBUF.0------
7440         TYA          =0
7450         LDY #86
7460         BNE .4
7470  .3     LDA TBUF,Y
7480  .4     EOR TBUF-1,Y
7490         TAX
7500         LDA BIT.PAIR.TABLE+3,X
7510         LDX SLOT.X16
7520         STA DRV.Q6H,X
7530         LDA DRV.Q6L,X
7540         DEY
7550         BNE .3
7560         LDA TBUF.0
7570  *---WRITE PORTION OF BUFFER------
7580  *---UP TO A PAGE BOUNDARY--------
7590         LDY #*-*     FILLED IN WITH LO-BYTE OF BUFFER ADDRESS
7600  WS...5 EOR BUFF.BASE,Y   HI-BYTE FILLED IN
7610         AND #$FC
7620         TAX
7630         LDA BIT.PAIR.TABLE+3,X
7640  WS...6 LDX #MODIFIER
7650         STA DRV.Q6H,X
7660         LDA DRV.Q6L,X
7670  WS...7 LDA BUFF.BASE,Y   HI-BYTE FILLED IN
7680         INY
7690         BNE WS...5
7700  *---BRANCH ACCORDING TO BUFFER BOUNDARY CONDITIONS-----
7710         LDA BYTE.AT.BUF00
7720         BEQ WS..17   ...BUFFER ALL IN ONE PAGE
7730         LDA INDEX.OF.LAST.BYTE
7740         BEQ WS..16   ...ONLY ONE BYTE IN NEXT PAGE
7750  *---MORE THAN ONE BYTE IN NEXT PAGE-------------------
7760         LSR          ...DELAY TWO CYCLES
7770         LDA BYTE.AT.BUF00  PRE.NYBBLE ALREADY ENCODED
7780         STA DRV.Q6H,X      THIS BYTE
7790         LDA DRV.Q6L,X
7800         LDA BYTE.AT.BUF01
7810         NOP
7820         INY
7830         BCS WS..12
7840  WS...8 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7850         AND #$FC
7860         TAX
7870         LDA BIT.PAIR.TABLE+3,X
7880  WS...9 LDX #MODIFIER
7890         STA DRV.Q6H,X
7900         LDA DRV.Q6L,X
7910  WS..10 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
7920         INY
7930  WS..11 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7940  WS..12 CPY INDEX.OF.LAST.BYTE
7950         AND #$FC
```

```
7960        TAX
7970        LDA BIT.PAIR.TABLE+3,X
7980  WS..13 LDX #MODIFIER
7990        STA DRV.Q6H,X
8000        LDA DRV.Q6L,X
8010  WS..14 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
8020        INY
8030        BCC WS...8
8040        BCS .15      ...3 CYCLE NOP
8050  .15    BCS WS..17   ...ALWAYS
8060  *---WRITE BYTE AT BUFFER.00--------------------------
8070  WS..16 .DA #$AD,BYTE.AT.BUF00    4 CYCLES: LDA BYTE.AT.BUF00
8080        STA DRV.Q6H,X
8090        LDA DRV.Q6L,X
8100        PHA
8110        PLA
8120        PHA
8130        PLA
8140  WS..17 LDX LAST.BYTE
8150        LDA BIT.PAIR.TABLE+3,X
8160  WS..18 LDX #MODIFIER
8170        STA DRV.Q6H,X
8180        LDA DRV.Q6L,X
8190        LDY #0
8200        PHA
8210        PLA
8220  *---WRITE DATA TRAILER:  $DE AA EB FF----------
8230        NOP
8240        NOP
8250  .19    LDA DATA.TRAILER,Y
8260        JSR WRITE3
8270        INY
8280        CPY #4
8290        BNE .19
8300        CLC          SIGNAL NO ERROR
8310  WS.RET LDA DRV.Q7L,X     DRIVE TO SAFE MODE
8320        LDA DRV.Q6L,X
8330        RTS
8340  *--------------------------------
8350  WRITE1 CLC
8360  WRITE2 PHA
8370        PLA
8380  WRITE3 STA DRV.Q6H,X
8390        ORA DRV.Q6L,X
8400        RTS
8410  *--------------------------------
8420  PRE.NYBBLE
8430        LDA RWB.BUFFER    PLUG IN ADDRESS TO LOOP BELOW
8440        LDY RWB.BUFFER+1
8450        CLC
8460        ADC #2
8470        BCC .1
8480        INY
8490  .1     STA PN...6+1
```

```
8500        STY PN...6+2
8510        SEC
8520        SBC #$56
8530        BCS .2
8540        DEY
8550 .2     STA PN...5+1
8560        STY PN...5+2
8570        SEC
8580        SBC #$56
8590        BCS .3
8600        DEY
8610 .3     STA PN...4+1
8620        STY PN...4+2
8630 *---PACK THE LOWER TWO BITS INTO TBUF-------------
8640        LDY #170
8650 PN...4 LDA BUFF.BASE-170,Y   ADDRESS FILLED IN
8660        AND #3
8670        TAX
8680        LDA BIT.PAIR.RIGHT,X
8690        PHA
8700 PN...5 LDA BUFF.BASE-84,Y
8710        AND #3
8720        TAX
8730        PLA
8740        ORA BIT.PAIR.MIDDLE,X
8750        PHA
8760 PN...6 LDA BUFF.BASE+2,Y
8770        AND #3
8780        TAX
8790        PLA
8800        ORA BIT.PAIR.LEFT,X
8810        PHA
8820        TYA
8830        EOR #$FF
8840        TAX
8850        PLA
8860        STA TBUF,X
8870        INY
8880        BNE PN...4
8890 *---DETERMINE BUFFER BOUNDARY CONDITIONS----------
8900 *---AND SETUP WRITE.SECTOR ACCORDINGLY------------
8910        LDY RWB.BUFFER
8920        DEY
8930        STY INDEX.OF.LAST.BYTE
8940        LDA RWB.BUFFER
8950        STA WS...5-1
8960        BEQ .7
8970        EOR #$FF
8980        TAY
8990        LDA (RWB.BUFFER),Y
9000        INY
9010        EOR (RWB.BUFFER),Y
9020        AND #$FC
9030        TAX
```

```
9040        LDA BIT.PAIR.TABLE+3,X
9050 .7     STA BYTE.AT.BUF00    =0 IF BUFFER NOT SPLIT
9060        BEQ .9
9070        LDA INDEX.OF.LAST.BYTE
9080        LSR
9090        LDA (RWB.BUFFER),Y
9100        BCC .8
9110        INY
9120        EOR (RWB.BUFFER),Y
9130 .8     STA BYTE.AT.BUF01
9140 .9     LDY #$FF
9150        LDA (RWB.BUFFER),Y
9160        AND #$FC
9170        STA LAST.BYTE
9180 *---INSTALL BUFFER ADDRESSES IN WRITE.SECTOR------
9190        LDY RWB.BUFFER+1
9200        STY WS...5+2
9210        STY WS...7+2
9220        INY
9230        STY WS...8+2
9240        STY WS..10+2
9250        STY WS..11+2
9260        STY WS..14+2
9270 *---INSTALL SLOT*16 IN WRITE.SECTOR---------------
9280        LDX SLOT.X16
9290        STX WS...6+1
9300        STX WS...9+1
9310        STX WS..13+1
9320        STX WS..18+1
9330        RTS
9340 *--------------------------------
9350 WAIT.FOR.OLD.MOTOR.TO.STOP
9360        EOR OLD.SLOT      SAME SLOT AS BEFORE?
9370        ASL               (IGNORE DRIVE)
9380        BEQ .2            ...YES
9390        LDA #1       LONG MOTOR.TIME
9400        STA MOTOR.TIME+1  (COUNTS BACKWARDS)
9410 .1     LDA OLD.SLOT
9420        AND #$70
9430        TAX
9440        BEQ .2       ...NO PREVIOUS MOTOR RUNNING
9450        JSR CHECK.IF.MOTOR.RUNNING.X
9460        BEQ .2       ...NOT RUNNING YET
9470        LDA #1       DELAY ANOTHER 100 USECS
9480        JSR DELAY.100
9490        LDA MOTOR.TIME+1
9500        BNE .1       KEEP WAITING
9510 .2     RTS
9520 *--------------------------------
9530        .BS $FF9B-*       <<<<EMPTY SPACE>>>>
9540 *--------------------------------
9550 IRQ
9560        PHA          SAVE A-REG
9570        LDA $45     SAVE LOC $45
```

```
 9580        STA SAVE.LOC45
 9590        PLA          SAVE A-REG AT LOC $45
 9600        STA $45
 9610        PLA          GET STATUS BEFORE IRQ
 9620        PHA
 9630        AND #$10     SEE IF "BRK"
 9640        BNE .2       ...YES, LET MONITOR DO IT
 9650        LDA $D000    SAVE $D000 BANK ID
 9660        EOR #$D8
 9670        BEQ .1
 9680        LDA #$FF
 9690 .1     STA INTBANKID
 9700        STA SAVE.D000
 9710        LDA #$BF     PUSH FAKE "RTI" VECTOR WITH
 9720        PHA               IRQ DISABLED
 9730        LDA #$50          AND SET TO RETURN TO $BF50
 9740        PHA
 9750        LDA #4
 9760        PHA
 9770 .2     LDA #$FA     PUSH "RTS" VECTOR FOR MONITOR
 9780        PHA
 9790        LDA #$41
 9800        PHA
 9810 CALL.MONITOR
 9820        STA $C082    SWITCH TO MOTHERBOARD
 9830 *--------------------------------
 9840 RESET
 9850        LDA RESET.VECTOR+1
 9860        PHA          PUSH "RTS" VECTOR FOR MONITOR
 9870        LDA RESET.VECTOR
 9880        PHA
 9890        JMP CALL.MONITOR
 9900 *--------------------------------
 9910 RESET.VECTOR
 9920        .DA $FA61    MON.RESET-1
 9930 *--------------------------------
 9940 INT.SPLICE
 9950        STA INTAREG
 9960        LDA SAVE.LOC45
 9970        STA $45
 9980        LDA $C08B    SWITCH TO MAIN $D000 BANK
 9990        LDA SAVE.D000
10000        JMP IRQXIT.3
10010 *--------------------------------
10020        .BS $FFFA-*       <<<<<EMPTY SPACE>>>>>
10030 *--------------------------------
10040 V.NMI      .DA $03FB
10050 V.RESET    .DA RESET
10060 V.IRQ      .DA IRQ
10070 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8311:DOS3.3:S.KILL.EXEC.txt
==============================================================================

 1000  *SAVE S.KILL.EXEC
 1010  *--------------------------------
 1020  RESET        .EQ $3F2
 1030  SET.PWR.BYTE .EQ $FB6F
 1040  DOS.ENTRY    .EQ $3D0
 1050  EXEC.STATUS  .EQ $AAB3
 1060  *--------------------------------
 1070         .OR $300
 1080         .TF B.KILL.EXEC
 1090  *--------------------------------
 1100  INIT   LDA #KILL.EXEC
 1110         STA RESET
 1120         LDA /KILL.EXEC
 1130         STA RESET+1
 1140         JMP SET.PWR.BYTE
 1150  *--------------------------------
 1160  KILL.EXEC
 1170         LDA #0
 1180         STA EXEC.STATUS
 1190         JMP DOS.ENTRY
```

==============================================================================
DOCUMENT :AAL-8312:Articles:Dataphile.Dgst.txt
==============================================================================

Demise of Bailey's DataPhile Digest

Unfortunately, we no sooner sent out last month's AAL than we received a letter from the Baileys saying that they have ceased to publish the DataPhile Digest.

```
==============================================================================
DOCUMENT :AAL-8312:Articles:Front.Page.txt
==============================================================================
```

$1.50

Volume 4 -- Issue 3                    December, 1983

In This Issue...



Demise of Bailey's DataPhile Digest

Unfortunately, we no sooner sent out last month's AAL than we received a letter from the Baileys saying that they have ceased to publish the DataPhile Digest.

Quarterly Disk 13

QD 13 is now ready, and it includes both installments of ProDOS commented source code as listed last month and this.  The code is in the format used by the S-C Macro Assembler.  (Since the disk also includes the CONVERT S-C TO TEXT program in this issue, all of you can use it!)  Quarterly Disks are $15 each, or $45 for a year's subscription.

Subscription Rates

Remember, subscriptions to Apple Assembly Line will be increasing to $18/year effective January 1.  Since some of you may not receive this issue (or your renewal notice) until after that date, we'll extend the deadline to January 15 for renewals.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
=============================================================================
DOCUMENT :AAL-8312:Articles:LabelGOTO.Gosub.txt
=============================================================================
```

Generalized GOTO and GOSUB.................Bob Sander-Cederlof

Tim Mowchanuk, a lecturer at Brisbane College in Australia, sent the following suggestion:

"How can I implement a named GOTO or GOSUB routine?  There are numerous routines that implement computed GOTO/GOSUB, but I consider that a futile exercise.  Computed GOTO/GOSUB mess up renumbering utilities, and violate modern trends toward structured programming.

"What I really want is something that will handle BASIC like

```
       100 & GOSUB NAME$
```

where NAME$ holds the name of a subroutine.  I envision subroutine names being defined by a special REM statement of the form

```
       200 REM "SUBROUTINE NAME"
```

The &GOSUB or &GOTO processor can search through the program for a line beginning with a REM token.  If the first non-blank after the REM token is a quotation mark, the processor can compare the characters to the string value.  If there is an exact match, the line containing the REM is the target for the &GOTO or &GOSUB."

The problem sounded just the right size for an interesting AAL article, so I started trying to write some code.

I published an &GOSUB routine back in April 1981 of the type that Tim thinks futile.  The following program combines the two "futile" computed &GOSUB and &GOTO routines with two new ones that allow the computed value to be a string expression.  If the expression after &GOTO or &GOSUB is numeric, the processor will search for a matching line number.  If the result is a string, the processor will search for a REM label as Tim described above.

Only REM's at the beginning of a numbered line will be considered as labels.  The label must be included in quotation marks.  Spaces are OK between the word REM and the first quotation mark.  Anything after the second quotation mark will be ignored.

You can now write a menu program that uses the actual command word as the name of a subroutine, and cease worrying about line numbers.  The accompanying Applesoft program is an example of just such a technique.

```
==================================================================
DOCUMENT :AAL-8312:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler Version 1.0..................................$80.00
S-C Macro Assembler Version 1.1 Update...........................$12.50
Full Screen Editor for S-C Macro Assembler..........(reg. $49.00)  $40.00**
    Includes complete source code.
S-C Cross Reference Utility......................................$20.00
S-C Cross Reference Utility with Complete Source Code............$50.00
DISASM Dis-Assembler (RAK-Ware)..................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $40.00**

S-C Word Processor (the one we use!).............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor.....(reg. $60.00)  $40.00**

AAL Quarterly Disks.........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983

Double Precision Floating Point for Applesoft....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research).....(reg. $79.00)  $50.00**
Fontrix (Data Transforms)........................................  $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00

Blank Diskettes..................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
ZIF Game Socket Extender.........................................$20.00
Shift-Key Modifier...............................................$15.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
```

```
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00
  Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***

(** Special price to subscribers only through December 31, 1983.)
 ** Last Chance!!  Order now and save.
```

=============================================================================
DOCUMENT :AAL-8312:Articles:ProDOS.Listing.txt
=============================================================================

Commented Listing of ProDOS $F90C-F995, $FD00-FE9A, $FEBE-FFFF
...........Bob Sander-Cederlof

Last month I printed the commented listing of the disk reading subroutines. This month's selection covers disk writing, track positioning, and interrupt handling. Together the two articles cover all the code between $F800 and $FFFF.

Several callers have wondered if this is all there is to ProDOS. No! It is only a small piece. In my opinion, this is the place to start in understanding ProDOS's features: A faster way of getting information to and from standard floppies. But remember that ProDOS also supports the ProFILE hard disk, and a RAM disk in the extended Apple //e memory.

Further, ProDOS has a file structure exactly like Apple /// SOS, with a hierarchical directory and file sizes up to 16 megabytes.

Further, ProDOS includes support for a clock/calendar card, 80-columns with Smarterm or //e, and interrupts.

ProDOS uses or reserves all but 255 bytes of the 16384 bytes in the language card area (both $D000-DFFF banks and all #E000-FFFF). The 255 bytes not reserved are from $D001 through $D0FF in one of the $D000 banks. The byte at $D000 is reserved, because ProDOS uses it to distinguish which $D000 bank is switched on when an interrupt occurs. The space at $BF00-BFFF is used by ProDOS for system linkages and variables (called the System Global Page).

In addition, if you are using Applesoft, ProDOS uses memory from $9600-BEFF. This space does not include any file buffers. When you OPEN files, buffers are allocated as needed. CLOSEing automatically de-allocates buffers. Each buffer is 1024 bytes long. As you can see, with ProDOS in place your Applesoft program has less room than ever.

Track Seeking: $F90C-F995

The SEEK.TRACK subroutine begins at $F90C. The very first instruction multiplies the track number by two, converting ProDOS logical track number to a physical track number. If you want to access a "half-track" position, you could either store a NOP opcode at $F90C, or enter the subroutine at $F90D.

A table is maintained of the current track position for each of up to 12 drives. I call it the OLD.TRACK.TABLE. The subroutine GET.SSSD.IN.X forms an index into OLD.TRACK.TABLE from slot# * 2 + drive#. There are no entries in the table for drives in slots 0 or 1, which is fine with me.  ProDOS uses these slots as pseudo slots for the RAM-based pseudo-disk and for ProFILE, if I remember correctly.

The code in SEEK.TRACK.ABSOLUTE is similar but not identical to code in DOS 3.3.  The differences do not seem to be significant.

Disk Writing:  $FD00-FE9A

The overall process of writing a sector is handled by code in RWTS, which was listed last month.  After the desired track is found, RWTS calls PRE.NYBBLE to build a block of 86 bytes containing the low-order two bits from each byte in the caller's buffer.  PRE.NYBBLE also stores a number of buffer addresses and slot*16 values inside the WRITE.SECTOR subroutine.  Next RWTS calls READ.ADDRESS to find the sector, and then WRITE.SECTOR to put the data out.

WRITE.SECTOR is the real workhorse.  And it is very critically timed. Once the write head in your drive is enabled, every machine cycle is closely counted until the last byte is written.  First, five sync bytes are written (ten bits each, 1111111100).  These are written by putting $FF in the write register at 40 cycle intervals.  Following the sync bytes W.S writes a data header of D5 AA AD.

Second, the 86-byte block which PRE.NYBBLE built is written, followed by the coded form of the rest of your buffer.  WRITE.SECTOR picks up bytes directly from your buffer, keeps a running checksum, encodes the high-order six bits into an 8-bit value, and writes it on the disk...one byte every 32 cycles, exactly.  Since your buffer can be any arbitrary place in memory, and since the 6502 adds cycles for indexed instructions that cross page boundaries, WRITE.SECTOR splits the buffer in parts before and after a page boundary.  All the overhead for the split is handled in PRE.NYBBLE, before the timed operations begin.

Finally, the checksum and a data trailer of DE AA EB FF are written.

Empty Space:  $FEBE-FF9A

This space had no code in it.  Nearly a whole page here.

Interrupt & RESET Handling:  $FF9B-FFFF

If the RAM card is switched on when an interrupt or RESET occurs, the vectors at $FFFA-FFFF will be those ProDOS installed rather than the ones permanently coded in ROM.  It turns out the non-maskable interrupt (NMI) is still vectored down into page 3.  But the more interesting IRQ interrupt is now vectored to code at $FF9B inside ProDOS.

The ProDOS IRQ handler performs two functions beyond those built-in to the monitor ROM.  First, the contents of location $45 are saved so that the monitor can safely clobber it.  Second, a flag is set indicating which $D000 bank is currently switched on, so that it can be restored after the interrupt handler is finished. (The second step is omitted if the interrupt was caused by a BRK opcode.)

If the IRQ was not due to a BRK opcode, a fake "RTI" vector is pushed on the stack. This consists of a return address of $BF50 and a status of $04. The status keeps IRQ interrupts disabled, and $BF50 is a short routine which turns the ProDOS memory back on and jumps up to INT.SPLICE at $FFD8:

```
       BF50-  8D 8B C0  STA $C08B
       BF53-  4C D8 FF  JMP $FFD8
```

Of course, before coming back via the RTI, ProDOS tries to USE the interrupt. If you have set up one or more interrupt vectors with the ProDOS system call, they will be called.

INT.SPLICE restores the contents of $45 and switches the main $D000 bank on. Then it jumps back to $BFD3 with the information about which $D000 bank really should be on. $BFD3 turns on the other bank if necessary, and returns to the point at which the interrupt occured.

The instruction at $FFC8 is interesting. STA $C082 turns on the monitor ROM, so the next instruction to be executed is at $FFCB in ROM. This is an RTS opcode, so the address on the stack at that point is used. There are two possible values: $FA41 if an IRQ interrupt is being processed, or $FA61 if a RESET is being processed. This means the RTS will effectively branch to $FA42 or $FA62.

Uh Oh! At this point you had better hope that you are not running with the original Apple monitor ROM. The Apple II Plus ROM (called Autostart Monitor) and the Apple //e ROM are fine. $FA42 is the second instruction of the IRQ code, and $FA62 is the standard RESET handler. But the original ROM, like I have in my serial 219 machine, has entirely different code there.

I have an $FF at $FA42, followed by code for the monitor S (single step) command. And $FA62 is right in the middle of the S command. There is no telling what might happen, short of actually trying it out. No thanks. Just remember that RESET, BRK, and IRQ interrupts will not work correctly if they happen when the RAM area is switched on and you have the old original monitor in ROM.

There is another small empty space from $FFE9 through $FFF9, 17 bytes.

Perhaps I should point out that the listings this month and last are from the latest release of ProDOS, which may not be the final released version. However, I would expect any differences in the regions I have covered so far to be slight

==============================================================================
DOCUMENT :AAL-8312:Articles:Shafer.Asm.Text.txt
==============================================================================

More Assembly Listing into Text Files...........Tracy L. Shafer
MacDill AFB, FL

In the October '83 issue of AAL, Robert F. O'Brien presented a way to create a text file containing the assembly listing of a large program. (See also "Assembly Listing Into a Text File", by Bill Morgan, July '83 AAL.) Actually, he created several text files; one for each .IN directive in the root file. You can't put the whole listing into one text file by using one .TF directive because of the way the .IN directive affects the DOS I/O hooks.

Robert's method for obtaining assembly listing text files is good, but I found a different way to create the text files of assembly listings that doesn't involve creating separate SYMBOLS sections, deleting duplicate labels, and putting up with "EXTRA DEFINITIONS ERROR" messages. It's a fairly simple approach and hinges on the fact that the problem presented by the .IN directive affects the source file containing the .IN, but not the source file to which the .IN refers. Instead of putting one .TF directive in the root file, put a .TF in each source file pointed to by a .IN directive.

For example:

```
        ROOT FILE

        1000    .DU
        1010    .IN PART 1
        1020    .IN PART 2
        1030    .ED

        PART 1

        1000    .TF LISTING 1
        1010    (source for part 1)

        PART 2

        1000    .TF LISTING 2
        1010    (source for part 2)
```

From here on, follow Bill Morgan's original instructions. What follows is a summary of those instructions.

After deleting all other .TF directives, or turning them into comments by inserting "*" at the beginning of the line, typing ASM will create two binary files named LISTING 1 and LISTING 2. Each of these contains the assembly listing of PART 1 and PART 2 respectively, in text form. These binary files will not have starting address and length in the first four bytes. DO NOT attempt to BLOAD these files.

You could really clobber DOS. To obtain true text files, make the following patch to the S-C Assembler before you assemble the program:

```
        $1000 versions:  $29DF:0  (original value is 04)
        $D000 versions:  $C083 C083 EAF9:0 N C083
```

After the patch is made, assemble the program and restore the original value to $29DF ($EAF9).

For really large programs, it could get very tedious adding a .TF directive to each sub-file to obtain a text file listing and then deleting those .TF directives to prevent messing up the object file the next time the program is assembled. Fortunately, the S-C Macro Assembler's conditional assembly feature makes our work a lot easier. By placing an equated flag in the root file and surrounding each .TF with .DO and .FIN, we only have to change one line to set up our program for text file output or object file creation. For example:

```
        ROOT FILE

        1000 LSTOUT .EQ 0       TO ASSEMBLE OBJECT
        1010 *          1       TO OUTPUT TEXT FILES
        1020        .DO LSTOUT
        1040        .DU
        1050        .ELSE
        1060        .TF OBJECT FILE
        1070        .FIN
        1080        .IN PART 1
        1090        .IN PART 2
        1100        .DO LSTOUT
        1110        .ED
        1120        .FIN

        PART 1

        1000        .DO LSTOUT
        1010        .TF LISTING 1
        1020        .FIN
        1030        (source for part 1)

        PART 2

        1000        .DO LSTOUT
        1010        .TF LISTING 2
        1020        .FIN
        1030        (source for part 2)
```

Don't forget to patch $29DF ($EAF9 for the language card version) with 0 to output true text files and back to 04 create object files.  The last thing to remember is to use .LIST ON during the assembly. You won't write any text files if the assembler isn't producing a listing.

================================================================
DOCUMENT :AAL-8312:Articles:Short.Stuff.txt
================================================================

Note on Aztec C....................................Bill Morgan

I just talked to the people at Manx Software about the ProDOS version of their C compiler, and this time they assured me that owners of the current Apple DOS version will be able to purchase the ProDOS version at a reduced upgrade price. That is enough to tip the balance in favor of buying the compiler right now, so I have ordered some. List price is $199: we'll have them for $180 + shipping.

Where To?, Revisited................................Bill Morgan

Many thanks to all of you who responded to my questions about 68000, C, and the future of Apple Assembly Line.

Your answers ran about eight to one in favor of including 68000 information in AAL. Several writers suggested starting with a few pages, and possibly splitting off a separate newletter someday. That sounds like a good plan, so we'll start a regular section next issue. Those of you who already know 68000 can now start teaching the rest of us. Bob Urschel has already sent in a brief article and program! He has a QWERTY Q68 board like that we reviewed last month, and speaks very highly of it.

Interest in Mackintosh (MacIntosh? Apple 32?) is growing rapidly: the announcement is expected at the Apple shareholder meeting in mid-January. Some reports claim that some developers have had Mac for up to 18 months now. We haven't been among those so privileged, but I hope to be the first on my block with one. (Unless the thing turns out to have some fatal flaw, like no expansion slots. That was one rumor!)

Several of you also expressed an interest in C, but not even a majority. More like 30%. It looks like a number of people are curious, but feel that too much coverage would dilute AAL. Stephen Bach said it best, "... don't spread yourselves too thin and try to do C also." I expect to do occasional reviews and mentions of books and other aids to learning C, and to report on anything specifically related to C on Apple computers, but not much more.

==============================================================================
DOCUMENT :AAL-8312:Articles:STB.128.Testing.txt
==============================================================================

Finding Trouble in a Big RAM Card..........Bob Sander-Cederlof

Last night (Monday, Nov 28th) I took home an Apple to do some spreadsheet work.  I took home the most portable one, but first swapped RAM cards.  I took the STB-128 out of my oldest Apple and put it into the Apple II Plus with the fewest attachments.

When I plugged it in at home and booted the spreadsheet program, all appeared to be well.  But it wasn't.  I loaded in a model, and during the re-calculation the spreadsheet program hit a BRK opcode and died. I pressed RESET and looked at the partially re-calculated sheet:  it was sprinkled with nonsense characters, and the keyboard was locked up.  I played with various combinations for an hour or so, including other programs which use the RAM card.  Everything pointed to there being a bad bit somewhere in the card.

Of course the RAM card test program was back at the office.  I decided to write another one rather than face the two mile round trip.

The 128K space on the STB-128 is divided into 8 banks.  You select a bank by storing a bank number (0-7) at any address in the $C080+slot*16 space which has bit 2 = 1.  For slot 0, that means store in $C080, $C081, $C082, $C083, $C088, $C089, $C08A, or $C08B.  The card has three green LEDs on top which show which bank is currently selected.

Each 16K bank is further divided to fit into the 12K address space between $D000 and $FFFF.  The softswitch controlled by bit 3 in the $C08x address selects which of two 4K banks will be enabled at $D000-DFFF.  The other 8K always sits at $E000-FFFF.  A red LED signals which $D000 bank is selected.

The low-order two bits of the $C08x address control the mode of the RAM card.  Accessing $C080 or $C088 write protects the card, and read enables it.  This means the $D000-FFFF references the RAM card rather than the motherboard ROM.  Accessing $C082 or $C08A write protects the RAM card and disables reading it; in other words, it switches on the motherboard ROM.

$C081 or $C089 also turn on the mother board ROM for reading, but if you access one of these twice in a row it will write enable the RAM card.  In this mode reads reference the motherboard ROM, but writes write into the RAM card.  This mode is used when loading the RAM card so that monitor and Applesoft routines which are in motherboard ROM can be used for the loading process.

Accessing $C083 or $C08B once read enables the STB-128 card and write protects it.  A second access write enables the card.  This is the mode we use for a memory test.

Thinking about how to test such a card, I wrote down the following "flow chart":

```
       For Bank = 0 to 7
           Store Bank in $C083
           Access $C083 again to write enable
               Test $D000-DFFF
           Access $C08B twice
               Test $D000-FFFF
       Next Bank
```

I broke the actual testing of a range of memory into four parts. First I stored zeroes into every location, and checked to be sure I read zeroes back. Then I did the same with $FF. Then, $55. Then, $AA. This is certainly not an exhaustive test, but I hoped it would be sufficient.

The tricky part was informing myself of the locations and values involved of any memory errors found during the test. I could not conveniently use the monitor subroutines to write addresses and values on the screen, because the monitor only existed in the motherboard ROM and it was switched off! So, I wrote a quick and dirty display routine.

The routine for display in the listing below is not quite so "quick and dirty". The program starts by clearing the screen using the monitor HOME subroutine at $FC58. Then it switches to the RAM card and runs the test. The program pokes test failure data directly to the screen. I direct the data for each of the 8 banks to a different line. When a failure occurs, I print the address, the value that should have been there, the actual value found, and the exclusive-or of the two values. The exclusive-or shows me which bit or bits was incorrect.

After running the test it was obvious that the least significant bit in banks 5 and 6 was not working. When it should be zero it was sometimes one, and vice versa.

I did not know which chip on the STB-128 card belonged to which bit slice or which bank, so I guessed. I was lucky, and guessed right the first time. I pulled out the chip I thought might be the bad one, and re-ran the test. This time the test indicated the least significant bit of banks 4-7 was missing. (It happened to be the chip in the lower-left corner when looking at the face of the card.)

I put the chip back in, hoping that it would miraculously heal itself. Then I looked at the back of the board to see if anything looked suspicious there. Sure enough! STB did not trim off the excess length of the socket pins after soldering the board. One of those long pins had bent over and was possibly shorted to another, on the lower left socket. I straightened the pin and re-ran the test. Voila! It passed!

After I finished patting myself on the back I tried to run the spreadsheet again. It still failed! This morning I put the cards back in their usual homes, and everything works fine.

Tuesday Afternoon....Lo and behold, the card is still bad. I found the STB Systems diskette, and ran their RAM test program. It identified the same chip as being bad. But after running the test for several hours, the errors stopped. Obviously the chip's problems are intermittent.

Wednesday Morning....The chip is still giving errors. I called STB and they said to bring the board by. Wednesday afternoon....STB replaced the chip, and all is well.

===========================================================================
DOCUMENT :AAL-8312:Articles:TimeMaster.txt
===========================================================================

Timemaster II from Applied Engineering.....Bob Sander-Cederlof

It may come as a surprise (it did to me), but there are apparently now only three calendar/clocks still on the market for the Apple II, II Plus, //e. The others, and there were a lot of them, seemed to have dropped off the map. And even one of the three (Mountain Computer) does not advertise anywhere I can find.

Another surprise: the most expensive clock has the fewest features, and the least expensive has the most features.

Mountain Computer Apple Clock

$280 in current catalog listing; most recent ad I could find was in Jan 1980 Byte, at $199. Features below are guessed at from ad and conversations with Dan Pote. Works with BASIC only, does not include any DOS Dater or ProDOS support.

Gives month, day of month, hour, minute, second, millisecond

Interrupt available: Second, Millisecond

Thunderware Thunderclock Plus

Gives month, day of month, day of week, hour, minute, second.

```
$150 with BASIC software for DOS or ProDOS
$ 29 extra for Pascal software
$ 29 extra for DOS-DATER/DEMO disk
```

Interrupts available: 64, 256, or 2048 times per second

Applied Engineering Timemaster

```
$129 includes Applesoft support for DOS or ProDOS
     includes Pascal and CP/M support
     includes DOS Dater
```

Gives year, month, day of month, day of week, hour, minute, second

Interrupts available: Millisecond, Second, Minute, Hour. Switchable to either NMI or IRQ interrupt line.

For some reason they have not chosen to explain, the wizards at Apple who created ProDOS decided to "wire in" support for the Thunderclock (and ONLY Thunderclock). A system call reads the time and date from Thunderclock, calculates the year from the given information, and stores year-month-day-hour-minute in a packed format at $BF90...BF93.

ProDOS automatically records time/date of creation and time/date of last modification.

In order to get the year with these dates, ProDOS goes through a calculation to derive year from given day of month, month, and day of week information. The calculation involves remaindering and table lookup...but it only works from 1982 through 1987. I suppose by 1988 they will have generated a new version which works beyond, or else we won't care anymore. Better yet, by 1988 maybe they will have driver-ized the clock support so we can use Dan's card directly.

Dan Pote sent me a Timemaster to play with, in hopes that I would figure out how to make it look like a Thunderclock to ProDOS. I did, so if you buy one now it will be completely compatible with ProDOS. You select by DIP Switch which page of the onboard EPROM will be mapped into the $CN00 space (where N is slot 1-7). One setting selects the ProDOS section, and the others select various versions designed for use with DOS and Applesoft.

You can talk to Dan's card directly, as well as through the EPROM. If you don't like the way his firmware works (unlikely), you can either ignore it or change it.

(By the way.... Call A.P.P.L.E., a club/magazine with a penchant for value and quality, has chosen to offer another one of Applied Engineering's boards in its latest catalog: the Viewmaster 80. Their price is $140, which is 20% below normal retail.)

==============================================================================
DOCUMENT :AAL-8312:Articles:Trans.Src.Files.txt
==============================================================================

Procedure for Converting S-C Source Files to Text Files
Without Owning an S-C Assembler
...........Bob Sander-Cederlof

Strangely enough, there are some of you who still do not own an S-C Assembler. And some of you buy or would like to buy our Quarterly Disks or the Applesoft Docu-Mentor disks.

These disks contain source files which are only usable by the S-C Macro Assembler. However, it is possible (even without an S-C Assembler) to convert them to regular text files so as to be readable by another brand assembler/editor.

The files appear in the catalog as type "I", which is supposed to mean Integer BASIC. Of course the contents has nothing to do with Integer BASIC, but making them "I-files" has several advantages:

* they LOAD/SAVE faster than text files
* standard DOS commands can be used for load/save
* when the S-C Assembler is in the RAM card, DOS can automatically switch between Applesoft and Assembler as it normally would between Applesoft and Integer BASIC.

There are also some dis-advantages:

* some users have trouble believing they are not really Integer BASIC programs, and try to RUN them.
* the files are harder for people without an S-C Assembler to convert to another brand.

Which brings us back to the point of this article.

To make the procedure simple, you need at least a 64K Apple. If you have an Apple //e, you are all set. An older Apple needs a "language card", or "RAM card".

The first step in the conversion process is to load the file into memory and find out where it is. Start by booting with your DOS 3.3 System Master disk, which loads Integer BASIC into the RAM card. Then LOAD the S-C source file which you want to convert. Integer BASIC will be switched on, but don't try to LIST or RUN!

Enter the Monitor by typing "CALL -151". At this point you will get an asterisk prompt. Look at locations $4C, $4D, $CA, and $CB. You can do it like this:

```
*4C.4D CA.CB
004C- 00 96
00CA- 58 73
```

Interpret the above as meaning that the source code begins in memory at $7358 and ends one byte before $9600.

If you use the monitor commands to look at the first 30 or 40 bytes (or more), you will discover how the source lines are stored. Each line begins with a byte count, which if added to the address will give the address of the first byte of the next line. Each line ends with a 00 byte. The byte count includes both of these bytes, and all in between. Here is a sample line:

```
0F E8 03 41 42 43 84 4C 44 41 81 23 24 35 00
```

The second and third bytes are the binary form of the line number. As usual in 6502 domain, the number is stored low-byte first. $3E8 means the line above is line 1000.

The fourth byte and beyond are ASCII codes for the text of the line, with two exceptions. If the bytes are less than $80, they are plain ASCII. If they are in the range from $81 through $BF, they represent a series of blanks. $81 means one blank, $84 means four blanks, and so on. The line above now decodes to:

```
1000 ABC    LDA #$5
```

The other exception is not illustrated above, but here is one:

```
08 F2 03 2A C0 20 2D 00
```

The token $C0 means "repeated character". The next byte after $C0 gives the number of repetitions, and the byte after that tells what character to repeat. Above the C0 20 2D means 32 "-" characters, so the whole line looks like this:

```
1010 *--------------------------------
```

Armed with all that information, you can probably see how to write a simple Applesoft program to convert the memory image of the S-C source file to plain text and then write it on a text file.

In fact, here is just such a program:

<<<<<listing of CONVERT S-C TO TEXT here>>>>>

Here is a blow-by-blow description of how to use the program.

1. Boot your DOS System Master to load INTBASIC into the RAM card.
2. Load the S-C source file.
3. Type CALL-151 to get into the monitor.
4. Type CA.CB to get the starting address of the S-C source program (xx yy).

5. Type 300:xx yy to store the starting address in a place Applesoft will not clobber.
6. Type 3D0G to return to Integer BASIC.
7. Type RUN CONVERT S-C TO TEXT to execute the Applesoft program listed above.
8. Stand back and wait while the program chugs through the bytes. When you see the Applesoft prompt again, it is all done!

If you add a line at 315 to turn on MONCIO, you can see the text as it is produced.

```
================================================================================
DOCUMENT :AAL-8312:DOS3.3:Conv.SC2Text.txt
================================================================================

)d≤CONVERT MEMORY IMAGE OF S-C SOURCEDn≤TO ORDINARY TEXT FILE[»HM-
,(76)»256 ,(77)t“PP-,(768)»256 ,(769)|‹£PPí,≤OPEN A TEXT FILEû6D$-
Á(4)“@∫D$"OPEN TEXTFILENAME":∫D$"DELETE TEXTFILENAME"     J∫D$"OPEN
TEXTFILENAME":∫D$"WRITE TEXTFILENAME"     êL–PP%    ö≠L-
HMƒ∫D$"CLOSE":Ä<      §∞500:≤ DO ONE LINEE      Æ´410]    Ù≤DO ONE
SOURCE LINEh    ˛N-,(L)è  LN-,(L»1)»256 ,(L»2):∫LN" ";:L–L»2Ø
     L–L»1:C-,(L):≠C–0ƒ∫:L–L»1:±∆    ≠C—128ƒ∫Á(C);:´530È       &≠C—
192ƒÅI–1¡C…128:∫" ";:Ç:´530
0≠C–192ƒÅI–1¡,(L»1):∫Á(,(L»2));:ÇI:L–L»2:´530M
:∫:∫D$"CLOSE":∫"***ERROR IN SOURCE AT "L"***":Ä
```

```
==========================================================================
DOCUMENT :AAL-8312:DOS3.3:S.Labelled.GOs.txt
==========================================================================

 1000  *SAVE S.LABELLED GO'S
 1010  *--------------------------------
 1020  *       & GOTO <STR EXP>
 1030  *       & GOSUB<STR EXP>
 1040  *       REM "<LABEL>"
 1050  *
 1060  *       AS SUGGESTED BY TIM MOWCHANUK
 1070  *--------------------------------
 1080  AS.VALTYP  .EQ $11
 1090  AS.TEMPPT  .EQ $52,53
 1100  INDEX.REM  .EQ $5E
 1110  INDEX.GO   .EQ $5F
 1120  PRGBOT     .EQ $67,68
 1130  AS.CURLIN  .EQ $75,76
 1140  PNTR       .EQ $9B,9C
 1150  STRLEN     .EQ $9D
 1160  STRADR     .EQ $9E,9F
 1170  VPNT       .EQ $A0,A1
 1180  TXTPTR     .EQ $B8,B9
 1190  *--------------------------------
 1200  TKN.GOTO   .EQ $AB
 1210  TKN.GOSUB  .EQ $B0
 1220  TKN.REM    .EQ $B2
 1230  *--------------------------------
 1240  AMPERSAND.VECTOR .EQ $3F5 ... 3F7
 1250  *--------------------------------
 1260  AS.CHRGET  .EQ $00B1
 1270  AS.CHRGOT  .EQ $00B7
 1280  AS.MEMCHK  .EQ $D3D6
 1290  AS.NEWSTT  .EQ $D7D2
 1300  AS.GOTO1   .EQ $D941
 1310  AS.GOTO.3  .EQ $D95E
 1320  AS.UNDERR  .EQ $D97C
 1330  AS.FRMEVL  .EQ $DD7B
 1340  AS.SYNERR  .EQ $DEC9
 1350  AS.FRETMP  .EQ $E604
 1360  AS.GETADR  .EQ $E752
 1370  *--------------------------------
 1380         .OR $300
 1390         .TF B.LABELLED GO'S
 1400  *--------------------------------
 1410  SETUP  LDA #LABELLED.GOTO.AND.GOSUB
 1420         STA AMPERSAND.VECTOR+1
 1430         LDA /LABELLED.GOTO.AND.GOSUB
 1440         STA AMPERSAND.VECTOR+2
 1450         RTS
 1460  *--------------------------------
 1470  LABELLED.GOTO.AND.GOSUB
 1480         JSR AS.CHRGOT
```

```
1490        CMP #TKN.GOTO
1500        BEQ .3
1510        CMP #TKN.GOSUB
1520        BEQ .2        ...GOOD SYNTAX SO FAR
1530 .1     JMP AS.SYNERR
1540 *---SETUP GOSUB RETURN DATA------
1550 .2     LDA #3
1560        JSR AS.MEMCHK
1570        LDA TXTPTR+1
1580        PHA
1590        LDA TXTPTR
1600        PHA
1610        LDA AS.CURLIN+1
1620        PHA
1630        LDA AS.CURLIN
1640        PHA
1650        LDA #TKN.GOSUB
1660        PHA
1670        BNE .4        ...ALWAYS
1680 *---SETUP FOR GOTO---------------
1690 .3     PLA           POP RETURN TO "NEWSTT"
1700        PLA
1710 *---FIND LABEL AFTER TOKEN-------
1720 .4     JSR AS.CHRGET
1730        BEQ .1
1740        JSR AS.FRMEVL     EVALUATE EXPRESSION
1750        BIT AS.VALTYP     $00 IF NUMERIC, $FF IF STRING
1760        BMI .5            ...STRING
1770 *---NUMERIC EXPRESSION-----------
1780        JSR AS.GETADR     CONVERT TO INTEGER
1790        JSR AS.GOTO1
1800        JMP AS.NEWSTT
1810 *---FREE ANY TEMP STRINGS--------
1820 .45    LDA AS.TEMPPT+1
1830        LDY #0
1840        JSR AS.FRETMP
1850 .5     LDA AS.TEMPPT
1860        CMP #$56      EMPTY?
1870        BCS .45       ...NO, FREE A STRING
1880 *---COPY STRING LENGTH/ADDRESS---
1890        LDY #2
1900 .55    LDA (VPNT),Y
1910        STA STRLEN,Y
1920        DEY
1930        BPL .55
1940 *---SEARCH PROGRAM FOR LABEL-----
1950        LDA PRGBOT+1      POINT TO BEGINNING
1960        LDX PRGBOT           OF PROGRAM
1970 *---LOOK AT NEXT LINE------------
1980 .6     STA PNTR+1    UPDATE PNTR TO NEXT LINE
1990        STX PNTR
2000        LDY #1        HI-BYTE OF FWD PNTR
2010        LDA (PNTR),Y
2020        BEQ .11       ...END OF PROGRAM
```

```
2030  *---CHECK FOR 'REM "'------------
2040         LDY #4
2050         LDA (PNTR),Y
2060         CMP #TKN.REM
2070         BNE .10      ...NOT REM STATEMENT
2080  .7     INY          NEXT BYTE OF LINE
2090         LDA (PNTR),Y
2100         CMP #' '     IGNORE BLANKS BEFORE "
2110         BEQ .7
2120         CMP #'"'     " YET?
2130         BNE .10      ...NO, NOT A LABEL
2140  *---COMPARE LABEL----------------
2150         STY INDEX.REM
2160         LDA #-1
2170         STA INDEX.GO
2180  .8     INC INDEX.REM
2190         LDY INDEX.REM
2200         LDA (PNTR),Y
2210         BEQ .1       ...EARLY END OF LABEL
2220         INC INDEX.GO
2230         LDY INDEX.GO
2240         CMP #'"'     LEGAL END OF LABEL?
2250         BEQ .9       ...YES
2260         CMP (STRADR),Y
2270         BEQ .8       ...KEEP MATCHING
2280         BNE .10      ...DOESN'T MATCH
2290  .9     CPY STRLEN   CORRECT LENGTH?
2300         BNE .10      ...NO, KEEP SEARCHING
2310  *---FOUND LABEL, SO GO TO IT-----
2320         JSR AS.GOTO.3
2330         JMP AS.NEWSTT
2340  *---DOESN'T MATCH, TRY NEXT LINE-
2350  .10    LDY #0       GET FORWARD POINTER
2360         LDA (PNTR),Y      LO-BYTE
2370         TAX
2380         INY               HI-BYTE
2390         LDA (PNTR),Y
2400         BNE .6       ...NOT END OF PROGRAM YET
2410  *---END OF PROGRAM, UNDEF LBL----
2420  .11    JMP AS.UNDERR
```

```
==================================================================
DOCUMENT :AAL-8312:DOS3.3:S.Test.STB.128.txt
==================================================================
```

```
 1000   .LIF
 1010  *SAVE S.TEST STB-128
 1020  *---------------------------------
 1030  *      TEST STB-128
 1040  *---------------------------------
 1050  YSAVE  .EQ 0
 1060  LIMIT  .EQ 1
 1070  ADDR   .EQ 2,3
 1080  BANK   .EQ 4
 1090  BYTE   .EQ 5
 1100  SCREEN .EQ 6,7
 1110  *---------------------------------
 1120  SELECT .EQ $C080
 1130  *---------------------------------
 1140  TTTT   JSR TEST
 1150         JSR TEST
 1160         JSR TEST
 1170         JSR TEST
 1180         RTS
 1190  *---------------------------------
 1200  TEST   LDA #0
 1210         STA BANK
 1220         STA ADDR
 1230         JSR $FC58    CLEAR SCREEN
 1240         LDA #$04
 1250         STA SCREEN+1
 1260         LDA #$28
 1270         STA SCREEN
 1280  *---SELECT BANK------------------
 1290  .1     LDA BANK
 1300         STA SELECT+$07
 1310         ORA #$B0     CONVERT TO SCREEN ASCII
 1320         LDY #0
 1330         STA (SCREEN),Y
 1340         LDA SELECT+$03
 1350  *---TEST D000...DFFF-------------
 1360         LDA #$E0
 1370         STA LIMIT
 1380         JSR TEST.ZEROS
 1390         JSR TEST.ONES
 1400         JSR TEST.FIVES
 1410         JSR TEST.AYES
 1420  *---SWITCH TO OTHER D000---------
 1430         LDA SELECT+$0B
 1440         LDA SELECT+$0B
 1450  *---TEST D000...FFFF-------------
 1460         LDA #0
 1470         STA LIMIT
 1480         JSR TEST.ZEROS
```

```
1490        JSR TEST.ONES
1500        JSR TEST.FIVES
1510        JSR TEST.AYES
1520 *---NEXT BANK-------------------
1530        LDA SCREEN
1540        EOR #$80
1550        STA SCREEN
1560        BMI .2
1570        INC SCREEN+1
1580 .2     INC BANK
1590        LDA BANK
1600        CMP #8
1610        BCC .1
1620 *---SWITCH TO ROMS---------------
1630        LDA SELECT+$01
1640        RTS
1650 *--------------------------------
1660 TEST.ZEROS
1670        LDA #0
1680        .HS 2C
1690 TEST.ONES
1700        LDA #$FF
1710        .HS 2C
1720 TEST.FIVES
1730        LDA #$55
1740        .HS 2C
1750 TEST.AYES
1760        LDA #$AA
1770        STA BYTE
1780        LDA #$D0
1790        STA ADDR+1
1800 .1     JSR FILL
1810        JSR COMPARE
1820        INC ADDR+1
1830        LDA ADDR+1
1840        CMP LIMIT
1850        BNE .1
1860        RTS
1870 *--------------------------------
1880 FILL   LDY #0
1890        LDA BYTE
1900 .1     STA (ADDR),Y
1910        INY
1920        BNE .1
1930        RTS
1940 *--------------------------------
1950 COMPARE
1960        LDY #0
1970 .1     LDA (ADDR),Y
1980        CMP BYTE
1990        BNE .3
2000 .2     INY
2010        BNE .1
2020        RTS
```

```
2030  .3     PHA          SAVE ACTUAL DATA
2040         STY YSAVE    SAVE Y-REG
2050         LDA ADDR+1   PRINT ADDRESS OF FAILURE
2060         LDY #2
2070         JSR CONBYTE
2080         LDA YSAVE    LO-BYTE OF ADDRESS
2090         JSR CONBYTE
2100         INY
2110         LDA BYTE     WHAT DATA SHOULD HAVE BEEN
2120         JSR CONBYTE
2130         INY
2140         PLA          WHAT DATA REALLY WAS
2150         PHA          KEEP ON STACK TOO
2160         JSR CONBYTE
2170         INY
2180         PLA          FIGURE WHICH BITS WERE WRONG
2190         EOR BYTE
2200         JSR CONBYTE
2210         LDY #0       DELAY LOOP TO SLOW THINGS DOWN
2220  .4     DEY          FOR OBSERVATION
2230         BNE .4
2240         LDY YSAVE
2250         JMP .2       REJOIN TEST
2260  *-------------------------------
2270  CONBYTE
2280         PHA
2290         LSR
2300         LSR
2310         LSR
2320         LSR
2330         JSR CONNYBBLE
2340         PLA
2350  CONNYBBLE
2360         AND #$0F
2370         CMP #10
2380         BCC .1
2390         ADC #6
2400  .1     ADC #$B0
2410         STA (SCREEN),Y
2420         INY
2430         RTS
2440  *-------------------------------
```

```
================================================================================
DOCUMENT :AAL-8312:DOS3.3:Test.Lbld.GOs.txt
================================================================================
```

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8401:Articles:Bill.Mensch.txt
==============================================================================

More on the new 6502.......................Bob Sander-Cederlof

I talked for about 15 minutes this morning (Dec 16th) with Bill Mensch. Bill used to work at Motorola, and was involved in the design of the original 6800 family there. Chuck Peddle joined the group, and noticed opportunities others were overlooking. Chuck and Bill decided to move to Pennsylvania, and with a few friends founded MOS Technology. They designed and built the 6501 microprocessor, but someone said it looked too much like the 6800 for comfort. Then came the 6502, leading to multiple millions of video games and personal computers. Bill is now at his own design company (Western Design Systems).

Bill told me he designed all the various CMOS versions of the 6502. Now he has designed the 65802 and 65816, CMOS versions with 16-bit registers and 16-megabyte address space. And he is currently working on a 32-bit version!

You probably read about these new versions on page 64 of the December Softalk, or in recent issues of Infoworld or Electronic Design. Elsewhere in Softalk you might also have noticed a box summarizing comments by Woz about plans for a new enhanced Apple //e with 16-megabyte capability. There are probably still other manufacturers out there with boxes and sockets just waiting on the first of Bill's new chips!

I just wish I could convey on paper how excited I am about this new chip! To me, it is as revolutionary as the original microprocessors were in their day. I predict that the 65816 and its successors will prove to be more powerful than the 68008: you will be able to write more compact code that runs faster, and build boards for less money that use less electricity.

With Bill's permission I am re-printing parts of his data sheet. You can get the complete package by calling (602) 962-4545 or writing to Western Design Center, 2166 E. Brown Rd, Mesa, Arizona 85203.

================================================================
DOCUMENT :AAL-8401:Articles:Front.Page.txt
================================================================

$1.80

Volume 4 -- Issue 4                    January, 1984

In This Issue...



News from Apple

Apple sent us a mouse the other day, and we hope to build some software around it.  The mouse came with a disk of graphic software (done by Bill Budge) which makes the plain old Apple II look almost as good as Lisa.  I didn't know your could do all that on a 280x192 screen, and as fast as he does it.  The mouse itself appears identical to the Lisa mouse.  It attaches to a cute red interface card you plug into any slot.  The card has a version of the 6805 microprocessor on it...the kind with internal ROM and RAM which is not visible from the outside world.

Apple also is spreading the word that future Apple //e's are going to have 32 icon characters in the alternate character set.  This probably means some changes to the Cx ROM... (Erv Edge says he hopes that means they are going to fix some bugs, too!)

Another tidbit from Apple is that future //e's will have most of the chips soldered to the motherboard, rather than riding in sockets. They say that should solve most of the remaining reliability problems. In my experience, Apple doesn't have any reliability problems.  And I like sockets, because I like to tinker.  And if something eventually does go bad, it is certainly easier to trade chips than motherboards. Nevertheless, they have made up their minds.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

================================================================
DOCUMENT :AAL-8401:Articles:Interrupt.Patch.txt
================================================================

DOS Patches to Avoid Interrupt Trouble.............Bill Morgan

As we reported a couple of years ago (V2N4, Jan 82), there is a serious problem in using interrupts in the Apple. The Monitor's IRQ interrupt handler uses location $45 to store the contents of the A-register while it is checking to see if the interrupt was from IRQ, or from a BRK instruction. Unfortunately, DOS 3.3 uses $45 for temporary storage in several different routines. If an IRQ interrupt occurs while DOS is active, the Monitor clobbers $45 and DOS can lose a variable.

The usual solution has been to change the Monitor to use some other address to stash the Accumulator. This can be done by copying the Monitor into the RAM card and patching in the new address, or by burning a new Monitor in EPROM and modifying the Apple to accept the chip. The byte that needs changing is at $FA41 in the Autostart ROM, or $FA87 in the Old Monitor ROM.

In the January 84 issue of Washington Apple Pi, Bruce Field reports about the other approach to resolving the conflict. He passes along Wilton Helm's details of the locations in DOS that refer to $45, and how to change things around to safely use interrupts without affecting anything else. Here's Helm's report:

"Location $45 is used at the following places in DOS 3.3:

$A133 $A13E $A158 $A1BE $A1D3 $A1E8 $A1F7 $A1F9 $A201 $A2CC $A767
$A77F $ADBA $AE0A $AE54 $AE58 $BED3 $BF16 $BF39 $BF55 $BF57 $BF5B
$BF9D $BFA3 $BFA5.

These locations should be changed to $46. Location $46 is used for only one purpose, at $BA06 and at $BDA4. These two locations should be changed to $2C. Location $2C is used only by RWTS subroutines and does not conflict with this additional use. The end result is that DOS no longer uses $45 and does not use any new locations."

Field also reports that "these modifications have been made in Universal DOS ... and similar patches have been made in Diversi-DOS."

Bob S-C put together the following Applesoft program to install the patches. The program first checks to make sure that the DOS in memory has not had the patches applied already, then puts them in place. The check beforehand will also avoid clobbering a non-standard DOS.

```
100  REM PREPARE DOS 3.3 FOR INTERRUPTS
110  READ A: IF A = 0 THEN 200
120  IF  PEEK (A) = 69 THEN 110
130  PRINT "THIS DOS IS ALREADY PATCHED": END
140 I = I + 2: GOTO 120
```

```
200  READ A: IF A = 0 THEN 300
210  IF  PEEK (A) = 70 THEN 200
220  PRINT "THIS DOS IS ALREADY PATCHED": END
300  RESTORE
310  READ A: IF A <  > 0 THEN  POKE A,70: GOTO 310
320  READ A: IF A <  > 0 THEN  POKE A,44: GOTO 320
330  END
1000  DATA  41267,41278,41304,41406,41427,41448,41463,41465,
     41473,41676,42855,42879,44474,44554,44628,44632,48851,
     48918,48953,48981,48983,48987,49053,49059,49061,0
1010  DATA  47622,48548,0
```

While we're on the subject of interrupts, I'd like to recommend a book to you: "Real Time Programming - Neglected Topics", by Caxton C. Foster. (Addison-Wesley, 1981. Paperback, $8.95 a couple of years ago.) Foster covers interrupts, ports, timing considerations, A/D conversion, filters, control loops, and communication issues. He points out that these are "enough topics to make up the better part of a full-fledged masters program in electrical engineering or computer science" and that "no book less than 10 inches thick could cover all these topics in detail." Nevertheless, in about 180 pages he does an excellent job of introducing the reader to the material, covering both hardware and software.

==============================================================================
DOCUMENT :AAL-8401:Articles:Lancaster.Books.txt
==============================================================================

A couple of people have pointed out to me that we have advertised Don Lancaster's "Micro Cookbook, Volume II", but have never described it. This one is subtitled Machine Language Programming, and picks up where Volume I left off. He devotes about 450 pages to machine language programming, simple I/O ports, and his Micro Applications Attack method of problem-solving.

Lancaster's method of teaching machine language looks a little strange from my perspective: he says don't even think about an assembler until you have thoroughly learned the instructions from hand assembly. He lays out a system of learning all the instructions and addressing modes by documenting them on 3X5 cards. All his examples refer to the 6502, but the system can be applied to any processor. I suppose this IS a great way to engrave into your memory exactly how a processor works. All this is handled in Don's usual entertaining and enlightening fashion.

This is a good place to mention another book of Lancaster's that has been around for a while: The Hexadecimal Chronicles. This is a huge collection of conversion tables for moving around between ASCII, decimal, hexadecimal and octal (including Apple's negative decimal way of handling addresses.) My TI Programmer calculator is a lot smaller and easier to use, but much more expensive too. If you can find a copy of this book, look it over carefully. It may be exactly what you need.

================================================================================
DOCUMENT :AAL-8401:Articles:LocksmithReview.txt
================================================================================

Locksmith 5.0 Reviewed......................Bob Sander-Cederlof

I received my copy of Locksmith 5.0 last week. I haven't tried any of the lock-busting capabilities, because I have no particular need for that. But there are other features which justify the price. The new manual has information on copy protection schemes which I think has never been published before. The new manual is 140 pages long! I remember the first edition came with a tiny 1/2 page summary of operation!

The other item I am in love with is the fast copy program for ordinary DOS 3.3 disks. In a 64K Apple with two disk drives on one controller, it will make a copy in only 19 seconds! And if you have a larger memory (32K beyond a 128K //e, or a II with a 128K card), it can make a complete copy in only 16 seconds. And if you want to make multiple copies of the same disk, and have large enough memory to hold an entire disk image, you can make additional copies in 8 seconds flat! These copies are without verify, but a verify pass only adds 7 more seconds.

I think this one feature is worth the $99.95 price tag, but there are many more reasons for owning a copy. If you own a previous version, they have an attractive upgrade policy. If not, we will send you a copy for $90 + shipping.

```
===========================================================================
DOCUMENT :AAL-8401:Articles:My.Ad.txt
===========================================================================

S-C Macro Assembler Version 1.0....................................$80.00
S-C Macro Assembler Version 1.1 Update.............................$12.50
Full Screen Editor for S-C Macro Assembler.......................  $49.00
    Includes complete source code.
S-C Cross Reference Utility........................................$20.00
S-C Cross Reference Utility with Complete Source Code..............$50.00
DISASM Dis-Assembler (RAK-Ware)....................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00

S-C Word Processor (the one we use!)...............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk......................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor....................$60.00

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983

Double Precision Floating Point for Applesoft......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research)....................$79.00
Fontrix (Data Transforms)........................................  $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00

Blank Diskettes..................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                               100 or more:  25 cents each
   Cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  5 cents each
ZIF Game Socket Extender...........................................$20.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Understanding the Apple II", Sather.................($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
```

```
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00

  Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Whatever Else You Need............................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8401:Articles:Profiler.txt
==============================================================================

Profiler............................................Bill Morgan

For the last several months, I've been intrigued by an article in the August '83 issue of Byte Magazine, "Chisel Your Code with a Profiler", by Dennis Leas and Paul Wintz. They describe a utility program, called a profiler, which measures where an executing program is spending most of its time. The largest application for such a tool is testing programs compiled from a high-level language. Typically such a program will spend nearly all of its time executing only a small section of the code. Leas and Wintz claim that the proportion is about 90% of the time in about 10% of the code. With a profiler you can identify the bottleneck and speed up the whole program by recoding one small piece.

The profiler first divides your program into sixteen "bins". It then interrupts your program periodically and reads the stored Program Counter from the stack. If the program is in the area you want to measure, it increments one of an array of counters. The profiler then returns control to your program until the next interrupt occurs. When the testing period is finished you can display the counters and spot your problem areas.

An essential part of this tool is a source of regularly timed interrupts. The best place to get a timed interrupt signal is from a suitable clock card. All of the clock cards have some provision for generating interrupts, usually at intervals of about 1 millisecond or 1 second. Some also have available 64 Hz or 256 Hz frequencies, or other values. Check the documentation with your clock to see exactly how to use its interrupt features.

The interrupt timing you want to use will in part be a function of how long your program, or subroutine, will run. If you're profiling a sort that takes several minutes to complete, a 1000 Hz interrupt will overflow the counters long before a significant amount has been done. If the routine takes a short time, a 60 Hz clock won't catch enough hits to be meaningful. Leas and Wintz use a 6 Hz signal picked up from their disk drives to profile a compiler that runs for about 10 minutes.

If all you have available is a high-frequency signal, it's easy enough to divide it down to something usable. Just initialize a counter in the setup portion of the program to the necessary value. Then whenever an interrupt occurs, decrement the counter. Most of the time the counter will be non-zero, so then branch directly to the exit portion of the handler. When the counter reaches zero, go ahead and do the full interrupt processing and then reset the counter.

What if I don't have a clock?, you ask. That is exactly the problem I had when I started thinking about this project. Then I ran across an article in the July 83 issue of Micro in which Charles Putney (a subscriber and sometimes contributor to these pages) told how to get a 60 Hz signal to the interrupt line. Charles' article tells how to use that signal to implement a real-time clock, but it seemed to me that here was exactly the interrupt signal I had been seeking for my profiler.

All you need to do is add a wire inside your Apple, from pin 11 of the 74LS161 at coordinate D11 to pin 4 of the 6502. I used a pair of plunger clips (Radio Shack #270-370, the smallest ones) to attach the wires, and also put a pushbutton in the circuit. When attaching the clips to the IC pins, be EXTREMELY careful not to short any adjacent pins, and try to arrange things so that the wire doesn't wobble around. TURN THE POWER OFF BEFORE MESSING WITH WIRING INSIDE YOUR COMPUTER.

Here's a drawing that shows where to connect the wires:

Note that the photograph in the Micro article does NOT show the correct pins. The description in Putney's text is correct, but whoever did the photo artwork garbled it.

The signal we are borrowing is one of the video timing signals, called V5. V5 is normally high (~5 volts). It goes low every 1/60th of a second, and stays low for about 380 microseconds. That's a pretty good interrupt signal, but we're going to have to allow time for V5 to get back to its high state before we return to the main program, or we'll get more than one interrupt per cycle.

The program

When you BRUN or CALL Profiler, lines 1120-1130 hook the Initialize portion of the program into the monitor's CTRL-Y vector.

Initialize first connects the Handler routine to the IRQ vector (1190-1220). It then gets the starting and ending addresses from where the Monitor left them, takes the difference and divides by 16 to get the size of each bin (1270-1390).

Build Table then starts the table with the Start address and loops to set each table entry Step bytes larger than the previous one (1420-1650). At the same time the routine sets the count for each bin to zero and adds an extra zero byte after the count (1610-1630). This extra byte makes the table easier to read with a Monitor memory dump.

Note that the last entry is set to the End value, rather than a calculated step (1670-1700). This makes the last bin larger than the others, by somewhere between 0 and 15 bytes, to compensate for the remainder left behind when we divided to get Step.

Now we come to the Handler itself. When an IRQ interrupt occurs we first save all the registers on the stack (1740-1790). The next step is to extract the Program Counter value from inside the stack and save it (1800-1840).

The next step is checking that PC value to see if it is inside the range we want (1860-1920). If not, go on to Exit.

If we are in range, search down the table to find the right bin (1930-1980) and register the count (2000-2040). Since the counters only go up to 255, I have the profiler stop when one of them wraps around (2070-2080).

The Exit routine includes a delay loop (2120-2140) to make sure that the Handler takes at least 380 microseconds. This insures that the V5 line we're using for an interrupt source has gone back high, and won't interrupt again as soon as the RTI is done. If you're lucky enough to be getting your interrupts from a clock card you won't need this loop, but you will need to do something to tell the card that you're done with this interrupt. Check your clock manual. Profiler then ends by restoring the registers and doing an RTI.

The Compare Entry routine (2220-2270) just compares the PC value to the current table value.

The funny-looking FILLER space (line 2340) makes sure that the Table begins on a new line in the Monitor memory dump, keeping things easy to read.

Using Profiler

When I want to profile a program, I first assemble Profiler to run somewhere out of the way above or below the program I want to test. Then I enter the Monitor and type addrG to connect Profiler to CTRL-Y. Next I enter addr1.addr2^Y (that's <Start-address>.<End-address><CTRL-Y>) to initialize things.

The next step is to start the program I want to measure, and then start the interrupts coming. My system has a pushbutton between the 60 Hz source and the IRQ line, so I just hold the button down for the period I want to check on. If you're using a clock card you can probably insert instructions into your program to start and stop the interrupts at the points you want.

If one of the counters passes 255 Profiler will Break into the Monitor. Otherwise get into the monitor after your program has finished and examine the table. There's a record of exactly where your program has been.

In a large program, the bin with the highest count may be too wide to really tell where the bottleneck is. If so, just use the control Y command to profile only the bin that had the largest count. This will divide that section into 16 segments so you can see more detail.

Limitations and possible improvements

The profiler described by Leas and Wintz displays the counts as a bar graph, so the largest count really stands out. My version just leaves the addresses and counts where you can read them with the Monitor, so I'm sure you can come up with ways to improve that.

Sometimes it would be nice to be able to build the address table interactively, rather than having it forced to sixteen equal-sized sections. Maybe something like entering the starting address you want for each bin, and a zero at the end.

The DOS problem

There has always been a problem with using interrupts in the Apple II under DOS 3.3, but the solutions are now pretty well-known. Elsewhere in this issue we cover the DOS or Monitor patches necessary to use the 6502's IRQ interrupt without trouble. This program assumes that all that has been taken care of, or that you don't care.

References

1. "Chisel Your Code with a Profiler" Dennis Leas & Paul Wintz. Byte Magazine. August, 1983, pp 286-290.
2. "A Clock Interrupt for Your Apple" Charles Putney. Micro Magazine. July, 1983, pp 36-41.

==============================================================================
DOCUMENT :AAL-8401:Articles:TEXT.TUTORIAL.txt
==============================================================================

WELCOME TO THE S-C WORD PROCESSOR!

The S-C Word Processor turns your Apple into a powerful electronic typewriter: you can type as fast as you like, make corrections instantly, move words and paragraphs around, and much more.

See that blinking rectangle in the top left corner of the screen?  We call that a cursor.

Find the key marked ESC and press it once.  Now look at the cursor:  it has changed to a flashing "+".  Press ESC again, and you will see that it changes back to a plain flashing rectangle.

Now press ESC again, getting the flashing plus.  Press the M key once. Notice that the cursor moves down to the next lower line.  Press M over and over, until the cursor reaches the bottom of the screen.  To continue reading this tutorial, keep typing M. More text will keep appearing at the bottom of the screen, and the text you already have read will disappear off the top of the screen.

You're doing great!  Keep pressing M.

Whenever the cursor is a flashing "+", the four letters I,J,K, and M will move the cursor around the screen.  Look at your keyboard, and you can see that these four letters form a "diamond" pattern, like this:

```
                       I
                      J K
                       M
```

When the cursor is a flashing "+":

```
     I moves the cursor up
     M moves it down
     J moves it left
     K moves it right
```

You can practice a while with these

keys now, if you like.

If you press any other letters, numbers, or punctuation marks while the cursor is a flashing "+", the cursor will change back to a flashing rectangle. Just press ESC again, and you will get the flashing "+" back.

By now you have probably noticed that the cursor always splits the text, making room for itself. This helps you to know exactly where the next character will be placed or where the next command will take effect.

You have also noticed that when you are moving the cursor down, it tends to stay on the left side of the screen. When you are moving the cursor up, it rides the ends of the lines. When you move the cursor left past the beginning of a line, it goes to the end of the previous line; when you move past the end going right, the cursor goes to the beginning of the next line.

All the cursor moves you have tried so far have moved the cursor a short distance: either up or down one line, or left or right one character. If you hold the shift key down, and type I, J, K, or M, you can move the cursor by leaps. J and K move the cursor left or right six characters at a time, while I and M move it up or down 12 lines at a time.

Play with the cursor move controls some more, until you feel fairly comfortable with them. Then come back to this point and continue the tutorial.

-----------------------

Four different cursors are used in the S-C Word Processor. You are pretty handy with one of them already. The two most used ones are the flashing rectangle and the flashing "+". When the cursor is a flashing rectangle, any characters you type will be inserted into the text.

Try it now.  Press ESC until you see
a flashing rectangle cursor, and then
type a few characters.

See them squeezing onto this screen?
(You can delete them if you wish by
using the left arrow key to back up
over them.)

You can change the flashing rectangle
to a flashing "^" cursor by holding
down the CTRL key while you type the
letter C.  (That is called typing a
control-C, or CTRL-C.)  Do this, and
then type a few random letters.  Notice
that they are all CAPITAL?  You are in
the caps-lock mode.  You can get out
of caps-lock mode by typing another
CTRL-C, or by typing ESC a couple of
times.

There is one more cursor:  the flashing
"#".  You get this one by typing
CTRL-C when the cursor is already a
flashing "+".  The flashing "#" serves
two functions.  You can move the cursor
around with the IJKM diamond, but when
you do an amazing thing happens to all
the text you pass over!  Lower case
letters change to capitals, and capitals
change to lower case!  Try playing with this one a while too.  You can
turn off
the flashing "#" by typing another
CTRL-C, or by pressing ESC.

--------------------

Summary of cursors:

```
Cursor   How you use it

       insert text

  ^    insert text all capitalized

  +    move cursor with IJKM

  #    move cursor, changing case
```

#
###################
Now let's get you to type something.
If this explanation scrolls off the
screen, and you want to retrieve it,

use the + cursor and the I key.

Use the M key to move the + cursor
down the page until it's on the blank
line after the "First Official Blank
Line:" after the next paragraph.

Now get ready to type: make a   cursor
by typing the space bar (or any key
except I or J or K or M).  Next, type
a few lines.  When you're done, press
ESC twice to get the + cursor, and use
the M key to see new information.

First Official Blank Line:

The keyboard is very similar to a
typewriter's.  One main difference is
that you can keep typing at the edge of
the screen: whatever you type will simp
ly be displayed on the next line, as sh
own here.  You may have seen this happe
n when you typed in your Official Blank
Line; the phenomenon is often called "
wrap-around".

When you use the Apple Writer to print
text onto paper, wrap-around does not
occur.  Wrap-around has been avoided in
the tutorial to make it easier to read.
The manual discusses the issue in more
detail.

The Apple Writer has many features that
let you modify what you type.  With the
  cursor, the left-arrow key deletes
the character before the cursor.  The
right-arrow key makes deleted
characters reappear.

Here's a chance to try using the left-
arrow and right-arrow keys:

Second Official Blank Line:

Because the Apple screen cannot display
lower case, all the characters you see
on the screen are upper case.  But when
this document is printed on a printer,
you'll see both upper and lower case
letters.  The screen's white-on-black
letters will all be lower case.  The
black-on-white letters, as at the start
of sentences, will print as upper case.

To make upper case letters, press the space bar, the press the ESC key ONCE to make the cursor look like this: ^. The next character you type will be shown black-on-white and will be printed on a printer as upper case. Notice that after typing one character, the cursor looks like this:  .  For now, you must press ESC once to make a ^ cursor before each capital letter.

Here's a chance for you to try typing with upper case letters.

Third Official Blank Line:

Next you'll learn how to leave the editor, and save this file, including your sample typing.  Read everything from the word START to the word END before you type anything.  A summary of the instructions appears after the discussion.

START

You need to know how to use the special command CTRL-Q, called "control Q". To use CTRL-Q, hold down the key marked "CTRL" while you press the Q.

To leave the editor, use:
ESC ESC CTRL-Q
You'll next see the editor menu.  The menu tells you things that you can do at that point.

If you turn off your Apple without saving the file you are working on, you will loose the file and all the changes you made.  The Save option on the editor menu allows you to save the file you've been editing on a diskette.

To save a file, use:
S
and press the RETURN key.  You will be asked the question
  USE "TUTORIAL"
  AS FILE NAME (Y/N) ?
since the editor remembers that you last loaded a file called "TUTORIAL".

It's not a good idea to save this file using the same name, since this file now contains your sample typing. So when asked whether to use "TUTORIAL" as the file name, type N for no, then press the RETURN key. You'll then see the message

```
  ENTER FILE NAME:
```

Choose a name for the file. Let's assume you call it XXX. Type this:
XXX
and press the RETURN key. The light on the disk drive will come on, and the disk drive will whirr as the file is stored.

Next check to see that things really worked as reported: turn off the Apple, turn it on again, then Load your file XXX and choose the Edit option.

Look though the file. Your sample typing should be there, a clear sign that you've successfully created a file using the Apple Writer. Finally, read the few paragraphs that follow the summary.

Here's the summary of commands you'll need to use to save the file and return to the editor.

```
WHAT YOU TYPE:   WHAT HAPPENS:

ESC ESC CTRL-Q  editor menu displayed

S               USE "TUTORIAL" AS NAME?

N               told "ENTER FILE NAME:"

XXX             drive light comes on,
                "SELECT:" displayed

Turn OFF the Apple, then turn it ON.

L               told "ENTER FILE NAME:"

XXX             drive light comes on,
                "SELECT:" displayed

E               you're at the start of
                the file XXX; read the
                paragraphs after this.
```

END: Now try saving the file.

The last editor menu option, Quit, returns you to BASIC. If you stop using the editor and then want to stop using your Apple, you can turn it off instead of using Quit.

The time has come to quit, even though we've barely scratched the surface of all the things you can do with this editor. We haven't discussed the editor's ability to look for a word or series of characters and substitute any other word or series of characters of your choice in its place. Many other things await you in your Apple Writer manual, so type this:
ESC ESC CTRL-Q
to get to the editor menu, then choose the Quit option, and start reading!

BYE

```
==============================================================================
DOCUMENT :AAL-8401:Articles:ThreeSuitPieces.txt
==============================================================================
```

It Was a Bad Dream, I Think................Bob Sander-Cederlof

For two hours two nights ago I tossed, turned, wrestled, and wrote a speech on trends in our favorite industry.  I think it went like this....

3-piece Suits

Woz likes blue jeans and jogging shoes.  Engineers and programmers tend to put their craft ahead of their tailors.  And the most productive rank skills before degrees.

But once an industry starts creating wealth, the business grads and 3-piece suiters quickly rise to the top.

Woz worked in a little cubicle at Hewlett-Packard.  With their blessing he left with the seeds of the most munificent Apple tree ever.  Now he is back to working in a cubicle.  Are other seeds incubating?  Will they stay in the same orchard?

Lawsuits

Another kind of action is drawn by the magnet of success:  legal. Friends suing friends for more than they ever made.  Visicorp suing Software Arts for $50 million:  "You were too slow putting advanced Visicalc onto the IBM-PC."  Software Arts suing Visicorp for $87 million:  "You didn't promote Visicalc well enough."

United Computer Corporation (why buy when you can rent) being sued by MicroPro and others.  Maybe some people only rent so they can make their own copies.  In any case, UCC shouldn't remove the license agreements from the packages!

UCC has also earned some lawsuits over their advertising debts.  They prepay the first month, and ask for 30-day terms to run until further notice.  That was last April...we caught on in July.

Following Suit

The whole world seems to be going IBM.  Last year it was all CP/M. Next year it may be all AT&T.  Remember back when everyone was copying Apple?

Businessmen buy those computers having the most on-going software and hardware development.  Developers, programmers, cloners, and other entrepreneurs gather around systems businessmen are buying.  The boys go where the girls are, which is where the boys are, which is where the girls are....  Being popular is so popular!

All of which slows down innovation in the marketplace. Not that innovation is all good and popularity is all bad. One secret of success is to stay the same long enough. Apple II/Plus/e has presented a stable yet growing environment for developers...contrast with Commodore/OSI/Radio Shack and their strings of mutually incompatible environments.

But innovators brought us the computer. And the supercomputer. And the minicomputer. And the microcomputer. And the Apple. And the....

==========================================================================
DOCUMENT :AAL-8401:Articles:Understanding.txt
==========================================================================

"Understanding the Apple II", a Review.....Bob Sander-Cederlof

If you want the real inside scoop on the Apple II, you need "Understanding the Apple II". Following close on the heels of Gayler's "Apple II Circuit Description", this book is no second-place sequel.

"Understanding..." was written by Jim Sather, a former ITT technical representative, after many moons of trial-and-error, pick-and-shovel research into the inner sanctum of our favorite computer. Jim has a gift for clearly explaining how things work. My degree is a little rusty, or mildewed, or whatever, and hardware never was my long suit. But Jim makes it all make sense for me.

The process of "understanding" starts with a few full color diagrams and charts. In the back of the book there are two foldout full color charts of bus structure and chip layout. Surprisingly, you find color sprinkled throughout the book, along with many black & white illustrations, photos, tables, diagrams, etc.

Sather describes microcomputer fundamentals with specific applications to the Apple II. He carefully documents all the circuits on the motherboard, as well as the firmware and language cards, and Wozniak's patented disk controller.

The chapter on the disk drive and controller is especially thorough, devoting some 45 large pages, including many diagrams, to the exact workings of these devices. I have never seen a better explication of the Apple's unique disk controller.

There are especially useful discussions of address decoding, RAM/ROM addressing, and bus structure. Sather's readable style avoids much of the reference-book prose common to authoritative technical books.

Each chapter ends with some of nearly two dozen hardware & software projects, including reprogramming screen character sets, an NMI based single stepper, detecting and using television sync, modifying the firmware card so the F8 ROM can be switch-selected, and more.

"Understanding..." begins with a foreword by Steve Wozniak, and ends near an appendix describing a conversation with Woz about some of the original design decisions that made our Apples what they are today.

This would be a good text book in computer hardware fundamentals at high school level or above. Most of the courses I took in college (now over 25 years ago!) were rather abstract and difficult to relate to real applications. What better way to understand how computers work, how they can be modified and maintained, and how to design them, than to dissect a living breathing example like the Apple II!

Here's a quick look at the structure of "Understanding the Apple II":

Chapters



Glossary of 7 pages, about 150 entries.

Appendices: references, trademarks, 6502 data sheets, program listings, logic circuits primer, number systems primer, apple ii revisional info, historical notes, conversation with Woz, how to remove the motherboard, list of figures and tables.

Schematics

Index

"Understanding the Apple II" describes the Apple II and Apple II Plus. Much of the book's information, especially the chapter on the disk controller, applies also to the Apple //e. "Understanding the Apple //e" is promised sometime in 1984.

Understanding the Apple II, Jim Sather. About 356 pages. Quality Software, $22.95 (Buy it from us for only $21 + shipping).

=========================================================================
DOCUMENT :AAL-8401:Articles:Urschels.Color.txt
=========================================================================

68000 "Color Pattern"...............................Bob Urschel
Valparaiso, Indiana

I have had my QWERTY Q68 board for about 2 weeks now. In my opinion this seems to be an excellent product and also a very inexpensive way to learn about the MC68000 MPU.

As an exercise I rewrote an Integer BASIC program called "ROD'S COLOR PATTERN" found in my red Apple II Reference Manual (1978). I am sending you two versions of my program. the first version does the calculation for the LORES screen base addresses. The second version looks up the base addresses in a table, consequently running slightly faster than the first version. The second version runs (as close as I can tell) about 50 times faster than the Integer BASIC program.

[ We're printing only the first version, since the GBASCALC routine is more interesting than a table lookup. QD 14 will include both version... Bill ]

After the 68000 source code has been assembled, I BRUN a very short 6502 program which consists of the following code:

```
       .OR $1080
       JSR $30B    TURN ON THE Q68 BOARD
HERE   JMP HERE    DON'T DO ANYTHING
```

This keeps the 6502 busy while the 68000 is doing all the work.

Here's a listing of ROD'S COLOR PATTERN in Integer BASIC:

```
10 GR
20 FOR W = 3 T0 50
30 FOR I = 1 TO 19
40 FOR J = 0 TO 19
50 K = I+J
60 COLOR = J*3 / (I+3) + I*W/12
70 PLOT I,K: PLOT K,I: PLOT 40-I,40-K: PLOT 40-K,40-I
80 PLOT K,40-I: PLOT 40-I,K: PLOT I,40-K: PLOT 40-K,I
90 NEXT J: NEXT I: NEXT W
100 GOTO 20
```

==============================================================================
DOCUMENT :AAL-8401:Articles:V4N4.6502.NOTES.txt
==============================================================================

Notes on the 6502 from various sources

EDN magazine, Nov 10, 1983, page 194, summarized the 650x and 65C0x processors. Softalk, Dec 1983, page 64, in an article about Hayden Software, discussed two new versions of this family.

Original design by MOS Technology (bought out by Commodore). Made by Commodore, Rockwell, Synertek, GTE, NCR, Ricoh, and Western Design Center.

6502...NMOS...available now in 1, 2, or 3 MHz...$5 each in 100 quantity for 1MHz.

65C02...CMOS...available now in 1MHz at $8.55 each or 2 MHz at $9.40. 3 and 4MHz coming in '84.

These clock rates are slow compared to Z-80, 68000, etc.; however, each instruction takes only 2-7 clock cycles. Thus a 1MHz 6502 is roughly the same net speed as a 4MHz Z-80.

65802...Adds 16-bit registers and operations using status bit. Available soon from Western Design Center, phone (602) 962-4545. See Softalk.

65816...Adds 16-bit registers and operations, plus 24-bit address-bus for direct access to 16 megabytes. Available soon from Western Design Center. See Softalk.

6500 family still has the highest volume of any 8-bit microprocesser...15 million in 1982! Commodore uses all the chips they make in their own products, so second sources supply the rest of the world's needs. Even without the new 16-bit enhancements, this chip will probably continue to be used in new designs for at least 5 years.

=============================================================================
DOCUMENT :AAL-8401:Articles:Woz.Online.txt
=============================================================================

On-Line with Steve Wozniak

Steve Knouse sent a printout of the "on-line with Woz" session you may have heard about. Some intriguing Woz-words:

About ProDOS use of >64K memory: "Our enhanced //e family is headed toward 16M bytes in short time with a revolutionary 6502-based processor."

About software for extended RAM cards: "I promise an alternative solution soon (6 mo?) for direct addressing of 24-bit address."

About MAC: "...look at LISA. Then imagine slightly fewer resources and memory but advantage taken to make it faster and better with fewer resources (sound familiar //e world?). Mouse, no color, no slots, finest software (BASIC and Pascal are finest ever done too). MAC will use its own op-sys which was developed to handle the user interface of LISA more directly with better performance. Such good software has been written for MAC (128K bytes in ROM) that it will be transferred to LISA soon!" "Initially MAC won't displace the PC as a small business machine but is intended to be a more finished product for the bulk of the personal market -- assuming which peripherals and features they would want and supplying them at lower cost than if they have slots to make their own choices. Interesting." "I believe that MAC is the most revolutionary computer of all time -- not that what it does hasn't been done before, but that it hasn't been done at a price which will wind up with millions experiencing it." "The MAC unfortunately is so perfect that we didn't leave much room for hackers to do hardware 'for themselves' or 'their own way' -- we feel there were no alternatives. The philosophy on software is different -- open, access the hardware at various levels."

About larger ProFILEs: "...yes, plans for larger ProFILEs. Pretty the minimal hard disk for small business has grown to 10MB, soon 20."

About the Apple: "The Apple II was not built to be a product for sale. It looked like the best thing available in 1976. The first computer ever (low cost) with color, hi-res, Basic in ROM, plastic case, switching power supply, dynamic memories, paddles, speaker, cassette, etc, all STANDARD. Look at virtually every "personal" computer since. We needed $250,000 to build a thousand--where do you get that kind of money when you're a couple of kids with no business experience? We sought venture money and Mike Markkula agreed to HELP us write a business plan. He realized we were onto something that happens once a decade -- a huge market expanding out of nothing. He joined us (equal partner) and loaned $250,000. He told me I had to quit HP and go 100% Apple. HP is a good company and it's hard to leave any company for anything when you believe it's good to its employees. I said "NO" on my ultimatum day and we were not going to do Apple. Steve Jobs was (in tears) and got relatives and friends of mine to call me at work and tell me why I should start Apple. Finally I realized I could have a great time doing the one important thing in my life -- design computers for myself and start the company to make money and in my head they didn't have to be dependent. So I turned around. Markkula decided that he and Jobs had better have 52% of Apple combined -- I realize now that they were probably afaid I was a little unpredictable. A true story."

About a faster //e: "The Accelerator [Saturn, 3.58MHz 6502 with 64K RAM] is my favorite card, largely because without any fancy jumpers EVERYTHING ran with it. The only exception with the software I use is Word Juggler under ProDOS. The current Accelerator should have problems with the //e extended memory usage once software uses it. I heard that they are working on a new one to get around this. Its amazing to see everything work faster. My main direction on return to Apple was to get 3.6 MHz built in. Look for it someday. Saturn has shown it's possible."

==============================================================================
DOCUMENT :AAL-8401:DOS3.3:Ptch.DOS33.IRQ.txt
==============================================================================

```
%d≤PREPARE DOS 3.3 FOR INTERRUPTS5náA:≠A–0ƒ200Fx≠,(A)–69ƒ110kÇ∫"THIS
DOS IS ALREADY PATCHED":ÄzåI–I»2:´120ä»áA:≠A–0ƒ300õ“≠,(A)–
70ƒ200¿‹∫"THIS DOS IS ALREADY PATCHED":Ä∆,Æfi6áA:≠A —
œ0ƒπA,70:´310ˆ@áA:≠A—œ0ƒπA,44:´320¸JÄö    ËÉ
41267,41278,41304,41406,41427,41448,41463,41465,41473,41676,42855,4287
9,44474,44554,44628,44632,48851,48918,48953,48981,48983,48987,49053,49
059,49061,0Æ    ÚÉ 47622,48548,0
```

```
======================================================================
DOCUMENT :AAL-8401:DOS3.3:Rods.Clr.Pat.txt
======================================================================

   10  L
   20
U333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333
   30
U111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111
   40
U000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
   50
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIW

   60  f33333333333333333333
   70           (
   80  g                                                    (
   90
YYYYYYYg44444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444(
  100  _
```

==============================================================================
DOCUMENT :AAL-8401:DOS3.3:S.PROFILER.txt
==============================================================================

```
 1000  *SAVE S.PROFILER
 1010  *--------------------------------
 1020  A2L    .EQ $3E
 1030  A2H    .EQ $3F
 1040  A3L    .EQ $40
 1050  A3H    .EQ $41
 1060
 1070  STACK      .EQ $100
 1080  IRQ.VECTOR .EQ $3FE
 1090  CONTROL.Y.VECTOR .EQ $3F9
 1100  *--------------------------------
 1110         .TF PROFILER
 1120         LDA /INITIALIZE
 1130         STA CONTROL.Y.VECTOR+1
 1140         LDA #INITIALIZE
 1150         STA CONTROL.Y.VECTOR
 1160         RTS
 1170  *--------------------------------
 1180  INITIALIZE
 1190         LDA #HANDLER      install vector
 1200         STA IRQ.VECTOR
 1210         LDA /HANDLER
 1220         STA IRQ.VECTOR+1
 1230         LDA #0       initialize variables
 1240         STA HITS
 1250         STA HITS+1
 1260         SEC
 1270         LDA A2L
 1280         SBC A3L      calculate step size
 1290         STA STEP
 1300         LDA A2H
 1310         SBC A3H
 1320         BCC ERROR    end<start
 1330         STA STEP+1
 1340
 1350         LDX #3       divide STEP by 16
 1360  .1     LSR STEP+1   (shift it right 4)
 1370         ROR STEP
 1380         DEX
 1390         BPL .1
 1400
 1410  BUILD.TABLE
 1420         LDA A3L       first table entry
 1430         STA TABLE     is start address
 1440         LDA A3H
 1450         STA TABLE+1
 1460         LDX #0
 1470         STX TABLE+2   zero count
 1480         STX TABLE+3   and fill byte
```

```
1490
1500  .1     INX           next entry
1510         INX
1520         INX
1530         INX
1540         CLC
1550         LDA TABLE-4,X
1560         ADC STEP      add step size to
1570         STA TABLE,X   last entry
1580         LDA TABLE-3,X
1590         ADC STEP+1
1600         STA TABLE+1,X
1610         LDA #0
1620         STA TABLE+2,X zero count
1630         STA TABLE+3,X and fill byte
1640         CPX #$3C      done?
1650         BCC .1        no
1660
1670         LDA A2L
1680         STA TABLE+4,X and make last entry
1690         LDA A2H       equal end
1700         STA TABLE+5,X
1710  ERROR  RTS
1720  *--------------------------------
1730  HANDLER
1740         LDA $45      get A back from where
1750         PHA          Monitor stashed it
1760         TXA
1770         PHA          save registers
1780         TYA
1790         PHA
1800         TSX
1810         LDA STACK+6,X     get PC from stack
1820         STA PCH
1830         LDA STACK+5,X
1840         STA PCL
1850  *---SEARCH TABLE----------------
1860         LDX #0            compare PC to start
1870         JSR COMPARE.ENTRY   of table
1880         BCC EXIT          below table
1890         LDX #$40          and compare to
1900         JSR COMPARE.ENTRY   end of table
1910         BCS EXIT          above table
1920
1930  .1     DEX               next entry
1940         DEX
1950         DEX
1960         DEX
1970         JSR COMPARE.ENTRY
1980         BCC .1            not there yet
1990
2000         INC HITS      count hit in total
2010         BNE .2
2020         INC HITS+1
```

```
.2     INC TABLE+2,X count hit in bracket
       BNE EXIT

* counter overflowed, so put it back to $FF and end
       DEC TABLE+2,X
       BRK
* ... do whatever it takes to clean up the stack
*     and display the results

EXIT   LDY #55      delay about 275 usec so
.1     DEY            V5 will be high on exit
       BNE .1
       PLA          restore registers
       TAY
       PLA
       TAX
       PLA
       RTI          and exit
*---------------------------------
COMPARE.ENTRY
       LDA PCL
       CMP TABLE,X
       LDA PCH
       SBC TABLE+1,X
       RTS
*---------------------------------
VARIABLES
PCL    .BS 1        program counter
PCH    .BS 1
HITS   .BS 2        total count
STEP   .BS 2        bracket size
FILLER .BS */8*8+8-* align table
*---------------------------------
TABLE  .EQ *
```

```
==============================================================================
DOCUMENT :AAL-8401:DOS3.3:S.Urschel.ClPat.txt
==============================================================================

 1000  *SAVE S.URSCHEL'S COLOR PATTERN
 1010  *--------------------------------
 1020  *   RODS COLOR PATTERN
 1030  *   RE-WRITTEN BY BOB URSCHEL
 1040  *   USING THE QWERTY Q68 MC68000 MPU
 1045  *
 1047         .OR     $1000
 1050         MOVE.L  #$1100,A0  MOVE PROGRAM TO FAST MEMORY
 1060         MOVE.L  #$18600,A1
 1070         MOVE    #END-START,D1
 1080  XFER   MOVE.B  (A0)+,(A1)+
 1090         DBF     D1,XFER
 1095         JMP     $18600
 1100  *
 1110  *--------------------------------
 1120  *
 1140         .OR     $18600
 1150         .TA     $1100
 1160  START
 1170         TST.B   $C050     >GR
 1180         BSR     CLRSCR    CLEAR SCREEN
 1190  *--------------------------------
 1200  START.W
 1210         MOVE.B  #3,W      >FOR W = 3 TO 50
 1220  START.I
 1230         MOVEQ   #1,D7     >FOR I = 1 TO 19
 1240  START.J
 1250         MOVEQ   #0,D3     >FOR J = 0 TO 19
 1260  SET.K  MOVE    D7,D6     >K = I + J
 1270         ADD.B   D3,D6
 1280  *--------------------------------
 1290         MOVEQ   #0,D0     >COLOR = J*3/(I+3)+I*W/12
 1300         MOVE    D3,D0
 1310         MULU    #3,D0     J*3
 1320         MOVEQ   #0,D1
 1330         MOVE    D7,D1
 1340         ADDQ    #3,D1     I+3
 1350         DIVU    D1,D0     J*3/(I+3) --> D0
 1360         MOVE    D7,D1
 1370         MOVEQ   #0,D2
 1380         MOVE.B  W,D2
 1390         MULU    D1,D2     I*W --> D2
 1400         DIVU    #12,D2    D2 / 12
 1410         ADD     D0,D2
 1420         ANDI.B  #$F,D2
 1430         MOVE.B  D2,COLOR  SET COLOR
 1440  *
 1450  *
 1460  *  SUBTRACT I AND K FROM 40
```

```
1470  *
1480         MOVEQ   #40,D5
1490         SUB     D7,D5     D5 = 40 - I
1500         MOVEQ   #40,D4
1510         SUB     D6,D4     D4 = 40 - K
1520         MOVE    D7,D0     >PLOT I,K
1530         MOVE    D6,D1
1540         BSR.S   PLOT
1550         MOVE    D6,D0     >PLOT K,I
1560         MOVE    D7,D1
1570         BSR.S   PLOT
1580         MOVE    D5,D0     >PLOT 40-I,40-K
1590         MOVE    D4,D1
1600         BSR.S   PLOT
1610         MOVE    D4,D0     >PLOT 40-K,40-I
1620         MOVE    D5,D1
1630         BSR.S   PLOT
1640         MOVE    D6,D0     >PLOT K,40-I
1650         MOVE    D5,D1
1660         BSR.S   PLOT
1670         MOVE    D5,D0     >PLOT 40-I,K
1680         MOVE    D6,D1
1690         BSR.S   PLOT
1700         MOVE    D7,D0     >PLOT I,40-K
1710         MOVE    D4,D1
1720         BSR.S   PLOT
1730         MOVE    D4,D0     >PLOT 40-K,I
1740         MOVE    D7,D1
1750         BSR.S   PLOT
1760         ADDQ    #1,D3     >NEXT J
1770         CMPI    #20,D3
1780         BNE     SET.K
1790         ADDQ    #1,D7     >NEXT I
1800         CMPI    #20,D7
1810         BNE     START.J
1820         ADDQ.B  #1,W      >NEXT W
1830         CMPI.B  #51,W
1840         BEQ     START.W
1850         BNE     START.I
1860  *
1870  *-------------------------------
1880  *      PLOT SUBROUTINE
1890  *
1900  *      A0 = SCREEN ADDRESS
1910  *      D0 = X-COORD
1920  *      D1 = Y-COORD
1930  *      D2 = WORK REGISTER
1940  *
1950  PLOT   MOVE    D0,A0     SAVE X-COORD
1960         LSR.B   #1,D1     GET CARRY
1970         MOVE    SR,D0     SAVE ODD-EVEN STATUS
1980         BSR.S   GBASCALC
1990         ADD     D1,A0     FINAL SCREEN ADDR
2000         MOVE.B  #$F0,MASK
```

```
2010         MOVE.B  COLOR,D1
2020         MOVE    D0,CCR    ODD OR EVEN?
2030         BCC.S   PLOT1     EVEN...
2040         MOVE.B  #$F,MASK
2050         LSL.B   #4,D1     ROTATE COLOR
2060  PLOT1  MOVE.B  (A0),D2   ORIGINAL BYTE
2070         AND.B   MASK,D2   MASK OUT OLD COLOR
2080         OR.B    D1,D2      AND GET NEW COLOR
2090         MOVE.B  D2,(A0)   PLOT TO SCREEN
2100         MOVEQ   #0,D0     CLEAR OUT CCR
2110         MOVEQ   #0,D1
2120         RTS
2130  *
2140  *      CALCULATE BASE ADDRESS
2150  *
2160  GBASCALC
2170         MOVE    D1,D2     000DEFGH
2180         AND.B   #$18,D1   000DE000
2190         LSL.B   #5,D2     FGH00000
2200         OR.B    D2,D1     FGHDE000
2210         MOVE.B  D1,D2
2220         AND.B   #$18,D2   000DE000
2230         LSR.B   #2,D2     00000DE0
2240         OR.B    D2,D1     FGHDEDE0
2250         OR      #$100,D1 1FGHDEDE0
2260         LSL     #2,D1  1FGHDEDE000
2270         RTS
2280  *
2290  *      CLEAR LORES SCREEN
2300  *
2310  CLRSCR CLR     D0
2320         MOVE    #511,D1   # OF WORDS TO MOVE MINUS 1
2330         MOVE    #$800,A0  ENDING SCREEN ADDR
2340  .1     MOVE    D0,-(A0)
2350         DBF     D1,.1
2360         RTS
2370  *--------------------------------
2380  *      WORK AND STORAGE
2390  *
2400  MASK   .BS     1
2410  COLOR  .BS     1
2420  W      .BS     1
2430  *
2440  *--------------------------------
2450  END
2460         .OR     $800
2470         .DA     $18800
2480         .DA     $1000
```

==============================================================================
DOCUMENT :AAL-8401:DOS3.3:S.Urschel.table.txt
==============================================================================

```
*SAVE S.URSCHEL'S COLOR PATTERN.TABLE
*---------------------------------
*   RODS COLOR PATTERN
*   RE-WRITTEN BY BOB URSCHEL
*   USING THE QWERTY Q68 MC68000 MPU
*
       .OR     $1000
       MOVE.L  #$1100,A0  MOVE PROGRAM TO FAST MEMORY
       MOVE.L  #$18600,A1
       MOVE    #END-START,D1
XFER   MOVE.B  (A0)+,(A1)+
       DBF     D1,XFER
       JMP     $18600
*
*---------------------------------
*
       .OR     $18600
       .TA     $1100
START
       TST.B   $C050     >GR
       BSR     CLRSCR    CLEAR SCREEN
*---------------------------------
START.W
       MOVE.B  #3,W      >FOR W = 3 TO 50
START.I
       MOVEQ   #1,D7     >FOR I = 1 TO 19
START.J
       MOVEQ   #0,D3     >FOR J = 0 TO 19
SET.K  MOVE    D7,D6     >K = I + J
       ADD.B   D3,D6
*---------------------------------
       MOVEQ   #0,D0     >COLOR = J*3/(I+3)+I*W/12
       MOVE    D3,D0
       MULU    #3,D0     J*3
       MOVEQ   #0,D1
       MOVE    D7,D1
       ADDQ    #3,D1     I+3
       DIVU    D1,D0     J*3/(I+3) --> D0
       MOVE    D7,D1
       MOVEQ   #0,D2
       MOVE.B  W,D2
       MULU    D1,D2     I*W --> D2
       DIVU    #12,D2    D2 / 12
       ADD     D0,D2
       ANDI.B  #$F,D2
       MOVE.B  D2,COLOR  SET COLOR
*
*
*  SUBTRACT I AND K FROM 40
```

```
*
        MOVEQ   #40,D5
        SUB     D7,D5     D5 = 40 - I
        MOVEQ   #40,D4
        SUB     D6,D4     D4 = 40 - K
        MOVE    D7,D0     >PLOT I,K
        MOVE    D6,D1
        BSR.S   PLOT
        MOVE    D6,D0     >PLOT K,I
        MOVE    D7,D1
        BSR.S   PLOT
        MOVE    D5,D0     >PLOT 40-I,40-K
        MOVE    D4,D1
        BSR.S   PLOT
        MOVE    D4,D0     >PLOT 40-K,40-I
        MOVE    D5,D1
        BSR.S   PLOT
        MOVE    D6,D0     >PLOT K,40-I
        MOVE    D5,D1
        BSR.S   PLOT
        MOVE    D5,D0     >PLOT 40-I,K
        MOVE    D6,D1
        BSR.S   PLOT
        MOVE    D7,D0     >PLOT I,40-K
        MOVE    D4,D1
        BSR.S   PLOT
        MOVE    D4,D0     >PLOT 40-K,I
        MOVE    D7,D1
        BSR.S   PLOT
        ADDQ    #1,D3     >NEXT J
        CMPI    #20,D3
        BNE     SET.K
        ADDQ    #1,D7     >NEXT I
        CMPI    #20,D7
        BNE     START.J
        ADDQ.B  #1,W      >NEXT W
        CMPI.B  #51,W
        BEQ     START.W
        BNE     START.I
*
*---------------------------------
CLRSCR CLR      D0
        MOVE    #511,D1   # OF WORDS TO MOVE MINUS 1
        MOVE    #$800,A0  ENDING SCREEN ADDR
.1      MOVE    D0,-(A0)
        DBF     D1,.1
        RTS
*---------------------------------
PLOT    LEA     SCREEN.ADDR,A0
        MOVE.B  D1,D2     SAVE Y-COORD
        ANDI.B  #$FE,D1   GET INDEX INTO TABLE
        MOVE    0(A0,D1),A0
        ADD     D0,A0     FINAL SCREEN ADDRESS
        MOVE.B  #$F0,MASK
```

```
2030          MOVE.B  COLOR,D1
2040          LSR.B   #1,D2     ODD OR EVEN?
2050          BCC.S   PLOT1     EVEN..
2060          MOVE.B  #$F,MASK
2070          LSL.B   #4,D1
2080   PLOT1  MOVE.B  (A0),D2   GET ORIGINAL BYTE
2090          AND.B   MASK,D2
2100          OR.B    D1,D2     NEW COLOR
2110          MOVE.B  D2,(A0)
2120          RTS
2130   *
2140   *
2150   SCREEN.ADDR
2160          .DA     /$400
2170          .DA     /$480
2180          .DA     /$500
2190          .DA     /$580
2200          .DA     /$600
2210          .DA     /$680
2220          .DA     /$700
2230          .DA     /$780
2240          .DA     /$428
2250          .DA     /$4A8
2260          .DA     /$528
2270          .DA     /$5A8
2280          .DA     /$628
2290          .DA     /$6A8
2300          .DA     /$728
2310          .DA     /$7A8
2320          .DA     /$450
2330          .DA     /$4D0
2340          .DA     /$550
2350          .DA     /$5D0
2360          .DA     /$650
2370          .DA     /$6D0
2380          .DA     /$750
2390          .DA     /$7D0
2400   *
2410   *--------------------------------
2420   *      WORK AND STORAGE
2430   *
2440   MASK   .BS     1
2450   COLOR  .BS     1
2460   W      .BS     1
2470   *
2480   *--------------------------------
2490   END
2500          .OR     $800
2510          .DA     $18800
2520          .DA     $1000
```

==============================================================================
DOCUMENT :AAL-8402:Articles:Biblio.68000.txt
==============================================================================

Annotated 68000 Bibliography........................Bill Morgan

Here is a quick look at some of the books and articles about the 68000 that I have found to be helpful.

Another possible source of 68000 information is the newsletter "DTACK Grounded", published by Digital Acoustics, 1415 E. McFadden, Suite F, Santa Ana, CA 92705.  I've only seen one or two issues, back before I got interested in 68000, so I don't know exactly what they've been up to lately.  I'll be finding out soon and pass it on.  I might note that the issue I have (#7, Feb-Mar 1982) contains about 12 pages of more-or-less interesting gossip, and no code.  I don't know if that is typical.

Books:

68000 Assembly Language Programming.  Gerry Kane, Doug Hawkins & Lance Leventhal.  OSBORNE/McGraw-Hill, 1981.
The Leventhal book.  Need I say more?  Recommended.

The 68000:  Principles and Programming.  Leo. J. Scanlon. Blacksburg/Sams, 1981.
Tutorial.  Looks pretty good.  Recommended.

MC68000 16-bit Microprocessor User's Manual, third edition. Motorola/Prentice-Hall, 1982.
Motorola's manual.  THE basic reference.  There is a fourth edition coming this year (1984).  There is also a Mostek version of this book, but the Motorola edition is better.

MK68000 Microcomputer Programming Reference Guide.  Mostek Corp, 1981.
A 42-page Quick Reference Card.  Isn't that a bit much?

Programming the M68000.  Tim King and Brian Knight.  Addison-Wesley, 1983.
Tutorial.  Looks very good.  Lots of examples, building up to a simple monitor/debugger.  Recommended.

Articles:

Design Philosophy Behind Motorola's MC68000.  Thomas W. Starnes (of Motorola, Inc.)  Byte.  April-June, 1983 (3 parts).
Very good.  Lives up to the title.  Recommended.

68000 Instructions and Addressing Modes.  Joe Hootman.  Micro.  #'s 52,54-57,60-62 (8 parts).
Summaries of the instruction set.  OK if you already have the stack of Micro back issues.

An MC68000 Overview. Joe Jelemensky & Tom Whiteside. Micro. #'s 52,54.
Some good examples of the instructions at work.

```
==============================================================================
DOCUMENT :AAL-8402:Articles:Creamers.Erase.txt
==============================================================================
```

Text Area Erase Routine............................Jeff Creamer
Yavapai College, Prescott AZ

Good programs interact frequently with their users, providing error messages, helpful prompts, and information about what the program is doing. For programmers, this raises the question of what to do with the messages once they have been printed, especially if you want to get rid of them while leaving the rest of the screen intact.

I have used several strategies to clear specific areas of the text screen. The simplest solution, and probably the most commonly used, is to place all messages at the end of the page. Then you can HTAB and VTAB to the first character of the message and CALL the Monitor routine at $FC42 (CLREOP). From Applesoft, CALL -958. Such messages must be kept to the lower part of the screen, however, and the method can interfere with decorative borders, etc., placed around your screens.

Another thing I have done is to print strings of blanks over the offending message. I use a loop to HTAB and VTAB to the left margin of the message area, incrementing the vertical coordinate each time, then printing a string variable set to a predetermined number of blanks. This method is slow, but not unbearable. Still, it is clumsy and wastes memory storing the blanks.

Of course, instantaneous clears of a given area are easily done by resetting the text window through POKEing values to locations $25-$28, then executing a HOME. This requires POKEing 4 values before the clear, however, and POKEing 4 coordinates to reset the current window when you are done (or "TEXT" to reset the default window). Downright unpleasant. For a time I resorted to this method to protect my decorative borders, however.

Now I have come up with a routine that I think is an improvement over the above. It clears rectangular areas of the text screen given the width and depth (number of lines) needed. Because it uses the Monitor COUT routine, it should also work with those hi-res character generator utilities that interface to the normal output hooks, giving a controlled hi-res screen clear. While it requires Applesoft in ROM, it is fully relocatable, making it ideal for people who use Ampersand utilities like AmperMagic or The Routine Machine.

The routine, which I call "ERASE", is used by first HTABing and VTABing to the upper left corner of the area to be cleared. Then CALL the routine giving the width and depth of the area to be cleared, using commas, like so:

```
     CALL ADDRESS,WIDTH,DEPTH
```

For example, assume you BLOAD the routine at $300, the most common place to do such things. (At least while we are testing the program.) Then, to clear an area 15 characters wide by 4 lines deep, write:

```
     CALL 768,15,4
```

The command shown above uses simple constants, but ERASE can handle any quantities "width" and "depth" up to formulas as complex as those Applesoft can normally handle. (I can't brag about that part, since all the work is done by Applesoft's formula evaluation routine "FRMEVL", called indirectly in my program by the "JSR GETBYT".

In case you don't have John Crossley's article on Applesoft Internal Entry Points, GETBYT is a subsidiary routine that evaluates formulas, bringing back a single-byte integer in the X-register and in location $A1--"FACLO". I don't use "FACLO" in this routine. GETBYT gives an illegal quantity error if the formula evaluates to more than 255 or less than 0.)

If you specify a width or depth of zero, ERASE will give an illegal quantity error. If the width of the line goes past the right edge of the screen, the blanks will wrap around the screen on the next line down. ERASE will pick up at the correct horizontal/vertical location when clearing subsequent lines, however. If the area to be erased goes past the bottom of the screen, do not fear: ERASE wraps around to the top of the screen. Your program and variables will not be hurt.

Here is a short Applesoft program that demonstrates ERASE in action. The program first fills the entire screen with asterisks, and then clears three windows. The first window wraps around from the right edge of the screen to the left. The second wraps around from the bottom to the top. The third is in the middle of the screen. (I am assuming a 40-column screen here.)

```
100  FOR I = 1 TO 24: PRINT
     "****************************************"; : NEXT
110  HTAB 30: VTAB 10: CALL 768,20,5
120  HTAB 10: VTAB 20: CALL 768,20,8
130  HTAB 15: VTAB 10: CALL 768,10,5
```

And here is another demo, one which is closer to the way you will find yourself using ERASE. This one prints an array of six messages in six windows on the scrren, and lets you selectively erase them in any order one-by-one. As it turned out, the way I located the upper corners of the messages involved some lengthy formulas, but these ended up in the HTAB and VTAB statements. Note that I could have used data statements for similar results.

================================================================
DOCUMENT :AAL-8402:Articles:Delays.txt
================================================================

Delays, delays, delays.....................Bob Sander-Cederlof

We always want speed.  Making computers compute faster keeps our industry humming.  Yet nearly every major program has pieces of code called delays.

We use them to generate carefully controlled, timed events:  for example, generating a musical tone.  We use them to synchronize events, or to provide time for external events to occur.  We even use them just to slow the computer down so we can watch it work.

Delays are used so often that Woz had the foresight to put a general purpose delay subroutine permanently inside the monitor ROM.  It resides at $FCA8.  It is short (only 12 bytes), sweet (only uses one register, the same one which controls how long a delay you get), and slow (on purpose).  Here is a listing:

```
WAIT   SEC              PREPARE TO SUBTRACT
.1     PHA              SAVE A COPY OF A-REG
.2     SBC #1           COUNT A-REG DOWN TO ZERO
       BNE .2           ...UNTIL A=0
       PLA              GET SAVED COPY OF A-REG
       SBC #1           COUNT THIS COPY DOWN TOO
       BNE .1           ...UNTIL A=0
       RTS
```

To use this subroutine, you load the A-register with a value which will determine the length of the delay, and then JSR WAIT.  When the subroutine returns, A=0 and somewhere between 29 and 167309 clock cycles have elapsed.  The formula, somewhat confusingly printed on page 165 of the white Apple II Reference Manual (and elsewhere in other manuals), is:

```
       # cycles = (5*A*A + 27*A + 26)/2
```

For an example of its use, look in the monitor listing at $FBDD (the bell routine).  Examples of other timing loops are found in the tape cassette I/O routines ($FCC9-$FD0B) and the paddle reading subroutine ($FB1E).

Bill and I spent the last two weeks working with software which surrounds the Novation Cat Modem.  It is loaded with calls on the monitor WAIT subroutine.  It is exceedingly tiresome to crank out a formula like the quadratic above by hand, or even with a calculator, over and over and over, when you have several Apples sitting in the same room!

After four or five trips to the manual and the calculator, I decided to work out the times for all possible values of the A-register. Once and for all.

Here is a little Applesoft program which does the job, and elsewhere in this AAL you will find a full page showing all the cycle counts.

The A-register values are given in both hex and decimal. The delay count is given in thousands of cycles. Each cycle is close to one microsecond, so you could think of the counts as being in milliseconds.

The purists among you will want to multiply these cycle counts by the ACTUAL clock period (.9799268644 microseconds average, according to Sather) to get ACTUAL time.

RWTS in DOS or ProDOS also give lessons in the use of precise delays. You will find weird little pieces of code which make no sense whatever inside RWTS. Things like PHA followed immediately by PLA, followed by a NOP. These are usually just delaying tricks. A PHA-PLA pair takes exactly seven cycles, a NOP 2 more. There is a delay while waiting for the motor to come up to speed. Another while stepping the head from track to track.

These last two are intertwined, so that delays used while stepping across tracks count towards the total delay required to get the disk rotating at 300 rpm.

Don Lancaster in his Enhancing the Apple books makes good use of delays in synchronizing graphics generation with the CRT. By updating a picture in one graphics page while displaying another, and then switching pages, you can get pretty impressive animation. However, the page flipping operations sometimes splatter the display. Using delays just right, you can make the switching occur when it won't be noticed. You can even mix graphics into the middle of a text screen or vice versa, or mix hi-res and lo-res on the same screen.

Jim Sather in "Understanding the Apple II" also uses delays to control the screen switches in interesting ways. Jim figured out exactly how many cycles everything in the video generation circuitry takes. Using his programs you can even use hi-res to draw underlines on text screens! A horizontal scan takes exactly 65 clock cycles. A vertical scan takes exactly 17030 cycles. The following program, adapted from one given on page 3-16 of Sather's book, splits the screen between hi-res and lo-res. Tapping the space bar moves the boundaries of the split. Play with it!

=============================================================================
DOCUMENT :AAL-8402:Articles:Front.Page.txt
=============================================================================

$1.80

Volume 4 -- Issue 5                    February, 1984

In This Issue...



Yes, ProDOS is Now Being Shipped

We bought an Apple //e last weekend, and it came with system disks for both DOS 3.3 and ProDOS.  There was no DOS Reference Manual, although a little of DOS is mentioned in the Owner's manual.  There is a very nice ProDOS User's Manual, 150 pages of text and photos and drawings. The dealer says he still has no word on ProDOS as a separate product.

I Can't Believe He Typed The Whole Thing!

One of our readers took a few evenings and typed in the source code of the whole CX ROM from the Apple //e Reference Manual Addendum.  This is the code from $C100 through $CFFC, which is listed on pages 23-49. He added some of his own comments to the source, which more fully explain what is going on in there.  The source for the F8 ROM is on the disk too, but without many comments (pages 3-18 of the addendum). Naturally, the source files are in the format of the S-C Macro Assembler.

We think having the source of these ROMs on disk could enhance the //e in two ways:  you can make a larger size copy of the listings, so they can be read in normal room light; and you can experiment with improvements to the code.  If you have a PROM burner that will burn 2764s, I think you can even replace the chips.  If you'd like a copy, send us $15:  we'll mail the disk to you, and pass along a percentage to the energetic typist.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3

for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

=============================================================================
DOCUMENT :AAL-8402:Articles:FstScroll.IIe80.txt
=============================================================================

Fast Scroll for //e 80-column..............Bob Sander-Cederlof

The //e 80-column firmware scrolls in an annoying fashion. If you are trying to watch a listing go by, it looks like a bunch of kids on the playground, jumping up and down. And it is slower than almost any brand of 80-column card that plugs into slot 3.

The "slot 3" kind of 80-column card usually has a general purpose CRT controller chip on it. These chips use a wrap-around memory, and have one register that tells the chip where in memory to start the screen display. Scrolling is instantaneous, because it only involves writing a new address into two registers.

The //e 80-column card has no built-in features at all. All it is, is plain old RAM. A few extra circuits allow alternate columns to be taken first from the mother board and then from the 80-column card, back and forth. And the video rate is doubled, so 80 columns appear on each line. The scroll routine moves the whole screen up in two steps. First all the odd columns (in main memory) are moved up, and then all the even columns (in 80-column card memory). That is why you see the zig-zag effect.

The scroll is slower than a 40-column scroll by a factor of two. After all, it is essentially the same code, just called twice.

As I said in my article on fast scrolling in the September 1982 issue of AAL, you have to bear in mind that the authors of the programs in Apple ROM were not usually aiming for speed. They were trying to squeeze as much as possible into that tiny space, and make it as general as they could. The //e 80-column firmware supports windows smaller than a full screen, and that is seldom found in other types of 80-column cards.

On the other hand, since I am used to not having nice windows in the other cards, I can live with that in the //e. And I am having a hard time adjusting to that see-saw slow-motion scroller.

I re-wrote all the fast screen tricks from the September 1982 article to work in the //e with the Apple 80-column card. It scrolls as smooth as glass, but I still can't read it: now it's too fast!

==============================================================================
DOCUMENT :AAL-8402:Articles:Mac.Thoughts.txt
==============================================================================

So That's a Macintosh!..............................Bill Morgan

Well, now we know. The rumors were basically correct: 68000 processor, 128K RAM, 3.5 inch disk drive (but only one), portable, Lisa descendant, about $2500, and no expansion slots.

That last "feature" still has me a little shaken. I thought that if anybody knew better, it would be Apple, whose whole fortune is based on the expandability of the Apple ][. My first reaction was totally negative: who wants to bother with a dead-end machine? A total of 128K of RAM, and the screen memory occupies over 20K. Now that I've read a little more about the internals, and about the design objectives, things look a lot brighter. The on-board memory will be expandable to 512K when the 256K chips get more affordable.

System expansion will take place via the high-speed RS-422 serial ports. One of the designers pointed out that at 1 million bits per second (which can be reached with external clocking) you can transfer the entire memory image of the machine in one second. A couple of manufacturers (Davong and Tecmar) have already announced hard disks. Tecmar also announced an IEEE 488 interface. Macintosh designers also speak of "virtual slot" protocols for the serial ports, and "multi-drop (party line) capability".

There's another departure from usual Apple practice: no programming language is resident in the machine, or included in the purchase price! Several options will be available, including Pascal, Mac Basic, Microsoft Basic, Logo, and an Assembler/Debugger. The prices for the above packages will run in the $100-$150 range, not too bad. One article also mentioned C, about six months from now. It wasn't clear whether that was from Apple or an outside vendor. All of the above languages are scheduled for release in the next few months, except for Microsoft Basic. Russ Weaver, at Simtec/Quest, tells me he received that yesterday.

There is also 64K ROM (two 23256's) in the Mac, which holds the key to most programming. That ROM contains the code to support the "desk top" environment of mouse, icons, etc., the disk I/O, and the serial I/O. That is supposed to be 64K of the most tightly coded 68000 machine language around (as opposed to Lisa's compiled Pascal operating system code). I am told that there are over 400 entry points available to the programmer, with complete documentation coming soon from Apple for $250.

Several information sources have already popped up. If you haven't seen the February issue of Byte, go get it. There is a large section on Mac, including the best technical data so far. There are already two magazines specializing in Macintosh: Macworld, from the publishers of PC World, and ST.Mac, from Softalk. (Saint Mac? Come on.)  Macworld looks very good, especially for evaluation and demonstration of software.  I haven't seen a copy of ST.Mac yet, but Softalk is about the best of the "general" Apple magazines so I expect good things from their entry.  You can pay $2495 for a Macintosh serial number and get a year's free subscription to ST.Mac.

```
==============================================================================
DOCUMENT :AAL-8402:Articles:Message.Search.txt
==============================================================================
```

Listing Buried Messages....................Bob Sander-Cederlof

Do you like treasure hunts? Dis-assembling, analyzing, understanding, and modifying programs written in assembly language, with nothing to go by but the program in memory and maybe a user's manual ... to me it is a treasure hunt.

Last week I desperately need to make full use of a Novation Cat II Modem. "Full use" of almost any peripheral device implies the use of assembly language. Even though Novation includes a very nice manual for the purpose, it did not answer half my questions.

Novation also includes a disk with a program called Com-Ware II. This program is assembly language, and takes 74 sectors on the disk. Somewhere, hidden in a small, dark corner, guarded by gnomes, surrounded by wild beasts, lay the answers to all my questions.

I started by BLOADing the file. Then "CALL -151" to get into the monitor, and typed "AA60.AA73". The first two bytes diplayed the length of the file, and the last two bytes are the starting address. I learned it loaded at $900, and was $4825 bytes long.

I started using the monitor L command to scan through the program, and discovered that the programmer had placed all the screen messages "in line". That is, rather than putting all the screen text at the end of the whole program, or in the middle, or wherever, he coded the ASCII strings right in place. Each message was preceded by "JSR $3866", and ended with a $00 byte. The subroutine at $3866 retrieved the return address from the stack, used it to address the message text while printing it out, and then placed a new return address on the stack to continue execution right after the $00 byte.

This makes it difficult to use a program like Rak-Ware's wonderful DISASM, because you have to tell the boundaries of all non-executable code. And there seemed to be LOTS of messages.

On the other hand, it also makes it easier to follow the flow of the program. The buried messages are almost like living comments, telling me exactly what is going on in every section of code.

I decided to get my Apple to help. I wrote a "quick and dirty" program to scan through the whole image from $900 through $5125, looking for every occurrence of "JSR $3866". I printed out the address of the next byte, which is the first byte of message text. Then I searched for the terminating $00 byte, and printed out its address. Then I went back and printed out the message text.

After several tries, I even made my quick and dirty program nice and clean. I printed all the messages out, nicely formated for easy visual scanning. I set my printer on 8 lines/inch and 12 chars/inch to save paper, and let 'er rip. Six whole pages! I think a third of Com-Ware is taken up by messages!

Here is a sample of the printout. Notice that I printed control characters, including <RETURN>, as "^" followed by the printing form of the character. Thus "^M" means <RETURN>.

<<<sample printout here>>>

I believe a lot of programs of interest use a similar technique for message printing, and slight adaptation of my MESSAGE SEARCH program could help YOU find some buried treasure!

```
==============================================================================
DOCUMENT :AAL-8402:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler Version 1.0....................................$80.00
S-C Macro Assembler Version 1.1 Update.............................$12.50
Full Screen Editor for S-C Macro Assembler.......................  $49.00
    Includes complete source code.
S-C Cross Reference Utility........................................$20.00
S-C Cross Reference Utility with Complete Source Code..............$50.00
DISASM Dis-Assembler (RAK-Ware)....................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00

S-C Word Processor (the one we use!)...............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk......................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor....................$60.00

AAL Quarterly Disks...........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983

Double Precision Floating Point for Applesoft......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research)....................$79.00
Fontrix (Data Transforms)........................................  $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00
IACcalc Spreadsheet Program.........................(reg. $84.95)  $75.00
          The one we use every day.  It's better than Visicalc!

Blank Diskettes.................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                              100 or more:  25 cents each
   Cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  5 cents each
ZIF Game Socket Extender...........................................$20.00

Buffered Grappler+ Interface and 16K Buffer............($239.00)  $200.00
quikLoader EPROM Card..................................($179.50)  $179.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Understanding the Apple II", Sather.................($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
```

```
  "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
  "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
  "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
  "What's Where in the Apple", Second Edition..........($24.95)  $23.00
  "What's Where Guide" (updates first edition)..........($9.95)   $9.00
  "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
  "6502 Subroutines", Leventhal........................($17.95)  $17.00

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

      *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
      ***                  (214) 324-2050                  ***
      *** We accept Master Card, VISA and American Express ***
```

```
==============================================================================
DOCUMENT :AAL-8402:Articles:QR.Macros.txt
==============================================================================
```

Macro to Generate Quotient/Remainder Table for Hi-Res Work
.......................Bob Sander-Cederlof

A few months back an article in Byte magazine presented some fast hi-resolution plotting routines. One of the secrets to fast plotting is table lookup rather than computation of base addresses and offsets. The article included a 560 byte table for all the possible quotients and remainers you can get when dividing X by 7, where X is the horizontal coordinate (0 to 279).

The table of quotients and remainders makes it easy to get the byte position on a line (quotient) and the bit position in the byte (remainder) for a given dot X-coordinate.

Typing a 560 byte table into the computer is no fun, no matter how you do it. You might go into the monitor and type directly in hex, then later BSAVE the table. Or you might use an Applesoft program to build the table. I think the easiest way is to write a few short macros, and let the assembler do the work.

If you have Version 1.1 of the S-C Macro Assembler, the following code will do the trick. Version 1.0 cannot handle it, because the nesting level goes too deep. The listing it prints out gets quite long, due to all the macro expansion. Therefore I am just printing the source code here. The table it produces is also long, so I am just showing the beginning and end of it.

```
1000 *--------------------------------
1010 *   GENERATE QUOTIENT-REMAINDER
1020 *   TABLE FOR ALL POSSIBLE VALUES
1030 *   OF X/7, WHERE X=0...279
1040 *--------------------------------
1050        .MA DO.QS
1060 R      .SE 0
1070        >DO.RS
1080 Q      .SE Q+1
1090        .DO Q<40
1100        >DO.QS
1110        .FIN
1120        .EM
1130 *--------------------------------
1140        .MA DO.RS
1150        .DA #Q,#R
1160 R      .SE R+1
1170        .DO R<7
1180        >DO.RS
1190        .FIN
1200        .EM
1210 *--------------------------------
```

```
1220 Q      .SE 0
1230 QR     >DO.QS
1240 *--------------------------------
```

<<<show the beginning and end few lines of the hex dump of the table here>>>

=============================================================================
DOCUMENT :AAL-8402:Articles:QuikLoader.Card.txt
=============================================================================

The Amazing "quikLoader" Card..............Bob Sander-Cederlof

Jim Sather, author of "Understanding the Apple II", has designed and programmed a great new plug-in.  It is basically a ROM card, but hold on to your hats!

The card has sockets for 8 EPROMs, and they can be any EPROM size from 2716 up through 27256.  That means the card can hold a up to 256 kilobytes!

The card comes loaded already with three 2764 devices, programmed with licensed copies of DOS 3.3, FID, COPYA, the quikLoad operating system, and possibly more.  I think Integer BASIC is on there too.  With DOS on the card, you can leave it off your disks.  You gain at least two tracks per disk this way.

The quikLoad operating system allows you to load any program from the card into RAM in a flash.  If you have an EPROM programmer that can burn 2764s or larger, you can put favorites like the S-C Macro Assembler and our word processor permanently there.  The programs don't even have to be modified, because they will be loaded into their normal RAM locations for execution.

You control the card by typing a control character along with RESET. For example, ctrl-C RESET catalogs a disk; ctrl-H RESET runs "HELLO"; others boot a disk or enter the monitor.  Ctrl-Q RESET gives you a catalog of your quikLoader ROMs, in the form of a menu; a single keystroke then selects a program.

The board is compatible with Apple II, II Plus, and //e.  In a II Plus with a 16K RAM card, you may need to perform a slight modification to the RAM card as explained in the documentation.

The boards are being manufactured by Southern California Research Group (SCRG), P. O. Box 2231, Goleta, CA 93118.  Phone (805) 685-1931. Their price is $179.50.  You can order them from us if you like, at $170 + shipping.

```
==============================================================================
DOCUMENT :AAL-8402:Articles:Revisit.48.0.txt
==============================================================================
```

Revisiting $48:0...........................Bob Sander-Cederlof

Remember all those warnings about storing 0 in $48 after DOS had a whack at your zero page? Maybe not, but let me remind you.

Apple's monitor uses locations $45 through $49 in a very special way. Ignoring this, the writers of DOS also used them. When you start execution from the monitor (using the G, S, or T commands) The data in these locations gets loaded into the registers: $45 into A, $46 into X, $47 into Y, $48 into P (status), and $49 into S (stack pointer). When a program hits a BRK opcode, or the S command has finished executing a single opcode, the monitor saves these five registers back into $45...$49.

No serious problem, unless you like to enter the monitor and issue the G, S, or T commands. Even less of a problem, because the S and T commands were removed from the monitor ROM when the Apple II Plus came out. And if you don't care what is in the registers anyway....

But the P-register is rather special, too. One of its bits, called "D", controls how arithmetic is performed. If "D" is zero, arithmetic will be done in the normal binary way; if D=1, arithmetic is done in BCD mode. That is, adding one to $49 will produce $50 rather than $4A. If the program you are entering doesn't expect to be in decimal mode, and tries arithmetic, you will get some rather amusing results.

Hence the warning: before using the G command from the monitor, type 48:0 to be sure decimal mode is off. Later versions of DOS store 0 into $48 after calling those routines which use $48. And the monitor stores 0 into $48 whenever you hit the RESET key (or Control-RESET).

```
******************************************************
*                                                    *
*   Now I am here to tell you that storing 0 into    *
*   $48 is ALL WRONG!  It took Bill and me 5 hours   *
*   to unravel the mystery caused by storing zero    *
*   there!                                           *
*                                                    *
******************************************************
```

You should put into $48 a sensible value. Better, DOS should never use $45 through $48; if it must use them, save and restore them. There are eight bits in the P-register, and in the 6502 seven of them are important. One of them, we discovered, is VERY important.

The bit named "I" controls the IRQ interrupt. If I=1, IRQ interrupts will not be accepted. If I=0, IRQ interrupts will be accepted. So...who cares about interrupts?

Hardly anyone uses interrupts in Apple II's, because of all the hidden problems. But there are some very nice boards for the Apple that are designed to be used with interrupts. Most of them are safe, because RESET disables their interrupt generators.

Need I say that we discovered a board that does not disable the interrupt generators when you hit RESET? The Novation Cat Modem (a very excellent product) leaves at least one of its potential IRQ sources in an indeterminate state. IRQ's don't immediately show up, though, because they are trapped until you have addressed any of the soft switches on the card. But, for example, if that card is in slot 2 and I read or write any location from $C0A0 through $C0AF, IRQ's start coming. Still no problem, because I=1 in the P-register.

UNTIL WE USE THE MONITOR G COMMAND!

If I use the monitor G command, location $48, containing 0, is loaded into the P-register. Then an IRQ gets through and sends the 6502 vectoring through an unprepared vector at $3FE,3FF and BANG!

Our solution was to put SEI instructions in various routines, and to make sure that $48 contains 4, not 0, before using the G command.

From now on, whenever you hear that you need to be sure $48 contains zero, think four.

==============================================================================
DOCUMENT :AAL-8402:Articles:Short.Subjects.txt
==============================================================================

International Personal Robotics Conference

If you are among the many experimenting with little personal robots, such as Heathkit's HERO, you may be interested in attending the above named conference in Albuquerque next April 13-15. They are expecting around 4000 to show up from all over the world. You can meet such well known robotics experts as Joseph Engleberger, Nels Winless and others. It's a fair bet you'll find Jack Lewis of Micromation there. For more info, call Betty Bevers of IRPC at (303) 278-0662, or write to them at 1547 South Owens St. #46, Lakewood, Colorado 80226.

DOS 3.3 Checksummer Debate Update..........Bob Sander-Cederlof

A letter from Bill Basham (Diversi-DOS author) defending the practice of omitting the automatic VERIFY after SAVE to gain speed, was published in the September 1983 Softalk (page 37, 38). At the top of page 38 Bill claimed that the checksumming method used by DOS was of no value at all, because the checksum only depended on the last two bytes. In other words, Bill claims that errors in the first 340 bytes of a sector will not be caught.

Diversi-DOS is a fine product, and many thousands are enjoying its advantages. Nevertheless, Bill is wrong about the checksum. It does indeed catch errors throughout a sector. For a complete explanation, see the February 1984 Softalk. David Wagner clearly explains how the checksummer works, and refutes Bill's claim. See his letter on page 40.

You can look at the code, too. We printed a full commented source listing of this code in the June 1981 issue of AAL.

Peeking at the CATALOG.....................Bob Sander-Cederlof

Have you ever wanted just a quick peek at the catalog entry for a file? Maybe you want to know where the track/sector list is? Or maybe you want to see if there are any control characters in the name? Or if the number of sectors is more than 255? You need to peek, because CATALOG won't tell you these details.

After all these years, I found out a simple way to do it. That is, assuming you can OPEN, SAVE, LOCK, or otherwise somehow make DOS go looking for the file.

After DOS has found the file, it leaves the directory sector containing the filename in the buffer at $B4BB-B5BA. DOS also leaves an index to the very byte at which the information on your file is found. The value in $B39C, if added to the address $B4C6, gives you the address of the start of the entry. $22 bytes later it ends.

A minute ago I saved the contents of this and a few other short articles on a file named V4N5 SHORT SUBJECTS. Then I left my word processor, typed CALL -151 to get into the monitor, and... Well, here, look for yourself:

```
]CALL-151
*B39C
B39C- D2          (offset from B4C6)
*C6+D2
=98               (first byte of entry)
*98+22
=BA               (last byte of entry)
*B598.B5BA
B598- 0C 0E 00 D6 B4 CE B5 A0
B5A0- D3 C8 CF D2 D4 A0 D3 D5
B5A8- C2 CA C5 C3 D4 C3 A0 A0
B5B0- A0 A0 A0 A0 A0 A0 A0 A0
B5B8- A0 07 00
```

The first byte at B598 is the track, and the second is the sector, where the track/sector list for this file is stored. The third byte is the file type (00 means an unlocked text file). The last two bytes are the file size. All the bytes in between are the file name.

If you are interested in the entry for a file you cannot reach directly, perhaps because there are hidden characters in the name, just LOCK, UNLOCK, or whatever a file above or below it in the catalog. Then peek at B39C and B4BB...B5BA to find the entry you are really interested in.

Bill and I found this technique extremely useful on the most recent consulting job we handled.

We also took advantage of the fact that the track/sector list of a file read or written on can be found at the beginning of the file buffer. If there are three buffers (MAXFILES=3), and if the file in question was the only one being accessed at the time, the T/S list will be found at $9600...$96FF. You can get the data you need immediately, without even finding your favorite ZAP utility.

--------------------

Yes, ProDOS is Now Being Shipped

We bought an Apple //e last weekend, and it came with system disks for both DOS 3.3 and ProDOS. There was no DOS Reference Manual, although a little of DOS is mentioned in the Owner's manual. There is a very nice ProDOS User's Manual, 150 pages of text and photos and drawings.

I Can't Believe He Typed The Whole Thing!

One of our readers took a few evenings and typed in the source code of the whole CX ROM from the Apple //e Reference Manual Addendum. This is the code from $C100 through $CFFC, which is listed on pages 23-49.

He added some of his own comments to the source, which more fully explain what is going on in there. The source for the F8 ROM is on the disk too, but without many comments (pages 3-18 of the addendum). Naturally, the source files are in the format of the S-C Macro Assembler.

We think having the contents of these ROMs on disk could enhance the //e in two ways: you can make a larger size copy of the listings, so they can be read and studied in normal room light; and you can experiment with improvements to the code. If you have a PROM burner that will burn 2764s, I think you can even replace the chips....

If you'd like a copy, send us $15: we'll mail a copy of the disk to you, and pass along a percentage to the energetic typist.

================================================================================
DOCUMENT :AAL-8402:Articles:SoftswitchChart.txt
================================================================================

Table of //e Soft Switches.................Bob Sander-Cederlof

For some reason none of the //e manuals I own give a complete chart in one place of all the new soft switches.  If I print one here, I'll have one when I need it, so that's what the first chart on the following page is.

I have ordered them according to the location you peek at to find which position the soft switch is in.  The first column is the location you read.  The sense of the switch is given by bit 7 of the byte you read, and that bit's value is given at the top of the next two columns.

Note that there is an error in the Apple //e Reference Manual, on both pages 133 and 214, where the SLOTCXROM soft switch is described.  In both places, the slot/internal designations are backwards.  It looks like the book was written rationally, and the circuit behaves irrationally, because the SLOTC3ROM switch operates the opposite manner from the SLOTCXROM switch.  Oh well...

The maze of information regarding the bank switching switches has me baffled.  The second chart should help demystify things.  I show which switches to throw which way to make any particular range of memory come from the main 64K or the auxiliary bank.   To keep the chart from growing beyond the page, I did not include the LCBANK, SLOTCX, or SLOTC3 switches.

```
==========================================================================
DOCUMENT :AAL-8402:Articles:SWITCH.TABLES.txt
==========================================================================

Status        0             1
====================================
C011       C08(8-B)      C08(0-3)
LCBANK     Bank 1        Bank 2
  D000-DFFF
------------------------------------
C012       C081,2,9,A    C080,3,8,B
LCRAM      Select ROM    Select RAM
  D000-FFFF
------------------------------------
C013       C002          C003
RAMRD      Read Main     Read Aux
  200-BFFF
------------------------------------
C014       C004          C005
RAMWRT     Write Main    Write Aux
  200-BFFF
------------------------------------
C015       C006          C007
SLOTCX     Slot          Internal
  C100-C7FF
------------------------------------
C016       C008          C009
ALTZP      Main          Aux
  0-1FF, D000-FFFF
------------------------------------
C017       C00A          C00B
SLOTC3     Internal      Slot
  C300-C3FF, C800-CFFF
------------------------------------
C018       C000          C001
80STORE    RAMRD/RAMWRT  PAGE2
  400-7FF, 2000-3FFF
------------------------------------
C019
VBL        in display    in blanking
------------------------------------
C01A       C050          C051
TEXT       Graphics      Text
------------------------------------
C01B       C052          C053
MIXED      All Text or   Mixed text
           all graphics  & graphics
------------------------------------
C01C       C054          C055
PAGE2      Page 1/Main   Page 2/Aux
  400-7FF, 2000-3FFF
------------------------------------
C01D       C056          C057
```

```
HIRES       Lo-Res        Hi-Res
------------------------------------

Status        0             1
====================================
C01E        C00E          C00F
CHARSET     Normal        Alternate
------------------------------------
C01F        C00C          C00D
80COL       40 Columns    80 Columns
====================================

Address     Main Memory     Aux Memory
=======================================
D000-FFFF   C008 ALTZP=0    C009 ALTZP=1
            C080,3,8,B      C080,3,8,B
               LCRAM=1         LCRAM=1
---------------------------------------
C000-CFFF   I/O Space
---------------------------------------
4000-BFFF   Read: C002      Read: C003
           Write: C004     Write: C005
---------------------------------------
2000-3FFF   C001 80STORE=1  C001 80STORE=1
            C057 HIRES=1    C057 HIRES=1
            C054 PAGE2=0    C055 PAGE2=1

   or
            C000 80STORE=0  C000 80STORE=0
            Read: C002      Read: C003
           Write: C004     Write: C005

   or
            C056 HIRES=0    C056 HIRES=0
            Read: C002      Read: C003
           Write: C004     Write: C005
---------------------------------------
800-1FFF    Read: C002      Read: C003
           Write: C004     Write: C005
---------------------------------------
400-7FF     C001 80STORE=1  C001 80STORE=1
            C054 PAGE2=0    C055 PAGE2=1

   or
            C000 80STORE=0  C000 80STORE=0
            Read: C002      Read: C003
           Write: C004     Write: C005
---------------------------------------
200-3FF     Read: C002      Read: C003
           Write: C004     Write: C005
---------------------------------------
0-1FF       C008 ALTZP=0    C009 ALTZP=1
=======================================
```

===============================================================================
DOCUMENT :AAL-8402:Articles:TimeMaster.II.txt
===============================================================================

New Clock Card from Applied Engineering....Bob Sander-Cederlof

Dan Pote has a new improved clock/calendar card, the Timemaster II. The improvements include a new circuit design, all new firmware, a new manual, and new sample programs.

In one of the four switch-selectable modes, the Timemaster II is completely compatible with ProDOS. It emulates the clock protocol and formats of Thunderclock, Appleclock, and older versions of Dan's cards.

You still get a disk full of programs showing various ways to use the clock, including date-stamping of DOS 3.3 files, various digital and analog clock displays, and interrupt handling.

If you have been putting off the purchase of a clock card, wondering why, whether and which, now may be the time. The reason: ProDOS. The right card: if you want BSR control on the same card, Thunderclock; if you want the most clock for the best price, Timemaster II from Applied Engineering.

==============================================================================
DOCUMENT :AAL-8402:Articles:WrapAround.Addr.txt
==============================================================================

Reminder about Wrap-Around Addressing.............Bill Parker

Buried on the right side of page 65 of the November, 1983 issue of Call APPLE is the examination by Martin Smith of another quirk of the 6502. I say "another quirk" because it is similar to the JMP indirect wrap-around bug. Remember it?

As reported in the October 1980 issue of Apple Assembly Line, "JMP ($xxFF)" will not jump to the address pointed to by the two bytes beginning at $xxFF; rather the two bytes at $xxFF and $xx00 will be used. (Where xx means any page of memory.

A similar wrap-around situation can be found when indexing like this:

```
       STACK  .EQ $100
              LDX #1
              LDA STACK-1,X
```

Since STACK-1 is $FF, a page zero address mode is assembled. Indexing from within page zero never leaves page zero, so the above example references loacation $0000 rather than $0100.

The above is important, because many programmers use it in a "WHEREAMI" section of code to find the program's current address:

```
       STACK    .EQ $100
       WHEREAMI JSR $FF58   (KNOWN rts INSTRUCTION)
                TSX
                LDA STACK-1,X   GET PCL
                LDY STACK,X     GET PCH
```

For the Merlin Assembler, the problem can be corrected by forcing the assembler to use an absolute addressing mode rather than a page zero addressing mode. This is done by suffixing a ":" to the opcode, like this:

```
                LDA: STACK-1,X
```

The S-C Assemblers have no syntactical way to force absolute mode, but it can be done by defining the symbol STACK after its use. Here's an interesting example:

```
     0800- BD FF 00 1000        LDA STACK-1,X
     0100-          1010 STACK  .EQ $100
     0803- B5 FF    1020        LDA STACK-1,X
```

Since the assembler doesn't know the value of STACK in the first line, it has to assume it will be a two-byte address, and allocates that much space. By the time it gets to the last line it knows better.

The fact that indexing wraps around inside page zero is a plus sometimes. (I guess that explains why the chip works that way!) It has the effect of letting you use both positive and negative index offsets. Just beware of getting so used to negative offsets that you try to use them OUTSIDE page zero!

Clarification about our copyrights.........Bob Sander-Cederlof

We frequently are asked if it is all right to use ideas and even programs published in the Apple Assembly Line in articles or books our readers write for publication elsewhere, or even in software they plan to sell.

Sure! Just give us credit. Say where you got it, and hopefully tell your customers how they too can subscribe. The more you sell, the more we sell. The more we spread the good information around, the more we all benefit.

==============================================================================
DOCUMENT :AAL-8402:DOS3.3:DELAY.TIMES.txt
==============================================================================

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8402:DOS3.3:ERASE.DEMO.1.txt
==============================================================================

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8402:DOS3.3:ERASE.DEMO.2.txt
==============================================================================

( DTC removed -- lots of garbage characters )

==================================================================
DOCUMENT :AAL-8402:DOS3.3:S.Erase.Creamer.txt
==================================================================

```
 1000  *SAVE S.ERASE (JEFF CREAMER)
 1010  *--------------------------------
 1020  *                              *
 1030  *        ERASE ROUTINE         *
 1040  *                              *
 1050  *        Jeff Creamer          *
 1060  *                              *
 1070  *   CALL 768,(WIDTH),(DEPTH)   *
 1080  *                              *
 1090  *--------------------------------
 1100  *      PAGE ZERO VARIABLES
 1110  *--------------------------------
 1120  MON.CH     .EQ $24
 1130  MON.CV     .EQ $25
 1140  *--------------------------------
 1150  *      APPLESOFT ROUTINES USED
 1160  *--------------------------------
 1170  AS.CHKCOM  .EQ $DEBE
 1180  AS.GETBYT  .EQ $E6F8
 1190  AS.IQERR   .EQ $E199
 1200  *--------------------------------
 1210  *      MONITOR ROUTINES USED
 1220  *--------------------------------
 1230  MON.VTAB   .EQ $FC22
 1240  MON.PRBL2  .EQ $F94A
 1250  *--------------------------------
 1260         .OR $300
 1270         .TF ERASE
 1280  *--------------------------------
 1290  *      JEFF'S ERASE ROUTINE
 1300  *--------------------------------
 1310  ERASE  LDA MON.CV    GET VERTICAL COORD
 1320         PHA           SAVE ON STACK
 1330         LDA MON.CH    AND HORIZ COORD
 1340         PHA           SAVE IT ON STACK, TOO
 1350         JSR AS.CHKCOM    COMMA?
 1360         JSR AS.GETBYT    YES, GET WIDTH TO ERASE
 1370         TXA           INTO ACC
 1380         BEQ .4        WIDTH MUST BE NON-ZERO
 1390         PHA           PUSH WIDTH ON STACK
 1400         JSR AS.CHKCOM    COMMA NEXT?
 1410         JSR AS.GETBYT    YES, GET DEPTH
 1420         TXA           AND TRANSFER TO ACC
 1430         BEQ .4        DEPTH MUST BE NON-ZERO
 1440         TAY           DEPTH INTO Y REGISTER
 1450         PLA           WIDTH BACK OFF STACK
 1460         PHA           BUT KEEP IT THERE ALSO
 1470         TAX           AND INTO X-REG
 1480  .1     LDA MON.CV    REMEMBER CV ON STACK
```

```
           PHA
           JSR MON.PRBL2     PRINT WIDTH # OF BLANKS
           PLA             GET OLD CV OFF STACK
           DEY             DECREMENT DEPTH
           BEQ .3          ZERO LINES LEFT?
           TAX             OLD CV INTO X-REGISTER
           INX             NEXT LINE
           CPX #24         OFF THE BOTTOM?
           BCC .2          NO, USE THIS ONE
           LDX #0          YES, WRAP BACK TO TOP
     .2    STX MON.CV
           JSR MON.VTAB      ADJUST BASE ADDRESS
           PLA             WIDTH OFF STACK
           TAX             TO SET UP X AGAIN
           PLA             HORIZ COORD OFF STACK
           PHA             BUT MAINTAIN IT THERE ALSO
           STA MON.CH        AND RESTORE HCURSOR
           TXA             PUSH WIDTH BACK ON STACK
           PHA             FOR NEXT TIME AROUND
           BNE .1          LOOP ALWAYS
     .3    PLA             POP WIDTH OFF
           PLA             GET HORIZ COORDINATE
           STA MON.CH        AND RESTORE IT
           PLA             GET VERTICAL COOORDINATE
           STA MON.CV        RESTORE IT, TOO
           JSR MON.VTAB      ADJUST BASE ADDRESS
           RTS             DONE
     .4    JMP AS.IQERR  ILLEGAL QUANTITY ERROR
```

```
==============================================================================
DOCUMENT :AAL-8402:DOS3.3:S.Msg.Search.txt
==============================================================================

 1000  *SAVE S.MESSAGE SEARCH
 1010  *---------------------------------
 1020  *   FIND ALL MESSAGES IN COM-WARE II VERSION 5.0-3
 1030  *
 1040  *      ALL MESSAGES ARE PRECEDED BY "JSR $3866"
 1050  *      AND END WITH A $00 BYTE:
 1060  *
 1070  *      20 66 38 <MSG> 00
 1080  *---------------------------------
 1090  MSG.PNTR   .EQ $00,01
 1100  END.PNTR   .EQ $02,03
 1110  *---------------------------------
 1120  PRINTAX .EQ $F941
 1130  COUT    .EQ $FDED
 1140  CROUT   .EQ $FD8E
 1150  *---------------------------------
 1160  KEYBOARD   .EQ $C000
 1170  STROBE     .EQ $C010
 1180  *---------------------------------
 1190  FIND   LDA #$900    COMWARE WAS BLOADED AT $900
 1200         STA MSG.PNTR
 1210         LDA /$900
 1220         STA MSG.PNTR+1
 1230  .1     LDA MSG.PNTR
 1240         CMP #$5125   COMWARE ENDS AT $5125
 1250         LDA MSG.PNTR+1
 1260         SBC /$5125
 1270         BCC .2       ...NOT AT END YET
 1280         RTS          ...FINISHED
 1290  *---SEARCH FOR A $20 BYTE--------
 1300  .2     LDY #0
 1310         LDA (MSG.PNTR),Y
 1320         CMP #$20
 1330         BEQ .4       FOUND $20
 1340  .3     JSR INC
 1350         BNE .1       ...ALWAYS
 1360  *---CHECK FOR $66, $38 AFTER $20---
 1370  .4     INY
 1380         LDA (MSG.PNTR),Y
 1390         CMP #$66
 1400         BNE .3
 1410         INY
 1420         LDA (MSG.PNTR),Y
 1430         CMP #$38
 1440         BNE .3
 1450  *---FOUND A MESSAGE!-------------
 1460         LDX #10
 1470         JSR MARGIN
 1480         JSR PAUSE
```

```
1490        JSR INC      SKIP OVER THE $20, $66, $38
1500        JSR INC
1510        JSR INC
1520        LDA MSG.PNTR+1        PRINT STARTING ADDRESS
1530        STA END.PNTR+1
1540        LDX MSG.PNTR
1550        STX END.PNTR
1560        JSR PRINTAX
1570  *---SEARCH FOR END OF STRING-----
1580        LDY #0
1590  .5    LDA (END.PNTR),Y
1600        BEQ .6       FOUND END
1610        INC END.PNTR
1620        BNE .5
1630        INC END.PNTR+1
1640        BNE .5
1650  *---FOUND END OF STRING----------
1660  .6    LDA #"."     PRINT "..."
1670        JSR COUT
1680        JSR COUT     PRINT THE END ADDRESS
1690        JSR COUT
1700        LDA END.PNTR+1
1710        LDX END.PNTR
1720        JSR PRINTAX
1730        LDA #$A0     PRINT "   "
1740        JSR COUT
1750        JSR COUT
1760        JSR COUT
1770  *---PRINT OUT THE STRING---------
1780        LDY #0
1790        LDX #0
1800  .7    LDA (MSG.PNTR),Y
1810        BEQ .9       ...END OF STRING
1820        ORA #$80
1830        CMP #$A0     PRINTING CHARACTER
1840        BCS .8       ...YES, GO PRINT IT
1850        ORA #$40     ...NO, CONTROL, CHANGE TO
1860        PHA             PRINTING FORM
1870        LDA #"^"     PRINT "^" FOLLOWED BE CHAR
1880        INX
1890        JSR COUT
1900        PLA
1910  .8    JSR COUT
1920        INX
1930        JSR INC      ADVANCE MSG.PNTR
1940        CPX #55      IS THIS LINE FULL?
1950        BCC .7       ...NO, KEEP GOING
1960        LDX #24      ...YES, START NEW LINE
1970        JSR MARGIN   INDENT
1980        LDX #0
1990        BEQ .7       ...ALWAYS
2000  *--------------------------------
2010  .9    JSR CROUT
2020        JMP .1
```

```
2030  *--------------------------------
2040  INC    INC MSG.PNTR
2050         BNE .1
2060         INC MSG.PNTR+1
2070  .1     RTS
2080  *--------------------------------
2090  PAUSE  LDA KEYBOARD      ANY KEY PRESSED?
2100         BPL .3            ...NO, RETURN
2110         STA STROBE        ...YES, CLEAR STROBE
2120         CMP #$8D          WAS KEY <RETURN>?
2130         BNE .2            ...NO, JUST A PAUSE
2140  .1     JMP $3D0          ...YES, ABORT
2150  .2     LDA KEYBOARD      ANY KEY PRESSED?
2160         BPL .2            ...NO, KEEP WAITING
2170         STA STROBE        ...YES, CLEAR STROBE
2180         CMP #$8D          WAS KEY <RETURN>?
2190         BEQ .1            ...YES, ABORT
2200  .3     RTS               ...NO, END OF PAUSE
2210  *--------------------------------
2220  MARGIN JSR CROUT    START A NEW LINE
2230         LDA #$A0     SKIP OVER (X) SPACES
2240  .10    JSR COUT
2250         DEX
2260         BNE .10
2270         RTS
```

==============================================================================
DOCUMENT :AAL-8402:DOS3.3:S.ScrnTrIIe80.txt
==============================================================================

```
 1000  * S.SCREEN TRICKS //E 80-COLUMN
 1010  *--------------------------------
 1020  *   FAST SCREEN CLEAR SUBROUTINE
 1030  *--------------------------------
 1040  GCLEAR LDA #255
 1050         .HS 2C       SKIP OVER NEXT TWO BYTES
 1060  CLEAR  LDA #$A0
 1070  SET    LDY #119
 1080  .1     LDX #1
 1090  .2     STA $C054,X
 1100         STA $400,Y   LINES:  0  8 16
 1110         STA $500,Y           2 10 18
 1120         STA $600,Y           4 12 20
 1130         STA $700,Y           6 14 22
 1140         STA $480,Y           1  9 17
 1150         STA $580,Y           3 11 19
 1160         STA $680,Y           5 13 21
 1170         STA $780,Y           7 15 23
 1180         DEX
 1190         BPL .2
 1200         DEY
 1210         BPL .1
 1220         RTS
 1230  *--------------------------------
 1240  *      SET SCREEN TO ALL VALUES
 1250  *--------------------------------
 1260  SETALL LDA #0
 1270  .1     PHA
 1280         JSR SET
 1290         PLA
 1300         CLC
 1310         ADC #1
 1320         BNE .1
 1330         RTS
 1340  *--------------------------------
 1350  *   ALTERNATE SCREEN UNTIL KEY PRESSED
 1360  *--------------------------------
 1370  ALTER  LDA #$20     INVERSE BLANK
 1380         JSR SET
 1390         JSR CLEAR
 1400         LDA $C000
 1410         BPL ALTER
 1420         STA $C010
 1430         RTS
 1440  *--------------------------------
 1450  *   FAST SCROLL UP SUBROUTINE
 1460  *--------------------------------
 1470  SCROLL LDY #0
 1480  .1     LDX #1
```

```
1490  .3      LDA $C054,X
1500          LDA $400,Y   SAVE LINES: 0 8 16
1510          PHA
1520          LDA $480,Y   MOVE 1>0, 9>8, 17>16
1530          STA $400,Y
1540          LDA $500,Y   MOVE 2>1, 10>9, 18>17
1550          STA $480,Y
1560          LDA $580,Y   MOVE 3>2, 11>10, 19>18
1570          STA $500,Y
1580          LDA $600,Y   MOVE 4>3, 12>11, 20>19
1590          STA $580,Y
1600          LDA $680,Y        ET CETERA
1610          STA $600,Y
1620          LDA $700,Y
1630          STA $680,Y
1640          LDA $780,Y
1650          STA $700,Y
1660          PLA          MOVE 8>7, 16>15
1670          CPY #40
1680          BCC .2       DISCARD OLD LINE 0
1690          STA $780-40,Y
1700  .2      DEX
1710          BPL .3
1720          INY
1730          CPY #120
1740          BCC .1
1750          RTS
1760  *--------------------------------
1770  *   SCROLL AROUND, MOVING TOP LINE TO BOTTOM
1780  *--------------------------------
1790  SCR     LDY #39      SAVE TOP LINE ON STACK
1800  .1      LDA $C054
1810          LDA $400,Y
1820          PHA
1830          LDA $C055
1840          LDA $400,Y
1850          PHA
1860          DEY
1870          BPL .1
1880          JSR SCROLL   SCROLL SCREEN UP ONE LINE
1890          LDY #0       STORE OLD TOP LINE
1900  .2      LDA $C055
1910          PLA             ON BOTTOM OF SCREEN
1920          STA $7D0,Y
1930          LDA $C054
1940          PLA
1950          STA $7D0,Y
1960          INY
1970          CPY #40
1980          BCC .2
1990          RTS
2000  *--------------------------------
2010  *   ROTATE SCREEN UNTIL KEY PRESSED
2020  *--------------------------------
```

```
2030  S      JSR SCR      SCROLL AROUND ONCE
2040         LDA $C000    ANY KEY PRESSED?
2050         BPL S        NO, SCROLL AGAIN
2060         STA $C010    YES, CLEAR STROBE
2070         RTS          ...AND RETURN
```

```
=================================================================
DOCUMENT :AAL-8403:Articles:BragnerGPLEEtc..txt
=================================================================
```

S-C Macro and GPLE.LC on the //e...................Bob Bragner
Istanbul, Turkey

I've long been bothered by the way loading the S-C Macro Assembler wipes out GPLE (Neil Konzen's Global Program Line Editor) when you go from Applesoft to Assembler. It shouldn't happen, because GPLE resides in the alternate bank at $D000, not used by Macro. I've also been unhappy that the //e version of S-C Macro Assembler doesn't have the automatic line of dashes provided by <esc> L after a line number when you are in 80-column mode.

Just the other day I discovered by accident that all is not lost. If things are done in just the right sequence both of my peeves vanish.

First load up the S-C Macro Assembler into the $D000 area. Then enter Applesoft by typing the FP command, and BRUN GPLE.LC. Initialize the 80-column card with ctrl-V and enter the assembler by typing the INT command. This leaves GPLE connected so that the assembler sees the <esc> L command. Try it by typing a line number and <esc> L.

It also allows the assembler to see <esc> L to turn a catalog line into a LOAD command, but due to the way the word LOAD is poked onto the screen you get L O A D which clobbers the file name. (I never use the automatic load anyway, so this does not bother me.)

RESET will partially disable GPLE.LC, but you can restore it by typing the & command from Applesoft. If you want RESET to NOT molest GPLE, change the reset vector to $B6B3. You can do this from the monitor with "3F2:B3 B6 13", or from S-C Macro with "RST $B6B3".

I don't know why this all works, but I think it has something to do with the way the 80-column card initializes itself by copying the //e's monitor ROM into the $F800-FFFF space of RAM.

By the way, GPLE uses some patch space inside DOS 3.3 which is also used by the fast DOS text file I/O patch, so beware of mixing them.

==============================================================================
DOCUMENT :AAL-8403:Articles:Customizing68K.txt
==============================================================================

Changing Tab Stops in the 68000 Cross Assembler................
Bob Sander-Cederlof

The procedure as described in the S-C Macro Assembler manual works for the 6502 version and for all the cross assemblers except the 68000 cross assembler. The procedure described in Appendix D will not work because the 68000 cross assembler uses both banks of memory at $D000-DFFF. In order to be certain the correct one is switched on, the command interpreter keeps using the selection soft switches. The result is that the bank stays write-protected, and no patches ever get installed.

Of course, there is a simple way around the problem. Here is how to change the tab stops in the 68000 Cross Assembler:

First, boot the cross assembler disk and select option 2, loading the language card version at $D000.

```
:BLOAD S-C.ASM.MACRO.68000.LC
:MNTR
*AA60.AA61
*AA60- xx yy   (probably C6 27)
*D010.D014
D010- 0E 16 1B 20 00
*C083 C083 D010:7 10 1B 2B  (or whatever values you like)
C083- zz
C083- zz
*D010.D014
D010- 07 10 1B 2B 00
*BSAVE S-C.ASM.MACRO.68000.LC,A$D000,L$yyxx
*3D0G
:     that's it!
```

Similar methods apply to the other customizing patches mentioned in Appendix D.

==============================================================================
DOCUMENT :AAL-8403:Articles:Felt.Pads.txt
==============================================================================

About Disk Drive Pressure Pads.............Bob Sander-Cederlof

After you have used your disk drive for six months or so, it will probably develop a scary noise or two.  I know mine have.

My oldest drive is serial 1901 (the Shugart mechanics inside the box have a number somewhere in the low 400's).  Every once and a while it will make the most dangerous sounding noise you ever heard, something like dragging rusty chains across the road.  I have read in various magazines and newsletters that these noises are almost always caused by a dirty pressure pad.

The pressure pad rides on the top surface of the disk, pressing the disk surface down against the recording head.  It is a 1/8 inch circle of felt glued to a slightly larger plastic stud.  The shaft of the stud is split and tapered, so it will fit through a hole and lock in place.  You can easily remove the pad and stud by pressing on the split end.

But where do you get new ones?  Maybe at a computer store, but they sure don't keep them on display.  I decided to try a little home maintenance, and it worked.  I gently scraped the felt surface with the blade of my pocket knife, and all the old caked oxide turned to powder and fell off.  Then I rubbed the oxide on a piece of paper, to smooth out the felt.  After putting the drive all back together, it ran quietly.

It worked so well, I performed the operation on two more drives.  And surprisingly, one drive which had been giving lots of errors was working accurately again.

A few other disk maintenance tips:

One particularly noisy drive a few years ago had loose screws trying to hold the drive motor down.  A few wrist twists and all was well.

If a drive can read, but writes garbage, it is probably the 74LS125 on the analog board inside the drive.  Replace that chip for 25 cents or so, and you have saved $60 in repair bills.

================================================================
DOCUMENT :AAL-8403:Articles:Front.Page.txt
================================================================

$1.80

Volume 4 -- Issue 6 March, 1984

In This Issue...



For some time now we have been selling our S-C Word Processor, complete with all source code on disk. We hoped that some users would send us their improvements, and sure enough they have. Bob Gardner recently sent us a bunch, and that motivated me to go back over the package.

The disk now includes both II/IIPlus and //e versions. The //e version allows an 80-column preview (still only 40-column during edit mode). I added titles and page numbers, a single sheet mode, and more. Even with all the new features, the new object code is a little shorter than the old, leaving even more room for your own modifications and enhancements. I improved the internal documentation. The "manual-ette" is now 10 pages rather than 6. A small tutorial file helps you get started.

The price is still only $50. Owners of the old version can get a new copy for only $5.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
========================================================================
DOCUMENT :AAL-8403:Articles:Garbage.Collec.txt
========================================================================
```

Fast Garbage Collection................Col. Paul Shetler, MD
Honolulu, Hawaii

When Applesoft programs manipulate strings, memory gradually fills up with little bits and pieces of old strings. Eventually this space needs to be recovered so the program can continue. The process of searching through all the still active strings, moving them back to the top of free memory, and making the remaining space available again is called "garbage collection".

Applesoft will automatically collect garbage when memory fills up. However, the garbage collector in the Applesoft ROMs is pitifully slow. Worse yet, the time to collect is proportional to the square of the number of strings in use. That is, if you have 100 active strings it will take four times as long to collect garbage as if you had only 50 active strings.

Cornelis Bongers, of Erasmus University in Rotterdam, Netherlands, published a brilliant Garbage Collector for Applesoft strings in Micro, August 1982. The speed of his program, when compared to the one residing in ROM, is incredible. And the time is directly proportional to the number of strings, rather than the square of the number of strings. The only problem with his program is that it belongs to the magazine that published it. Or worse yet, it is tied to a program called Ampersoft, marketed by Microsparc (publishers of Nibble magazine) for $50. When I asked them about a license, they wanted big bucks.

So, I decided to write my own garbage collector, based on the ideas behind Cornelis Bongers' program. And then I further decided to make it available to all readers of Apple Assembly Line, where I myself have received so much help.

There are several catches. Normal Applesoft programs save all string data with the high-order bit of each byte zero (positive ASCII). Further, normal Applesoft programs never allow more than one string variable to point to the same exact memory copy of the string. The method of garbage collection my program uses (Bongers' method) DEPENDS on these constraints. If either is not true, LOOK OUT! Of course, if your Applesoft programs are normal, you need have no fear. Only if you are doing exotic things with your own machine language appendages to Applesoft might these constraints be violated.

The basic concept is fairly simple. Applesoft uses descriptors to point to the string in the string pool. The descriptor consists of three bytes -- the length, and the address of the characters in the string pool.

Strings build down from the top of memory (HIMEM) and the descriptors build up from the end of the program in the variable space. Since a new value assigned to a string is added to the bottom of the string pool, the pool is soon full of "garbage".

Applesoft frees the garbage one string at a time. This n-square method takes forever, when there are large string arrays. Bongers introduced the idea of marking active strings in the pool by setting the third byte in the string to a negative ASCII value, then storing the location of the descriptor in the first two bytes. The first two bytes of the string are saved safely in the address of field of the descriptor. The address previously in the address field will be changed anyway after all the strings are moved up in memory.

Another pass through the string pool moves all active strings as high in memory as they can go, retrieves the first two characters from storage in the descriptor, and points it to the new string location.

Since three bytes are used in the active strings, one and two character strings require different treatment. On the first pass through the variable space, the characters pointed to by the 'short' descriptors are stored in the length and, if len=2, the low address byte of the descriptor. The short descriptor is flagged with one or more "FF"'s, since no string can have an address greater than $FF00.

If short strings are found on the first pass, a third pass returns them to the string pool and points the descriptors to their new location.

Short strings do slow collection a little, however, the number of passes is proportional to the number of strings, and not the number squared.

Bongers' program was driven by calls via the &-statement. Mine differs in that it invoked with the USR function. Although it is easily converted to an ampersand routine, I wrote it using the USR function to provide fast garbage collection with Hayden's compiler (which also uses string descriptors and a string pool). The compiler allows USR functions, but makes & difficult. Another reason is to investigate some uses for USR.

USR(#) converts '#' to a floating point value in the FAC (floating point accumulator) and then jumps via $0A to the address pointed to in $0B, $0C. The results of the machine language subroutine can be returned in the FAC. The USR function, floating point calls, and addresses are described in Apple's BASIC REFERENCE MANUAL FOR APPLESOFT (Product #A2L0006).

The USR argument for my garbage collector requires a number in the range of +32767 to -32767. If the number is negative, the string pool is checked for negative ASCII. If any such characters are found, USR(-1) will return a value of 0, and no garbage collection will be attempted. If no negative ASCII characters are found, garbage collection will proceed. In this case USR(-1) returns the number of bytes of free space after collection.

If the USR argument is zero, for example K = USR(0), then collection is forced and USR will return the amount of free space. This is slightly faster than calling with USR(-1), because the preliminary scan for negative ASCII bytes is skipped. But USR(-1) is safer, if you are not sure.

If you use a positive argument N in the USR function, then no garbage collection will be performed unless there is less than 256*N bytes of free space left. Whether or not collection is performed, USR will tell you how much free space is left.

Only the lower five bits of the USR argument are tested. This means that USR(32) is the same as USR(0), USR(33) is the same as USR(1), and so on.

I have shown the program as residing at $9400, but of course you may re-assemble it for any favorite place.

The following Applesoft program makes a lot of garbage, and sees to the collection of it using my garbage collector. If the call to the USR function in line 245 left out, the program dies for 47 seconds while Applesoft does its own garbage collection. With the USR call as shown, the delay is less than one second.

<<<<sample here>>>>

<<<<collector listing here>>>>

==============================================================================
DOCUMENT :AAL-8403:Articles:Lancaster.SCWP.txt
==============================================================================

Lancaster's OBJ.APWRT ][ F..................Bob Sander-Cederlof

You may have noticed a little ad in the last few issues for an obscure title, "OBJ.APWRT ][ F". Don Lancaster, author of such favorites as Enhancing the Apple, Incredible Secret Money Machine, Micro Cookbook, etc., has torn into Applewriter //e. After a thorough analysis, he completely documented it, in the style of Beneath Apple DOS. The results, or at least part of them, will be chapter 12 in volume 2 of Enhancing.

He sent me a pre-print to look at and make comments about. My main comment is WOW! It doesn't matter if you like Applewriter or not. It doesn't even matter if you have never seen Applewriter. You still can learn a tremendous amount by reading through Don's text and comments. Of course it is better if you DO have Applewriter //e, because he tells you how to make some great customizing modifications.

You can get it all on disk for only $29.95. Actually, it is not on "disk" ... it is on SIX disk sides, jam-packed full. Don even throws in a free book for good measure.

You can order OBJ.APWRT ][ F directly from Synergetics, or from S-C Software.

Speaking of Word Processors................Bob Sander-Cederlof

For some time now we have been selling our S-C Word Processor, complete with all source code on disk. We hoped that some users would send us their improvements, and sure enough they have. Bob Gardner recently sent us a bunch, and that motivated me to go back over the package.

The disk now includes both II/IIPlus and //e versions. The //e version allows an 80-column preview (still only 40-column during edit mode). I added titles and page numbers, a single sheet mode, and more. Even with all the new features, the new object code is a little shorter than the old, leaving even more room for your own modifications and enhancements. I improved the internal documentation. The "manual-ette" is now 10 pages rather than 6. A small tutorial file helps you get started.

The price is still only $50. Owners of the old version can get a new copy for only $5.

```
==================================================================
DOCUMENT :AAL-8403:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler Version 1.0...................................$80.00
S-C Macro Assembler Version 1.1 Update............................$12.50
Full Screen Editor for S-C Macro Assembler........................$49.00
    Includes complete source code.
S-C Cross Reference Utility.......................................$20.00
S-C Cross Reference Utility with Complete Source Code.............$50.00
DISASM Dis-Assembler (RAK-Ware)...................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $45.00

S-C Word Processor (the one we use!)..............................$50.00
    With fully commented source code.
Applesoft Source Code on Disk.....................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor...................$60.00

AAL Quarterly Disks..........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984

Double Precision Floating Point for Applesoft.....................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research)...................$79.00
Fontrix (Data Transforms)..........................................  $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00
IACcalc Spreadsheet Program.........................(reg. $84.95)  $75.00
          The one we use every day.  It's better than Visicalc!

Blank Diskettes..................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                                100 or more:  25 cents each
   Cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  5 cents each
ZIF Game Socket Extender..........................................$20.00

Buffered Grappler+ Interface and 16K Buffer............($239.00)  $200.00
quikLoader EPROM Card..................................($179.50)  $170.00

Books, Books, Books..........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Understanding the Apple II", Sather.................($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
```

```
  "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
  "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
  "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
  "What's Where in the Apple", Second Edition..........($24.95)  $23.00
  "What's Where Guide" (updates first edition)..........($9.95)   $9.00
  "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
  "6502 Subroutines", Leventhal........................($17.95)  $17.00

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

      *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
      ***                  (214) 324-2050                  ***
      *** We accept Master Card, VISA and American Express ***
```

```
==============================================================================
DOCUMENT :AAL-8403:Articles:Putney.ClrPat.txt
==============================================================================
```

Rod's Color Pattern in 6502 Code.............Charles H. Putney

When I read the January AAL with Bob Urschel's article about running Rod's Color Pattern on the QWERTY 68000 board, it sounded like a challenge.  You may remember I like speed challenges, at least inside computers.

Fifty times faster than Basic didn't sound too fast, so I checked a simple loop to see if it might be possible to save the dignity of the 6502.  It did look possible, at least by using tricky table-driven code.

So, I wrote some more code and it looked like 8.0 seconds per loop. This clocks out at 55 times faster than Integer BASIC, but I didn't have the internal calculation for the color value exactly like the original.

I finally decided to use a table lookup for the color calculation. Now the problem was how to create all those data statements.  I thought about using some macros, but the calculations are too involved.  I wrote an Applesoft program to generate the lines of code for the assembler, and then EXECed them into my source code.  I finally got all the bugs out and timed it.

The table-driven version performs a main loop every 6.2 seconds, compared to 446 seconds per loop for the Integer BASIC version.  That is nearly 72 times faster.

Well, my only worry now is that Bob Urschel made an error in his timing, and his really runs 200 times faster.  If not, we have saved face for the venerable 6502.

Of course, we did use a little more memory.  But that is frequently a trade-off worth making in important programs.

For comparison purposes, here is the Integer BASIC program again:

```
10 GR
20 FOR W = 3 TO 50
30 FOR I = 1 TO 19
40 FOR J = 0 TO 19
50 K = I+J
60 COLOR = J*3 / (I+3) + I*W/12
70 PLOT I,K: PLOT K,I: PLOT 40-I,40-K:PLOT40-K,40-I
80 PLOT K,40-I: PLOT 40-I,K: PLOT I,40-K: PLOT 40-K,I
90 NEXT J: NEXT I: NEXT 2
100 GO TO 20
```

My program to generate the data tables includes similar logic. I broke the tables into two planes, rather than storing one data table 48*19*20 = 18,240 bytes long. One plane computes J*3/(I+3), and includes 380 bytes. The other computes I*W/12 and includes 48*19=912 bytes. My table lookup routine uses I and J to index into the first plane, and I and W into the second. Then the two values are added together. Pretty tricky!

I believe in letting computers work for me, so I had to use some macros to simplify typing in all the code for those eight plot statements. I wrote a PLOT macro, but then I noticed that there was some redundant code that way. By rearranging the order of the PLOT statements, I can separate the y-setup from the x-setup and plot. That way the base address does not get re-calculated as often, saving more time. Here is my program:

================================================================
DOCUMENT :AAL-8403:Articles:Redunancy.Table.txt
================================================================

Redundancy in Tables for Faster Lookups....Bob Sander-Cederlof

When speed is the main objective, you can sometimes use table lookups to great advantage. You trade program size for speed.

Here is an easy example. Suppose I want to convert the two nybbles of a byte to ASCII characters. I can do it all with code, like this:

```
CONVERT
       PHA              Save original byte
       LSR              Position first nybble
       LSR
       LSR
       LSR
       JSR MAKE.ASCII
       STA XXX
       PLA              Original byte
       AND #$0F         Isolate second nybble
       JSR MAKE.ASCII
       STA XXX+1
       RTS

MAKE.ASCII
       ORA #$B0         Make B0...BF
       CMP #$BA
       BCC .1           It is 0-9
       ADC #6           Make A-F codes
.1     RTS
```

That takes 30 bytes, and 75-77 cycles including a JSR CONVERT to call it. Actually 75 cycles if both nybbles are 0-9, 77 cycles if they both are A-F, and 76 cycles if there is one of each. If I move the code from MAKE.ASCII in-line, it saves 24 cycles (two JSRs, two RTSs), and only lengthens the program by one byte.

Or I can do a table lookup by substituting these two lines for both JSR MAKE.ASCII lines above:

```
       TAX
       LDA ASCII.TABLE,X
```

and making a little table like this:

```
ASCII.TABLE .AS -/0123456789ABCDEF/
```

In this form, the program takes 49 cycles, and uses a total of 39 bytes including the table. Perhaps it could be an advantage that the # of cycles is always constant, regardless of the value being converted.

You can make it even faster by using two whole pages for table space, like this:

```
CONVERT
       TAX
       LDA HI.TABLE,X
       STA XXX
       LDA LO.TABLE,X
       STA XXX+1
       RTS

HI.TABLE
       .AS -/0000000000000000/
       .AS -/1111111111111111/
       .
       .
       .AS -/FFFFFFFFFFFFFFFF/

LO.TABLE
       .AS -/0123456789ABCDEF/
       .AS -/0123456789ABCDEF/
       .
       .
       .AS -/0123456789ABCDEF/
```

The program itself is 14 bytes long, but there are 512 bytes of tables. The conversion, including JSR and RTS, now takes only 30 cycles. And since the program is now so short, it would probably get placed in line, saving the JSR-RTS, converting in only 18 cycles. And if the in-line routine already had the nybble in the X-reg, whack off another two cycles.

The redundancy in the tables gives a huge speed increase.

I have been tearing into the super fast copy utility that comes with Locksmith 5.0, and I discovered some of these redundancy tricks in their disk I/O tables. For example, the table for converting a six-bit value into a disk-code normally takes 64 bytes. The table looks like this:

```
TABLE  .HS 96979A9B9D9E9FA6
       .
       .
       .HS F7F9FAFBFCFDFEFF
```

Code to access the table might look like this:

```
       LDA BUFFER,X
       AND #$3F        Mask to 6 bits
       TAY
       LDA TABLE,Y
```

When you are writing to a disk, every single cycle counts. Therefore, it is pleasant to discover redundant tables. By making four copies of the table, using 256 bytes rather than 64, we no longer need to strip off the first two bits. The code can be shortened to this:

```
       LDY BUFFER,X
       LDA TABLE,Y
```

It only saves 3 cycles, but those three cycles can and do make the whole difference in the fast copy program. That is part of Locksmith's secret to reading a whole disk into RAM in only 8 seconds.

Speaking of Locksmith........................Warren R. Johnson

Did you know that Locksmith 5.0 can nearly be copied by plain old COPYA? Or with its own fast backup copier? All but the last few tracks copy, and they may not be necessary.

The only problem is, the resulting copy will not boot until you make a small patch using some sort of disk ZAP utility. (You can use Omega's Inspector/Watson team, Bag of Tricks, Disk Fixer, CIA, for example.) Patch Track-0F Sector-0E Byte-6F: change it from 6C to 0F. [ Editor's note: in my copy, Locksmith had C6 in that byte rather than 6C. And I have not tried the resulting disk to see if all functions work. ]

I have modified my Apple a little to make my life easier. I have 2732's in the motherboard ROM sockets, with bank switch selection. Applesoft is in one bank, and a modified version of Applesoft in the other. My modifications include replacing the old cassette commands (LOAD/SAVE/SHLOAD etc.) for an INWAT command. INWAT downloads the Inspector and Watson from some expansion chassis ROM boards.

1

==============================================================================
DOCUMENT :AAL-8403:Articles:Shorts.txt
==============================================================================

Will ProDOS Work on a Franklin?.....................Bob Stout

If you try to boot up ProDOS on a Franklin, it probably will fail. The ProDOS boot routine checks to see if you are in a genuine Apple monitor ROM. However, you can make it work.

Start the boot procedure; when meaningful action appears to have ceased, press the RESET switch. Get into the monitor and type 2647:EA EA and 2000G. Voila!

Will Rockwell 65C02's work in an old Apple..........Bob Stout

Not unless you have the 2 MHz part. For some reason the timing is too tight and slightly different to use a 1MHz 65C02, unless you have an Apple //e. The 2 MHz chips WILL work in Apple II and II Plus.

Will ProDOS Really Fly?....................Bob Sander-Cederlof

ProDOS appears to have been eclipsed by MacIntosh. The major software houses are probably putting their main effort into Mac.

ARTSCI has announced a ProDOS version of their MagiCalc spreadsheet program. Owners of the DOS 3.3 version may upgrade for $40, new customers pay $149.95. The only differences claimed are faster disk I/O and ability to edit the printer setup string. Nice, but $40 is a lot. And the spreadsheet files would no longer be accessible to DOS-based utilities.

ARTSCI will also send you their ProDOS catalog sorter program, complete with BASIC.SYSTEM, CONVERT, FILER, and the ProDOS image for only $24.95. Apple will reputedly be selling ProDOS with a user's manual and some tutorial files in addition to the files on ARTSCI's disk, but price and date are still unclear. (You get them free with a new disk drive.)

Practical Peripherals has announced a new clock card which is ProDOS compatible. Their design is apparently an upgrade of Superclock II (by Jeff Mazur, Westside Electronics). ProDOS was designed around Thunderclock, so other clocks must either emulate one of the Thunderclock modes or patch ProDOS during the boot process. Applied Engineering's new Timemaster II emulates Thunderclock and several others, so it is fully ProDOS compatible.

According to Don Lancaster, Applewriter //e has been written so that changing to ProDOS would be easy. Therefore we might expect a ProDOS-based version of this popular word processor to be announced soon. Or maybe they won't bother to announce it.

Meanwhile, I know of at least two people with plans to integrate the faster RWTS ProDOS uses into their enhanced DOS 3.3 packages.  Have you seen the latest ads for David-DOS?  Dave Weston compares the speeds of his fast DOS with DOS 3.3 and Pro-DOS.  Guess what ... Pro-DOS doesn't win.

Unless you MUST have file compatibility with Apple /// SOS; or you MUST have hard hard-disk support for very large files; or you MUST have a hierarchical file directory; then stick with DOS 3.3, enhanced by Dave, or Bill Basham, or Art Schumer, or others.  And if you MUST have at least 32K of program space with Applesoft; or you MUST have Integer BASIC support; or you MUST have compatibility with hundreds of existing software products; then stick with DOS 3.3.

================================================================================
DOCUMENT :AAL-8403:Articles:SILLY.SONGS.txt
================================================================================

The Baloney Song.......by Bob Sander-Cederlof

A piece of baloney got stuck in my nose,
I can't get it out with sniffles or blows.
It comes out a little, then back up it goes.
I reached in to get it, and pulled out my toes.
...but I got that baloney out of my nose!

A plate of spaghetti spilled over my beard,
and I can tell you it sure did look weird.
I hoped no one noticed, but just as I feared,
The rest of the restaurant stood up and cheered,
...and now you know why I shaved off my beard!

A big piece of bubble gum got in my hair.
I sure don't know how it got up there.
I blew a big bubble with the utmost care,
and then it exploded and flew everywhere.
...that must be how it got in my hair!

The Canoe Song........by Bob Sander-Cederlof

My canoe is blue all-around.
I'm a-gonna ride it up the river and down.
When I'm done I'm gonna put it away,
And there it's going to stay,
...until the next time that I ride it.

Momma's guitar is smaller than mine,
But she says she likes it just fine.
She's gonna play it all over town,
And then she's gonna put it down,
...until the next time that she plays it.

Trisha's pretty shoes are on the closet shelf,
She can put them on all by herself.
She's gonna wear them to the Sunday School today,
And then she's gonna put them away,
...until the next time that she wears them.

This is my song, I made it up one day.
Maybe you think I shoulda thrown it away.
By the look on your face I can tell it hurts your ear.
It's the worst you'll ever hear,
...until the next time that I sing it!

==============================================================================
DOCUMENT :AAL-8403:Articles:VerifyN2Display.txt
==============================================================================

Changing VERIFY to DISPLAY.................Bob Sander-Cederlof

In the July 1982 issue of AAL we showed how to make a text file display command inside DOS. Bob Bragner added 80-column display to it in the July 1983 issue. The Dec 1983 InCider printed an article by William G. Wright about a DOS modification that provided text file display without removing any previous features.

Wright's patches modify the VERIFY command so that as sectors are being verified, if the file is a text file, they are displayed on the screen or printer. If there are any $00 bytes in a sector, they are merely skipped over, so his patches will handle some random access files, as well as sequential. Non-text files are still verified in the normal manner.

I was prompted by his article to write ukp another little program. This one will hook into the VERIFY processor in the file manager when you BRUN the program. Later, 30BG from the monitor or CALL 779 from Applesoft will dis-install the patch. My patch modifies VERIFY so that as each sector of a file is verified it is displayed in hexadecimal on the screen or a printer. I do not distinguish between text and non-text files, although it would be a simple matter to do so. As with Wright's patches, random access files may also be displayed, up to the first hole in the track/sector list.

The creative among you will want to add many bells and whistles to my little program. Perhaps 80-column display, showing an entire sector at a time rather than half a sector. Perhaps display of the bytes in both hex and ASCII on the same line. Perhaps the ability to scan back and forth through a file, using the arrow keys. All these are possible, and not too difficult. Have fun!

```
==============================================================================
DOCUMENT :AAL-8403:DOS3.3:GARBAGE.TEST.txt
==============================================================================
```

```
d£37888:≤ $9400>n∫Á(4)"BLOAD B.FAST GARBAGE
COLLECTOR"Wxπ10,76:π11,0:π12,148q»N–25:ÜA$(N,N),B$(N,N)É“ÅI–1¡N:ÅJ –
1¡N£‹A$(I,J)–"X":B$(I,J)–"Y":Ç:ÇÆÊÅI–1¡N„F–
(,(112) 256»,(111))…(,(110) 256»,(109)):∫F" ";ÓıL–’(2)˘˙ÅJ–1¡N
     ÅK–1¡10:A$(I,J)–A$(I,J)»"*":Ç'      ∫"*";:Ç/        ∫:Ç
```

```
==============================================================================
DOCUMENT :AAL-8403:DOS3.3:PutneyTableMake.txt
==============================================================================

#d≤BUILD TABLES FOR ROD'S COLOR/nD$–Á(4)Px∫D$"OPEN TTT":∫D$"WRITE
TTT"YÇ∞200hå∫D$"CLOSE"nñÄz»ÅI–1¡19ô“∫"WI"Ë(‰(I)»" ",2)" .HS ";Æ‹ÅW –
3¡26:∞500:Ç:∫¿Ê∫"     .HS ";÷ÅW–27¡50:∞500:Ç:∫›˙ÇIÈ,ÅI–1¡19
     6∫"JI"Ë(‰(I)»" ",2)" .HS ";     @ÅJ–0¡19A      JC–”(J 3À(I»3)):C–
C…”(CÀ16) 16:≠Cœ9ƒC–C»7R  T∫"0"Á(C»48);Z       ^Ç:∫a      hÇIg
     r±ë  ÙC–”(I WÀ12):C–C…”(CÀ16) 16:≠Cœ9ƒC–C»7¢  ˛∫"0"Á(C»48);®
     ±
```

```
=======================================================================
DOCUMENT :AAL-8403:DOS3.3:QR.Table.Maker.txt
=======================================================================
```

```
 1000  *--------------------------------
 1010  *   GENERATE QUOTIENT-REMAINDER
 1020  *   TABLE FOR ALL POSSIBLE VALUES
 1030  *   OF X/7, WHERE X=0...255
 1040  *--------------------------------
 1050         .MA DO.QS
 1060  R      .SE 0
 1070         >DO.RS
 1080  Q      .SE Q+1
 1090         .DO Q<40
 1100         >DO.QS
 1110         .FIN
 1120         .EM
 1130  *--------------------------------
 1140         .MA DO.RS
 1150         .DA #Q,#R
 1160  R      .SE R+1
 1170         .DO R<7
 1180         >DO.RS
 1190         .FIN
 1200         .EM
 1210  *--------------------------------
 1220  Q      .SE 0
 1230         >DO.QS
 1240  *--------------------------------
```

```
=====================================================================
DOCUMENT :AAL-8403:DOS3.3:S.DISPLAY.FILE.txt
=====================================================================

1000  *SAVE S.DISPLAY FILE
1010  *--------------------------------
1020  *      PATCH DOS TO CHANGE VERIFY
1030  *      INTO DISPLAY
1040  *--------------------------------
1050  MON.PRNTAX .EQ $F941
1060  MON.PRBL2  .EQ $F94A
1070  MON.CROUT  .EQ $FD8E
1080  MON.PRBYTE .EQ $FDDA
1090  MON.COUT   .EQ $FDED
1100  *--------------------------------
1110         .OR $300
1120  *--------------------------------
1130  PATCH  LDA #DISPLAY      HOOK INTO VERIFY COMMAND
1140         STA $AD1C
1150         LDA /DISPLAY
1160         STA $AD1D
1170         RTS
1180  *--------------------------------
1190  UNPATCH
1200         LDA #$B0B6        RESTORE NORMAL VERIFY
1210         STA $AD1C
1220         LDA /$B0B6
1230         STA $AD1D
1240         RTS
1250  *--------------------------------
1260  DISPLAY
1270         JSR MON.CROUT     START SECTOR WITH <RET>
1280         JSR $B0B6         READ NEXT SECTOR
1290         BCS .1            END OF FILE
1300         LDY #0            DISPLAY FIRST HALF SECTOR
1310         JSR SHOW
1320         JSR SHOW          DISPLAY SECOND HALF
1330         CLC               SIGNAL NOT END OF FILE
1340  .1     RTS
1350  *--------------------------------
1360  SHOW   LDA $B5E5    DISPLAY SECTOR POSITION
1370         LDX $B5E4
1380         JSR MON.PRNTAX
1390         LDA #16      16 LINES PER BLOCK
1400         STA LCNT
1410         BNE .2       ...ALWAYS
1420  .1     LDX #4       PRINT 4 BLANKS
1430         JSR MON.PRBL2     SO COLUMNS LOOK NEATER
1440  .2     LDA #8       8 BYTES PER LINE
1450         STA BCNT
1460         TYA          PRINT BYTE COUNT
1470         JSR MON.PRBYTE
1480         LDA #"-"     PRINT "-"
```

```
1490          JSR MON.COUT
1500  .3      LDA #" "     PRINT " "
1510          JSR MON.COUT
1520          LDA ($42),Y  NEXT BYTE FROM FILE
1530          INY
1540          JSR MON.PRBYTE
1550          DEC BCNT
1560          BNE .3       MORE TO THIS LINE
1570          JSR MON.CROUT     NEXT LINE
1580          DEC LCNT
1590          BNE .1
1600          RTS
1610  *--------------------------------
1620  BCNT    .BS 1
1630  LCNT    .BS 1
1640  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8403:DOS3.3:S.FastGarbage.txt
==============================================================================

 1000  *SAVE S.FAST GARBAGE COLLECTOR
 1010  *-------------------------------
 1020  *   FAST GARBAGE COLLECTOR
 1030  *-------------------------------
 1040  *   BY COL. PAUL SHETLER, MD
 1050  *   INSPIRED BY CORNELIS BONGERS
 1060  *-------------------------------
 1070  *
 1080  *   CALL FROM APPLESOFT WITH K=USR(N)
 1090  *
 1100  *   IF N=0, THEN COLLECTION FORCED
 1110  *
 1120  *   IF N<0, THEN POOL CHECKED FOR NEG ASCII.
 1130  *           IF NO NEG ASCII, THEN GC FORCED
 1140  *           IF NEG ASCII FOUND, THEN
 1150  *                SET USER(#)=0 AND QUIT.
 1160  *
 1170  *   IF N>0, THEN COLLECTION PERFORMED ONLY IF
 1180  *          LESS THAN N*256 BYTES OF FREE
 1190  *          SPACE LEFT.
 1200  *-------------------------------
 1210  *      THE APPLESOFT PROGRAM MUST INLCUDE
 1220  *      THE FOLLOWING STATEMENTS TO SET UP
 1230  *      THIS GARBAGE COLLECTOR:
 1240  *
 1250  *      100 HIMEM:37888:REM$9400
 1260  *      110 PRINT CHR$(4)"BLOAD B.FAST GARBAGE COLL
 1270  *          ECTOR"
 1280  *      120 POKE 10,76 : POKE 11,0 : POKE 12,148
 1290  *-------------------------------
 1300  * EQUATES FOR GARBAGE COLLECTION
 1310  *-------------------------------
 1320  SHORT.FLAG          .EQ $06
 1330  STRING.LENGTH       .EQ $07
 1340  INDEX               .EQ $19
 1350  OFFSET              .EQ $1B
 1360  ARRAY.END           .EQ $1D
 1370  *-------------------------------
 1380  *      USER(#)  EQUATES
 1390  *-------------------------------
 1400  FACMO               .EQ $A0
 1410  FACLO               .EQ $A1
 1420  AYINT               .EQ $E10C
 1430  GIVAYF              .EQ $E2F2
 1440  *-------------------------------
 1450  *      STANDARD EQUATES
 1460  *-------------------------------
 1470  LOWTR               .EQ $9B
 1480  FORPNT              .EQ $08
```

```
STREND             .EQ $6D
VARTAB             .EQ $69
FRESPC             .EQ $71
FRETOP             .EQ $6F
MEMSIZE            .EQ $73
ARYTAB             .EQ $6B
*--------------------------------
       .OR $9400
       .TF B.FAST GARBAGE COLLECTOR
*--------------------------------
USR.GARBAGE.COLLECTOR
       JSR AYINT    CONVERT USR ARGUMENT TO INTEGER
*                   WITH HIBYTE IN FACMO, LO IN FACLO
       LDA FACMO    IS # MINUS?
       BMI .3       ...NEED TO CHECK FOR NEG ASCII
       LDA FACLO
       AND #$1F     8136 BYTES
       BEQ .4       ...IF =0 THEN FORCED COLLECTION
       CLC
       ADC STREND+1
       CMP FRETOP+1
       BCS .4       ...NEED TO COLLECT NOW
*---CALC FREE SPACE--------------
.1     SEC
       LDA FRETOP
       SBC STREND
       TAY          LO BYTE
       LDA FRETOP+1
       SBC STREND+1
*---FLOAT (AY) FOR USR RESULT----
.2     JMP GIVAYF   FLOAT (AY) AND RETURN
*---CHECK POOL FOR NEG ASCII-----
.3     JSR SET.STRING.POOL.STROLL
       JSR FIND.NEXT.NEG.BYTE.IN.POOL
       LDA #0       PREPARE ZERO IN CASE NEG ASCII
       TAY
       BCS .2       ...FOUND SOME NEG ASCII
*---COLLECT GARBAGE NOW----------
.4     JSR MARK.ALL.ACTIVE.STRINGS
       JSR RAISE.ALL.ACTIVE.STRINGS
*---FINAL CLEAN UP---------------
       LDA LOWTR    STORE NEW BOTTOM OF STRING POOL
       STA FRESPC
       LDA LOWTR+1
       STA FRESPC+1
       LDA SHORT.FLAG    NEED TO FIX SHORT STRINGS?
       BEQ .5            ...NO, NOT ANY SHORT ONES
       JSR FIX.SHORT.STRINGS
.5     LDA FRESPC   SET FRETOP TO TOP OF FREE SPACE
       STA FRETOP
       LDA FRESPC+1
       STA FRETOP+1
       JMP .1
*--------------------------------
```

```
*   MARK ACTIVE STRINGS WITH NEG BYTE
*--------------------------------
MARK.ALL.ACTIVE.STRINGS
       LDA #0       FLAG->NONE
       STA SHORT.FLAG
       JSR INITIATE.SEARCH
.1     JSR FIND.NEXT.STRING.VARIABLE
       BCS .5       ...NO MORE VARIABLES
       LDY #0       POINT AT LENGTH BYTE OF DESC.
       LDA (LOWTR),Y
       BEQ .1       STRING LEN =0
*---CHECK LOCATION OF STRING-----
       TAX          SAVE STRLEN IN X-REG
       INY          IF STRING DATA INSIDE PROGRAM,
       LDA (LOWTR),Y     THEN NO NEED TO FIDDLE
       STA FORPNT        WITH IT FURTHER.
       CMP VARTAB
       INY
       LDA (LOWTR),Y
       STA FORPNT+1
       SBC VARTAB+1      IN PROGRAM?
       BCC .1            ...YES, SO PASS
*---CHECK FOR SHORT STRING-------
       CPX #3       IF 1 OR 2, SPECIAL TREATMENT
       BCS .3       ...LONG STRING
*---SHORT STRING HANDLER---------
       STX SHORT.FLAG    NON-ZERO TO FLAG
       LDA #$FF
       STA (LOWTR),Y     MARKER IN 3RD DESC. BYTE
       DEY               POINT AT 2ND DESC. BYTE
       DEX               CHECK LENGTH
       BEQ .2            LEN=1, PUT $FF IN 2ND BYTE
       LDA (FORPNT),Y    LEN=2, SAVE CHAR IN 2ND BYTE
.2     STA (LOWTR),Y
       DEY               POINT AT 1ST BYTE OF DESC.
       LDA (FORPNT),Y    MOVE FIRST BYTE OF STRING
       STA (LOWTR),Y         TO DESC.
       BPL .1       ALWAYS
*---LONG STRING HANDLER----------
.3     LDA (FORPNT),Y    MARK FIRST BYTE OF STRING
       ORA #$80          MAKE NEG ASCII
.4     STA (FORPNT),Y
       DEY               BACK UP TOWARD BEG. OF DATA
       BMI .1            ...FINISHED MARKING THIS
       LDA (FORPNT),Y    SAVE STRING CHAR IN DESC.
       INY
       STA (LOWTR),Y     IN LAST 2 BYTES
       DEY          OF DESCRIPTOR
       LDA LOWTR,Y  SAVE ADDR INSIDE STRING
       BCS .4       ALWAYS SET
*---FINISHED MARKING STRINGS-----
.5     RTS
*--------------------------------
*   MOVE THE STRINGS AS HIGH AS POSSIBLE
```

```
2570  *--------------------------------
2580  RAISE.ALL.ACTIVE.STRINGS
2590         JSR SET.STRING.POOL.STROLL
2600         STX LOWTR+1  STARTS AT TOP
2610         STA LOWTR    OF STRNG POOL
2620  .1     JSR FIND.NEXT.NEG.BYTE.IN.POOL
2630  *--------------------------------
2640  * CARRY CLEAR ON RETURN WHEN THRU
2650  *--------------------------------
2660         BCC .4       ...NO MORE STRINGS IN POOL
2670         LDY #0
2680         AND #$7F
2690         STA (FRESPC),Y
2700  *--------------------------------
2710  * RESTORE STRING POOL TO POS ASC
2720  * THEN RESET POINTERS
2730  *--------------------------------
2740         SEC
2750         LDA FRESPC   RECOVER ADDR.
2760         SBC #2       OF DESCRIPTOR
2770         STA FRESPC   FROM THE STR
2780         BCS .2       ...NO BORROW
2790         DEC FRESPC+1
2800  .2     LDA (FRESPC),Y
2810         STA FORPNT   AND PUT IT
2820         INY          IN FORPNT
2830         LDA (FRESPC),Y
2840         STA FORPNT+1
2850         INY          Y=2
2860         LDA (FORPNT),Y
2870  *--------------------------------
2880  * RESTORE STRING BY RETURNING
2890  * THE FIRST TWO BYTES WHICH WERE
2900  * STORED IN THE DESCRIPTOR.
2910  *
2920  * THEN POINT DESCRIPTOR TO THE
2930  * NEW, CORRECT STRING POSITION.
2940  *--------------------------------
2950         DEY
2960         STA (FRESPC),Y
2970         LDA (FORPNT),Y
2980         DEY          Y=0
2990         STA (FRESPC),Y
3000         LDA (FORPNT),Y
3010         STA STRING.LENGTH
3020         SEC
3030         LDA LOWTR
3040  *--------------------------------
3050  * POINT LOWTR & STRING DESCRIPTOR
3060  * TO BOTTOM OF NEW STRING POOL.
3070  *
3080  * LOWTR HOLDS THE MOVING BOTTOM
3090  * OF THE STRING POOL.
3100  *--------------------------------
```

```
3110        SBC STRING.LENGTH
3120        STA LOWTR
3130        INY
3140        STA (FORPNT),Y
3150        LDA LOWTR+1
3160        SBC #0
3170        STA LOWTR+1
3180        INY
3190        STA (FORPNT),Y
3200 *---------------------------------
3210 * NOW MOVE THE STRING TO ITS
3220 * NEW ADDRESS.
3230 *---------------------------------
3240        LDY STRING.LENGTH
3250 .3     DEY
3260        LDA (FRESPC),Y
3270        STA (LOWTR),Y
3280        TYA
3290        BNE .3        ...NOT FINISHED YET
3300        BEQ .1        ...ALWAYS
3310 *---FINISHED MOVING STRINGS------
3320 .4     RTS
3330 *---------------------------------
3340 *   RESTORE NORMAL CONFIGURATION OF PNTR AND DATA
3350 *      FOR STRINGS OF 1 OR 2 CHARACTERS
3360 *
3370 *      SCAN THRU VARIABLE SPACE AGAIN:
3380 *          DESCRIPTORS OF STRINGS MARKED WITH $FF
3390 *          CONTAIN THE CHAR(S) TO RESTORE TO POOL.
3400 *
3410 *      FRESPC POINTS AT BOTTOM OF POOL
3420 *      LOWTR POINTS AT DESCRIPTORS
3430 *---------------------------------
3440 FIX.SHORT.STRINGS
3450        JSR INITIATE.SEARCH
3460 .1     JSR FIND.NEXT.STRING.VARIABLE
3470        BCS .5        ...FINISHED!
3480        LDY #2        POINT AT 3RD BYTE, 2ND OF ADDR
3490        STY STRING.LENGTH
3500        LDA (LOWTR),Y     IF 3RD BYTE =$FF, SHORTY.
3510        CMP #$FF      A SHORTY?
3520        BNE .1        ...NO, KEEP SCANNING VARIABLES
3530        DEY           ...YES, POINT AT 2ND BYTE
3540        LDA (LOWTR),Y     IF 2ND BYTE ALSO $FF,
3550        PHA               THEN LEN=1
3560        BPL .2        ...NOT $FF, ITS A STR CHAR
3570        DEC STRING.LENGTH
3580 .2     DEY           POINT AT 1ST BYTE OF DESCRIPTOR
3590        LDA (LOWTR),Y     GET 1ST ASC CHAR OF STRING
3600        PHA               SAVE ON STACK
3610 *---CALC PLACE IN POOL FOR DATA--
3620        SEC
3630        LDA FRESPC    REPOINT FRESPC
3640        SBC STRING.LENGTH
```

```
3650        STA FRESPC
3660        BCS .3
3670        DEC FRESPC+1
3680  *---RESTORE LENGTH TO DESC.-------
3690  .3     LDA STRING.LENGTH
3700        STA (LOWTR),Y
3710  *---STORE CHARS INTO POOL--------
3720  *--AND ADDR INTO DESCRIPTOR------
3730        PLA          FIRST CHAR
3740        STA (FRESPC),Y
3750        INY
3760        LDA FRESPC   LOBYTE OF ADDR
3770        STA (LOWTR),Y
3780        PLA          2ND CHAR
3790        BMI .4       ...IT IS $FF, ONLY 1 CHAR
3800        STA (FRESPC),Y
3810  .4     INY
3820        LDA FRESPC+1 HIBYTE OF ADDR
3830        STA (LOWTR),Y
3840        BNE .1       ALWAYS
3850  *---ALL FINISHED WITH SHORTIES---
3860  .5     RTS
3870  *--------------------------------
3880  *      STRING POOL STROLL
3890  *--------------------------------
3900  SET.STRING.POOL.STROLL
3910        LDX MEMSIZE+1 POINT FRESPC
3920        LDA MEMSIZE   AT HIMEM
3930        STA FRESPC    TO START
3940        STX FRESPC+1  STROLL.
3950        RTS
3960  *--------------------------------
3970  *   SEARCH STRING POOL FROM TOP TO BOTTOM
3980  *      FOR A NEGATIVE BYTE.
3990  *
4000  *      RETURN .CS. IF NEG BYTE FOUND,
4010  *             .CC. IF REACHED END OF POOL
4020  *--------------------------------
4030  FIND.NEXT.NEG.BYTE.IN.POOL
4040        LDX FRESPC+1
4050        LDY FRESPC
4060        LDA #0       PAGE AT A TIME
4070        STA FRESPC
4080        TYA          IS IT ZERO?
4090        BNE .2       NO!
4100  .1     DEX          YES
4110        CPX FRETOP+1 STILL IN POOL?
4120        BCC .5       ...NO
4130        STX FRESPC+1 DO NEXT PAGE
4140  .2     DEY
4150        BEQ .3
4160        LDA (FRESPC),Y
4170        BPL .2       POS ASCII
4180        BMI .4       NEG SO QUIT
```

```
4190  .3     LDA (FRESPC),Y
4200         BPL .1 NEW PAGE
4210  .4     CPX FRETOP+1
4220         BNE .5       FRESPC>FRETOP
4230         CPY FRETOP   FOR CARRY FLAG
4240  .5     STY FRESPC   FRESPC POINTS TO NEG ASC
4250         RTS
4260  *--------------------------------
4270  *   SET UP SEARCH OF VAR TABLE
4280  *--------------------------------
4290  INITIATE.SEARCH
4300         LDA VARTAB   START AT BEGINNING OF VARIABLES
4310         STA INDEX
4320         LDX VARTAB+1
4330         STX INDEX+1
4340         LDY #7       EACH VAR TAKES 7 BYTES
4350         STY OFFSET
4360         RTS
4370  *--------------------------------
4380  *   FIND NEXT STRING VARIABLE
4390  *--------------------------------
4400  FIND.NEXT.STRING.VARIABLE
4410  .1     LDX INDEX+1   SETUP SEARCH FOR NEXT STRING
4420         LDA INDEX
4430         LDY OFFSET
4440         CPY #7       STILL IN SIMPLE VARIABLES?
4450         BNE .4       ...NO, IN ARRAYS
4460         CPX ARYTAB+1 WE WERE, CHECK FURTHER...
4470         BCC .2       ...YES, STILL SIMPLE
4480         CMP ARYTAB
4490         BCS .3       ...NO
4500  .2     JSR IS.THIS.A.STRING.VARIABLE
4510         BCS .8       ...STRING FOUND
4520         JSR NXTEL    NOT A STRING
4530         BCC .1       ...ALWAYS, TRY AGAIN
4540  .3     LSR OFFSET   SET OFFSET TO 3 NOW
4550         STA ARRAY.END
4560         STX ARRAY.END+1
4570  .4     CPX ARRAY.END+1 INSIDE AN ARRAY?
4580         BCC .8       ...YES
4590         CMP ARRAY.END
4600         BCC .8
4610         CPX STREND+1 STILL IN VAR SPC?
4620         BCC .5       ...YES
4630         CMP STREND
4640         BCC .5       ...YES
4650         RTS          CARRY SET WHEN THRU VAR SPC
4660  *---SET UP A NEW ARRAY-----------
4670  .5     LDY #2
4680         CLC
4690         LDA (INDEX),Y
4700         ADC INDEX
4710         STA ARRAY.END   POINTER TO
4720         INY          NEXT ARRAY
```

```
4730        LDA (INDEX),Y
4740        ADC INDEX+1
4750        STA ARRAY.END+1
4760        JSR IS.THIS.A.STRING.VARIABLE IS THIS A STR?
4770        BCS .6        ...YES
4780        LDA ARRAY.END
4790        STA INDEX     NO
4800        LDX ARRAY.END+1
4810        STX INDEX+1
4820        BNE .4        ...ALWAYS
4830 *---FOUND STRING ARRAY-----------
4840 .6     LDY #4        POINT AT
4850        LDA (INDEX),Y     #DIMENSIONS OF ARRAY
4860        ASL           *2
4870        ADC #5
4880        ADC INDEX     POINT INDEX TO
4890        STA INDEX     FIRST ELEMENT
4900        BCC .7        OF NEW ARRAY
4910        INC INDEX+1
4920 .7     LDX INDEX+1
4930 *
4940 .8     STA LOWTR     LOWTR->STR DESCRIPTOR
4950        STX LOWTR+1
4960 *---NEXT VARIABLE----------------
4970 NXTEL  CLC
4980        LDA OFFSET    POINT INDEX TO
4990        ADC INDEX     NEXT VAR OR ELEMENT
5000        STA INDEX
5010        BCC .1
5020        INC INDEX+1
5030        CLC
5040 .1     RTS           STR FOUND,CARRY CLEAR
5050 *---------------------------------
5060 * SUBROUTINE STRING CHECK
5070 *---------------------------------
5080 IS.THIS.A.STRING.VARIABLE
5090        LDY #0
5100        CLC           INCASE NOT STR
5110        LDA (INDEX),Y
5120        BMI .2        ...NOT STRING
5130        INY
5140        LDA (INDEX),Y
5150        BPL .2        ...NOT STRING
5160        LDA #2        POINT PAST STR NAME
5170        ADC INDEX
5180        BCC .1        ...STRING
5190        INX           INDEX+1
5200 .1     SEC           CARRY SET IF STR FOUND
5210 .2     RTS
5220 *---------------------------------
```

```
===============================================================================
DOCUMENT :AAL-8403:DOS3.3:S.PutneysColor.txt
===============================================================================

 1000        .LIST MOFF
 1010  *SAVE S.PUTNEY'S COLOR
 1020        .OR $6000
 1030        .TF B.PUTNEY
 1040  *--------------------------------
 1050  *
 1060  *      FAST ROD'S COLOR PATTERN
 1070  *
 1080  *      CHARLES H. PUTNEY
 1090  *      18 QUINNS ROAD
 1100  *      SHANKILL
 1110  *      CO. DUBLIN
 1120  *      IRELAND
 1130  *
 1140  *--------------------------------
 1150  *
 1160  *      PAGE ZERO ADDRESSES
 1170  *
 1180  INVI   .EQ $EE      VARIABLE 40 - I
 1190  INVK   .EQ $EF      VARIABLE 40 - K
 1200  POINTR .EQ $F9      LORES PAGE POINTER (TWO BYTES)
 1210  I      .EQ $FB      VARIABLE I
 1220  J      .EQ $FC      VARIABLE J
 1230  K      .EQ $FD      VARIABLE K
 1240  W      .EQ $FE      VARIABLE W
 1250  COLOR1 .EQ $07      HALF OF COLOR FORMULA
 1260  *--------------------------------
 1270  COLOR  .EQ $08,09
 1280  COLEVN .EQ $08      EVEN ROW COLOR
 1290  COLODD .EQ $09      ODD ROW COLOR
 1300  MASK   .EQ $0A,0B
 1310  MSKODD .EQ $0A
 1320  MSKEVN .EQ $0B
 1330  *
 1340  *--------------------------------
 1350  *
 1360  *      ADDRESS TABLE
 1370  *
 1380  ODDMSK .EQ $F0      MASK FOR ELIMINATING UPPER BLOCK (LOWER
NIBBLE)
 1390  EVNMSK .EQ $0F      MASK FOR ELIMINATING LOWER BLOCK (UPPER
NIBBLE)
 1400  GRAPH  .EQ $FB40    ENABLE LO RES GRAPHICS
 1410  *--------------------------------
 1420  *   MACRO DEFINITIONS
 1430  *--------------------------------
 1440        .MA PLY      PLY ]1
 1450        LDY ]1       Y-COORD
 1460        LDA LORESL,Y
```

```
1470        STA POINTR
1480        LDA LORESH,Y
1490        STA POINTR+1
1500        TYA
1510        AND #1       GET LSB
1520        TAX
1530        .EM
1540 *--------------------------------
1550        .MA PLX      PLX ]1
1560        LDY ]1       X-COORD
1570        LDA (POINTR),Y GET THE PAGE BYTE
1580        AND MASK,X
1590        ORA COLOR,X
1600        STA (POINTR),Y PUT IT BACK
1610        .EM
1620 *--------------------------------
1630        .MA NEXT     NEXT VAR,LIMIT+1
1640        INC ]1       INCREMENT ]1 VARIABLE
1650        LDA ]1       TEST IF ]1=]2 YET
1660        CMP #]2
1670        BCS :1       YES, LEAVE LOOP
1680        JMP NEXT.]1  NO, KEEP LOOPING
1690 :1
1700        .EM
1710 *--------------------------------
1720        .MA GET      GET FORMULA,INDEX
1730        LDY I
1740        LDA ]1L-1,Y
1750        STA POINTR
1760        LDA ]1H-1,Y
1770        STA POINTR+1
1780        LDY ]2
1790        LDA (POINTR),Y
1800        .EM
1810 *--------------------------------
1820 *
1830 *
1840 *      MAIN LOOP
1850 *
1860 ROD    LDA $C056    SET LORES GRAPHICS ON
1870        LDA $C053    MIXED MODE
1880        JSR GRAPH
1890        LDA #ODDMSK
1900        STA MSKODD
1910        LDA #EVNMSK
1920        STA MSKEVN
1930 *--------------------------------
1940 BIG.LOOP
1950        JSR $FBE2    *** TESTING BEEP ***
1960        LDA #0       FOR W=3 TO 50 (0...47)
1970        STA W
1980 *---NEXT W COMES HERE------------
1990 NEXT.W LDA #1       FOR I=1 TO 19
2000        STA I
```

```
2010        LDA #39
2020        STA INVI
2030        LDA $C030    JUST FOR AUDIBLE FEEDBACK
2040  *---NEXT I COMES HERE------------
2050  NEXT.I LDA I        SET UP K = I+J
2060        STA K
2070        LDA INVI
2080        STA INVK
2090        >GET FORM1,W
2100        STA COLOR1   SAVE IT FOR INNER LOOP
2110        LDA #0       FOR J=0 TO 19
2120        STA J
2130  *---NEXT J COMES HERE------------
2140  NEXT.J >GET FORM2,J
2150        CLC          ADD THE FORMULAS
2160        ADC COLOR1   ACC = J*3/(I+3)+I*W/12
2170        AND #$0F     MASK OFF TOP
2180        STA COLEVN   EVEN COLOR
2190        ASL          SHIFT 4 BITS
2200        ASL
2210        ASL
2220        ASL
2230        STA COLODD   ODD COLOR
2240  *--------------------------------
2250        >PLY I
2260        >PLX K
2270        >PLX INVK
2280  *--------------------------------
2290        >PLY INVI
2300        >PLX K
2310        >PLX INVK
2320  *--------------------------------
2330        >PLY K
2340        >PLX I
2350        >PLX INVI
2360  *--------------------------------
2370        >PLY INVK
2380        >PLX I
2390        >PLX INVI
2400  *--------------------------------
2410        INC K
2420        DEC INVK
2430        >NEXT J,20
2440  *--------------------------------
2450        DEC INVI
2460        >NEXT I,20
2470  *--------------------------------
2480        >NEXT W,48
2490  *--------------------------------
2500        LDA $C000    ANY KEY PRESSED?
2510        BMI ROD4     YES
2520        JMP BIG.LOOP   NO, KEEP LOOPING
2530  ROD4   STA $C010
2540        RTS
```

```
2545         .LIST OFF
2550  *--------------------------------
2560  *
2570  *      LORES GRAPHICS PAGE
2580  *      BASE ADDRESSES (40 COL)
2590  *
2600  LORESL .HS 0000808000008080
2610         .HS 0000808000008080
2620         .HS 2828A8A82828A8A8
2630         .HS 2828A8A82828A8A8
2640         .HS 5050D0D05050D0D0
2650  LORESH .HS 0404040405050505
2660         .HS 0606060607070707
2670         .HS 0404040405050505
2680         .HS 0606060607070707
2690         .HS 0404040405050505
2700  *
2710  *--------------------------------
2720  *
2730  *      TABLE FOR I*W/12
2740  *
2750  FORM1L .DA #WI1,#WI2,#WI3,#WI4,#WI5,#WI6,#WI7
2760         .DA #WI8,#WI9,#WI10,#WI11,#WI12,#WI13,#WI14
2770         .DA #WI15,#WI16,#WI17,#WI18,#WI19
2780  FORM1H .DA /WI1,/WI2,/WI3,/WI4,/WI5,/WI6,/WI7
2790         .DA /WI8,/WI9,/WI10,/WI11,/WI12,/WI13,/WI14
2800         .DA /WI15,/WI16,/WI17,/WI18,/WI19
2810  *--------------------------------
2820  WI1  .HS 000000000000000000010101010101010101010101020202
2830       .HS 020202020202020202030303030303030303030303040404
2840  WI2  .HS 000000010101010101020202020202030303030303040404
2850       .HS 040404050505050505060606060606070707070707080808
2860  WI3  .HS 000101010102020202030303030404040405050505060606
2870       .HS 0607070707080808080909090A0A0A0A0B0B0B0B0C0C0C
2880  WI4  .HS 010101020202030303040404050505060606070707080808
2890       .HS 0909090A0A0A0B0B0B0C0C0C0D0D0D0E0E0E0F0F0F000000
2900  WI5  .HS 01010202020303040405050506060707070808090 90A0A0A
2910       .HS 0B0B0C0C0C0D0D0E0E0F0F0F000001010102020303040404
2920  WI6  .HS 0102020303040405050606070708080909 0A0A0B0B0C0C0D
2930       .HS 0D0E0E0F0F000001010202030304040505060607070808 09
2940  WI7  .HS 01020203040405050607070808090 90A0B0B0C0C0D0E0E0F
2950       .HS 0F0000010202030304050506060707080909 0A0A0B0C0C0D
2960  WI8  .HS 02020304040506060708080 90A0A0B0C0C0D0E0E0F000001
2970       .HS 020203040405060607080809 0A0A0B0C0C0D0E0E0F000001
2980  WI9  .HS 020303040506060708090 90A0B0C0C0D0E0F0F0001020203
2990       .HS 04050506070808090A0B0B0C0D0E0E0F0001010203040405
3000  WI10 .HS 0203040505060708090A0A0B0C0D0E0F0F000102030404 05
3010       .HS 0607080909 0A0B0C0D0E0E0F00010203030405060708 0809
3020  WI11 .HS 02030405060708090A0B0B0C0D0E0F000102030405060607
3030       .HS 08090A0B0C0D0E0F000101020304050607 08090A0B0C0C0D
3040  WI12 .HS 030405060708090A0B0C0D0E0F000102030405060708090A
3050       .HS 0B0C0D0E0F000102030405060708090A0B0C0D0E0F000102
3060  WI13 .HS 030405060708090A0B0D0E0F000102030405060708 0A0B0C
3070       .HS 0D0E0F000102030405070809 0A0B0C0D0E0F000102040506
```

```
3080  WI14 .HS 0304050708090A0B0C0E0F000102030506070809 0A0C0D0E
```

=================================================================
DOCUMENT :AAL-8403:DOS3.3:SATHER.3.16.txt
=================================================================

```
 1000  *SAVE SATHER 3-16
 1010  *--------------------------------
 1020  *      HIRES-LORES SPLIT
 1030  *      SATHER 3-16
 1040  *--------------------------------
 1050  KYBD     .EQ $C000
 1060  STRB     .EQ $C010
 1070  GRAPHICS .EQ $C050
 1080  TEXT     .EQ $C051
 1090  NOTMIXED .EQ $C052
 1100  PAGE1    .EQ $C054
 1110  LORES    .EQ $C056
 1120  *--------------------------------
 1130  *      TOGGLE HI/LO-RES EVERY 8515 CYCLES
 1140  *--------------------------------
 1150         .OR $300
 1160  SPLIT  LDY PAGE1     HI/LO PAGE 1
 1170         LDY NOTMIXED
 1180         LDY GRAPHICS
 1190  *--------------------------------
 1200  SLEW   LDY #39      (2)    SLEW SCREEN IF KEY PRESSED
 1210         JSR WAITX10  (390)  6*65+7 CYCLES
 1220         LDY STRB     (4)
 1230  *--------------------------------
 1240  KEYCHK LDY KYBD     (4)    ANY KEY PRESSED?
 1250         BMI SLEW     (2 OR 3)  YES, SLEW ONE LINE
 1260         ADC #1       (2)    MAKE ALTERNATING 0 AND 1
 1270         AND #1       (2)
 1280         TAX          (2)    REMEMBER, 0 OR 1
 1290         LDY LORES,X  (4)    LORES IF X=0, HIRES IF X=1
 1300         LDX #8       (2)
 1310         JSR WAITX1K  (8000)
 1320         LDY #49      (2)
 1330         JSR WAITX10  (490)
 1340         CLC          (2)
 1350         BCC KEYCHK   (3)    ...ALWAYS
 1360  *                   ======
 1370  *                   (8515)
 1380  *
 1390  *--------------------------------
 1400  *      TIMING ROUTINES
 1410  *--------------------------------
 1420  *
 1430  *---WAIT 10Y CYCLES-------------
 1440  *---(INCLUDING JSR)-------------
 1450  WAITX10  DEY        (2)  WAIT Y-REG TIMES 10
 1460  .1       DEY        (2)
 1470           NOP        (2)
 1480           BNE .2     (3 OR 2)
```

```
1490           RTS        (6)
1500  .2       BNE .1     (3)  ...ALWAYS
1510  *--------------------------------
1520  *
1530  *---WAIT 1000X CYCLES------------
1540  *---(INCLUDING JSR)--------------
1550  LOOP1K   PHA        (3)
1560           PLA        (4)
1570           NOP        (2)
1580           NOP        (2)
1590  WAITX1K  LDY #98    (2)  WAIT X-REG TIMES 1000
1600           JSR WAITX10 (980)
1610           NOP        (2)
1620           DEX        (2)
1630           BNE LOOP1K (3 OR 2)
1640           RTS        (6)
1650  *--------------------------------
1660  *--------------------------------
1670  *      HORIZONTAL SPLIT
1680  *      BY BOB SANDER-CEDERLOF
1690  *--------------------------------
1700  HSPLIT
1710         LDA GRAPHICS (4    4)
1720         LDA KYBD     (4    4)    SEE IF SHOULD SLEW
1730         BPL .1       (3    2)
1740         STA STRB          (4)
1750         BMI .3            (3)
1760  .1     NOP          (2)
1770         BPL .3       (3)
1780  .3     JSR DLY12    (12  12)
1790         NOP          (2    2)
1800         NOP          (2    2)
1810  *                   ---  ---
1820  *                   (32  33)
1830  *
1840         LDA TEXT     (4)
1850         JSR DLY21    (21)
1860         CLC          (2)
1870         BCC .2       (3)
1880  .2     BCC HSPLIT   (3)
1890  *                   ----
1900  *                   (33)
1910  *
1920  *--------------------------------
1930  *      JSR DLY..    (6)     (6)
1940  DLY21  PHA          (3)
1950         PLA          (4)
1960         NOP          (2)
1970  DLY12  RTS          (6)     (6)
1980  *                   ----   ----
1990  *                   (21)   (12)
```

```
===============================================================================
DOCUMENT :AAL-8404:Articles:BurnErase.EPROM.txt
===============================================================================
```

Burning and Erasing EPROMs.................Bob Sander-Cederlof

We get a lot of questions about EPROM burners and erasers. Perhaps this list will help...

Burners_______

PROM Blaster System, $119, Apparat, 4401 South Tamarac Parkway, Denver, CO 80237. Phone (303) 741-1778 or (800) 525-7674. Will burn most 24-pin EPROMS. Price includes personality modules for 2704, 2708, 2508, 2758, 2716(TI), 2516, 2716, 2532, 2732, 2732A, 68764, 2815, and 2816. ZIF socket for EPROM. No power switch, so you must power down the Apple whenever you insert or remove an EPROM.

Apple-PROM, $149.95, Computer Technology Associates, 1704 Moon N.E., Suite 14, Albuquerque, NM 87112. Phone (505)298-0942. Will burn most 24-pin EPROMS. DIP switch selection for 2708, 2716, 2516, 2532, 2732, 2732A, 2764, 2564. Low insertion force socket for EPROM.

Romwriter, $175, Mountain Computer....(I cannot find any recent ads, but they are still listed in distributor catalogs). We have heard that they are no longer manufacturing this card, but there are still many available. Only burns 2716 (single voltage version, not TI). ZIF for EPROM. Power switch on card allows you to safely insert and remove EPROMs without turning off your Apple. I have been using this one for several years with no problems, although I did rewrite the software to suit my own tastes and needs.

Quick EPROM Writer, $149, available from Handwell Corp., 4962 El Camino Real, Suite 119, Los Altos, CA 94022. Phone (415) 962-9265. Made in Taiwan by "COPAM". Burns both 24- and 28- pin EPROMs. All software is in firmware on the card. Nice menu select for 2716, 2516, 2532, 2732, 2732A, 2564, 2764, and 27128. No personality modules or switch selection required, as all configuration is software controlled. Power is applied to and removed from the ZIF socket under software control, so that EPROMs can be inserted and removed without turning off your Apple. Manual includes schematic, pinout diagrams for EPROMs, and a (sparsely) commented listing of firmware. The firmware apparently implements an intelligent burning algorithm, which burns twice as long as it takes to get the byte burned, rather than using a fixed delay for each byte. The result is much faster burn times than most other burners listed here.

HM3264, $395, Hollister Microsystems, 5081 Fairview, Hollister, CA 95023. Phone (408) 637-0753. Programs 2716, 2732, 2732A, 2764, and 27128. Henry Spragens uses this one, and says it is very well designed and built, though expensive. Henry has modified the software Hollister provides to use the intelligent burn algorithm (it was pretty slow until he did this). Hollister use the C800-CFFF address space, like Mountain Computer does, as a 2048-byte window into the EPROM. Bank switching on the card lets you program larger EPROMS. Power switch on card allows you to safely insert and remove chips. A program switch helps prevent inadvertent programming.

Model EP-2A-79, $169 plus $17 to $35 each for personality modules and $19 to $40 for software. Optimal Technology, Earlysville, VA 22936. Phone (804) 973-5482. Programs full range from 2708 through 27128, plus 38E70 and 8751 MPUs, assuming you purchase the corresponding personality modules and software. It is not clear to me whether this plugs directly into an Apple or requires a separate serial interface card.

Erasers_______

QUV-T8 EPROM Erasers, Logical Devices, 1321E N.W. 65 Place, Fort Lauderdale, FL 33309. Phone (305) 974-0967 or (800) EE1-PROM (that is 331-7766). Four models, ranging from $49.95 to $124.95. I use and recommend the $97.50 model, which includes a slide-out tray, anti-static foam pad, UV indicator lens, timer, and safety interlock switch.

Spectronics, marketed by JDR Microdevices, 1224 S. Bascom Avenue, San Jose, CA 95128. Phone (800) 662-6279 or (408) 995-5430. Six models from $83 to $595. The $83 unit has no timer, all the others do. [ JDR's latest ad in Byte shows eight 250nsec 4116's for $7.95! ]

Jade Computer Products carries both brands of EPROM Erasers. Their price on the least expensive Spectronics is only $69.95.

Jameco Electronics lists an eraser for $79.95.

More Clocks for Apple......................Bob Sander-Cederlof

Some more clock cards have been brought to my attention recently.

Prometheus Versacard includes a clock, and it is compatible with ProDOS due to its ability to emulate a Thunderclock. List price is $199.

Naturally, there is a clock on the Mountain Computer CPS/Multifunction Card. Naturally, because "CPS" stands for Clock Parallel Serial, the three functions the card supports. I cannot find a current price for this card.

Practical Peripherals is advertising ProClock, no price mentioned.

==============================================================================
DOCUMENT :AAL-8404:Articles:CRC.txt
==============================================================================

Cyclic Redundancy Check Subroutine.........Bob Sander-Cederlof

In the May 1983 AAL I wrote about checksums and parity, two ways to guarantee the integrity of data. In the world of microprocessors, you may encounter checksums at the end of data records on tape or disk, and parity bits on characters sent via a modem between computers. Tacking on parity bits and checksums to data helps in checking whether the data has been changed. However, there are more secure methods.

The best method I have ever heard of is commonly called Cyclic Redundancy Check, or CRC for short. Since it appears a lot more complicated than parity or checksum, it stands to reason it should have a more complex name. Right? But programmers have a way of reducing all complexity to three capital letters, so we will call it CRC.

First, a little review. The kind of parity I most frequently see adds an 8th bit on the left of a 7-bit value. The parity bit is chosen so that the total number of one-bits in the 8-bit byte is odd. For example, the seven bit number 1011010 (which might stand for an ASCII "Z") becomes 11011010, or $DA. If we run into the value 01011010 ($5A), we know there has been an error somewhere. Of course we don't know which bit is wrong, but we know at least one is because the total number of one-bits is even.

Checksums I run into are normally 8-bit or 16-bit "sums" of a large number of bytes or double bytes. I put "sums" in quotation marks because the checksum may be formed by the exclusive-or operation rather than true addition. In fact, it usually is. Eight-bit checksums formed with exclusive-or are in reality a kind of lengthwise parity. Each bit of the checksum is a parity bit for the column of bits in that position in the block of data.

In the old days, when dinosaurs first began to associate with herds of wildly spinning tape drives, you heard the words "vertical parity" and "longitudinal parity". Vertical parity was in those days a seventh bit for each six-bit character written on the tape, and longitudinal parity was a 7-bit character tacked on the end of each tape record, just like a checksum.

Enough review.

CRC is a much better scheme. A typical CRC implementation would add a 16-bit code to the end of a 256-byte block of data. A simple checksum would warn you of all single-bit errors, and some errors involving more than one bit. But CRC could detect all single and double bit errors, all errors with an odd number of error bits, all bursts of errors up to 16-bits in a row, and nearly all bursts of 17 or 18 bits in a row. Wow!

Also, you can even use CRC codes to CORRECT single-bit errors, if you trade off against some detection of longer error bursts.

You will run into CRC if you look into hard disks, or well- written modem software.

I like to write well-written programs, so I have been trying to understand CRC for some time now. A long time ago I had access to a book called "Error Correcting Codes", which is a classic. But I can't locate a copy now. I have seen numerous articles on the topic, especially in Computer Design. There was even one in Byte, Sept. 83, page 438. But I couldn't make the program in Byte work the way CRC's are supposed to, and I don't save my old Computer Design magazines.

Bobby Deen to the rescue. Bobby had a copy of a public domain routine by Paul Hansknecht, of Carpenter Associates, Box 451, Bloomfield Hills, MI 48013. Actually four little subroutines, to:

* clear the CRC code value
* cycle the eight bits of a data byte through the CRC algorithm
* finish the CRC calculation for an outbound message
* check the CRC bytes of a received message.

What is the basic idea of CRC? You perform an algorithm on each bit of a block of data, and get a CRC value. You append the CRC value to the data, and transmit both data and CRC. The receiver performs the same algorithm on the total record, both the data and the CRC code; when finished, the result of the receiver's CRC algorithm should be zero. If not zero, there was an error.

I am speaking in terms of sending and receiving, as in transmitting data with a modem. It all applies equally to writing and reading records on a disk, or even in adding check codes to a ROM. The programs I wrote and will list here merely operate on a buffer in RAM, so that I can see what is happening. You can extend them to practical uses from this base.

Which brings us to algorithms. The one Bobby gave me works like this:

The 16-bit value is initialized to zero. Then each bit in the data buffer is presented one at a time where the input arrow is. The bits in the 16-bit value are all shifted left one position, and the new data bit comes in the right end to become the new bit 0. The bit which shifts out the left end is Exclusive-ORed with the bits now found in bits 12, 5, and 0. That is, if the bit shifted out was a zero, nothing happens. If the bit shifted out was a one, exclusive or the 16 bit value with $1021.

If you understand the math of cyclic polynomials (I don't), this is supposed to be equivalent to X^16 + X^12 + X^5 + 1. An organization known to me only as CCITT recommends this polynomial. Another good one is reputed to be X^16 + X^15 +X^2 + 1, which is implemented by changing the exclusive or value from $1021 to $8005.

After all the bits of the data have been processed through the algorithm, 16 more zero bits are shifted through. After the zeroes, the value in the CRC register is the CRC code we append to the data.

The "receiver" processes the data the same way, starting by zeroing the CRC register. But instead of ending by shifting in 16 more zeroes, the receiver ends by shifting in the CRC code received.

I wanted to see if it really could find all those kinds of errors mentioned above. I wrote a program which would compute the CRC value and append it to a data block. Then I wrote another program which would "receive" the block and print out the resulting CRC value. Then I modified it to one-by-one toggle each bit position in the entire block, simulating a single bit error in each bit position in the whole buffer. My buffer is 256 bytes long, so that means 8*256 or 2048, different error positions. Actually 2064, because of the two bytes of CRC.

This way I experimentally "discovered" that the value produced by the CRC computation on the received message is dependent on the error bit position. It doesn't matter what the data was. Therefore, if I had a lookup table of 2064 16-bit entries, I could search through it and find out which bit position was wrong. There must be an easier way to figure out which bit position is wrong, but that is one of the things I still need to learn.

Okay. CRC.BYTE (lines 2890-3060) is a subroutine to process the eight bits of one byte through the CRC algorithm. CRC.BYTE needs to be called once for each byte of data in the buffer, plus either two zero bytes for a SEND routine or two CRC bytes for a RECV routine.

CRC.BUFFER (lines 2700-2850) is a little subroutine which calls CRC.BYTE once for each byte in the extended buffer. I assume it is called with PNTR pointing at the first byte in the buffer, and LIMIT is equated to the byte just beyond the end. The extended buffer includes either two zeroes on the end, or the two CRC bytes.

SETUP (lines 2610-2690) is a subroutine to initialize the CRC value register to zeroes, and to set PNTR to point at the beginning of the buffer.

The SEND and RECV routines at lines 1160-1380 simulate "sending" and "receiving" the buffer. Note that both SEND and RECV finish by displaying the calculated CRC value. SEND also stores the calculated CRC value into the end of the extended buffer. RECV should end up with a CRC value of $0000, unless there have been bits changed between calls to SEND and RECV.

TEST.SINGLE.BIT.ERRORS (lines 1390-1800) is the testing subroutine which I described above. It calls CRC.BUFFER 2064 times. Each time a different bit is changed. I print out the resulting CRC code each time, eight to a line, with the address of the byte containing the error bit leading the line. Before running TEST.SINGLE.BIT.ERRORS, you should run SEND to be sure a valid CRC code is installed in the extended test buffer.

I wrote another test routine which tests all two-bit errors. See TEST.DOUBLE.BIT.ERRORS, lines 1810-2410. The only trouble is it would take about 72 hours to run, so I haven't let it go all the way. I designed it to step through every bit position in two nested loops. If both loops happen to be at the same bit position, the bit will be toggled twice resulting in no error. I designed the program to print the address of the current byte whenever there was no error.

You might experiment with error bursts of various lengths, which should take no longer than TEST.SINGLE.BIT.ERRORS to run.

I would really be interested in finding out how to go backwards from a non-zero received CRC value to correct single-bit errors. Is there some easy way, without either a huge table or a long computation? If any of you know how, or have an article that tells how, pass it along.

1

================================================================================
DOCUMENT :AAL-8404:Articles:Disasm.wExec.txt
================================================================================

Using EXEC Files with Rak-Ware's DISASM...........Bob Kovacs

[ Bob is the author of DISASM, owner of Rak-Ware ]

I recently received a phone call from Alan Lloyd who had just purchased DISASM. He wanted to use it to disassemble the Autostart ROM so he could customize the code for a particular application. He was frustrated by the limited editing capabilities of DISASM which makes you start all over again if you don't catch your mistake before hitting RETURN. Since he had to enter the starting and ending addresses of over a dozen data tables, he began searching for an easier (and less painful) way of entering the data. He decided to try using an EXEC file with DISASM, and it worked! Well, sort of.

I thought about the problems he ran into, and found out some interesting things about the S-C Macro Assembler along the way. It turns out that with the help of a small patch to DISASM that it is possible to run the entire program via "remote control" using an EXEC file.

The first step is to create the TEXT file that will later be EXECed. You can do this in a word processor, if your word processor makes ordinary DOS text files. Or you can write an Applesoft program to help you build an array of addresses and the proper answers to the various prompts in DISASM, and then write a complete EXEC file. I decided to use the S-C Macro Assembler, because you can use the TEXT <filename> command to write a text file. You can have the assembler in the language card, DISASM at $800, the thing to be disassembled wherever you want, and pop back and forth fast as lightning.

Just enter each line of "source" as if you were responding to the questions put to you be DISASM. You can even include lines to turn on display of DOS commands and I/O (MONIOC), and the BLOADing of DISASM and NAMETABLE.

The S-C Macro Assembler does make one thing difficult. Some of the questions asked by DISASM require a null line (a RETURN all by itself), and S-C makes it very hard to get a null line. The first of these is used to terminate the entry of data table addresses. (Alan was satisfied to have his EXEC file stop here and take over manually.)

Normally, S-C does not let you enter totally empty lines. Typing a line number without any following text is one of the ways to DELETE a line, just as in both BASIC's. After a little experimenting I discovered a trick to fool the S-C input routine. I still don't get a totally empty line, but I can put extra RETURNs into an existing line. Here's how:

1. Type in the text of all the non-null lines

you want in your EXEC file.

2. Use the EDIT command to insert extra RETURNs in the proper places: move the cursor to the character position desired, and type ctrl-O followed by RETURN to insert each extra RETURN. Each extra RETURN will show up as an inverse "M" as you are editing. Then type one more RETURN to exit the EDIT mode.

The next problem I ran into was the Y/N responses for the "Full Cross-Reference" and "Generate Text File" questions. DISASM looks directly at the keyboard for those two responses, so it is blind to any EXEC file inputs. A five byte patch fixes all that, so you can use EXEC file as well as keyboard inputs. Just change the code starting at location $C5A from AD 00 C0 10 FB to 20 18 FD 09 80.

The following arbitrary example illustrates how an EXEC file might look when typed into the S-C assembler (extra RETURNs are indicated by <M>):

```
1000 MONIOC
1010 BLOAD DISASM
1020 BLOAD NAMETABLE
1030 $800G          (Use call 2048 to EXEC from BASIC)
1040 2              (select S-C Assembler format)
1050 F800           (starting physical address)
1060 F9B9           (ending physical address)
1070 F800           (starting execution address)
1080 F8CD           (table #1 start)
1090 F8CF           (table #1 end)
1100 3              (table #1 format)
1110 F962           (table #2 start)
1120 F9A5           (table #2 end)
1130 5              (table #2 format)
1140 F9A6           (table #3)
1150 F9B3
1160 8
1170 F9B4           (table #4)
1180 F9B9
1190 6
1200 <M>2000        (end of tables, and NAMETABLE address)
1210 0              (no printer output)
1220 <M>NYDEMO      (RETURN for no single cross reference,
                     N for no full cross reference,
                     Y for creating a textfile named DEMO)
```

(Of course, you realize that the explanatory comments in parentheses are not supposed to be typed.) I advise you to SAVE the lines on a file as S-C source code, using the SAVE <filename> command. This will become the copy you re-LOAD when you want to make changes. Then use the TEXT <filename> command to write out the EXEC file. Finally, EXEC <filename> to run the disassembly!

When EXECing, the table addresses are entered at a blinding speed that is almost imposssible to follow. If your text file has an error in it such that it does not conform to the DISASM input syntax, then things can go very wrong very fast. For those of you who would rather not see things move along quite so fast, I suggest adding a small patch to the COUT vector which provides a short delay. The following program works fine:

```
     $300:48       PHA
          A9 80    LDA #$80
          20 A8 FC JSR $FCA8
          68       PLA
          4C F0 FD JMP $FDF0
```

You can hook this into DOS from the assembler by typing "$36:00 03 N 3EAG". Then change line 1030 above to $812G (or CALL 2048+18 for EXEC from BASIC) to bypass DISASM's effort to setup the default DOS vectors.

Or you can even include all this stuff along with the original EXEC file. Either way, it is easier to use DISASM with an EXEC file when there are lots of data tables to be entered and you have fumble-fingers at the keyboard.

From now on, DISASM will be shipped with the five-byte patch indicated above already installed, and with two sample EXEC files designed to be EXECed from BASIC.

```
==============================================================================
DOCUMENT :AAL-8404:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 4 -- Issue 7                              April, 1984

In This Issue...



Have we got news for you this month!

First, the simple announcements:  We now have a new S-C Macro Cross Assembler for the Zilog Z-8 microprocessor.  For only $32.50 Macro Assembler owners can add the ability to develop code for this popular chip.

And some good news for you Corvus hard disk owners:  The problem in the Macro Assembler with having your Target File on a different volume from your source files is now whipped.  Just send in your original Version 1.1 diskette for a free update.

Now the big story:  After repeated requests from many users, we have decided to make available the complete Source Code for S-C Macro Assembler Version 1.1.  See the last page of this issue for details.

Another product for which we have held back selling source code is the Double Precision Floating Point package for Applesoft (DPFP).  From now on that product will be sold WITH source code, at the unforgiveably low price of $50.  If you already are a registered owner of DPFP, or can supply other proof-of- purchase, we will send you the source code for $15.  In case you never heard of DPFP, it is a 2048-byte &-module that provides 21-digit arithmetic and I/O for Applesoft.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050.  Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8404:Articles:Ideas....txt
==============================================================================

IDEAS FOR V4N7

  MAKE YOUR OWN SHIFTKEY MOD

  INTELLEC FORMATTER

==============================================================================
DOCUMENT :AAL-8404:Articles:Intellec.Hex.txt
==============================================================================

Converting to Intellec Hex Format..........Bob Sander-Cederlof

The Prom Burners reviewed elsewhere in this issue all were designed especially for Apple owners, and plug directly into an Apple slot. Believe it or not, there are other computers.... There are many brands of industrial grade prom burners which are not specifically designed for a particular computer host. Most of these connect to a serial port on whatever host computer you choose.

Many of these expect to receive data in a special format, called by some the Intellec Hex Paper Tape Format. Or, since at least two of those capitalized words are rather old- fashioned, the Intellec Hex Format. Intellec is also used to communicate with a variety of Emulation hardware, and Development Systems.

The S-C Assemblers produce either binary files or the binary image in memory of the object code. Can we convert a file or range of RAM to the Intellec format, and send it via a serial port? Sure, it only takes a little software....

Let's first simplify a little by assuming we can BLOAD a binary file first into Apple RAM. Then we only need a program which can translate and send a block of Apple RAM.

I would like to be able to operate the program by a control-Y monitor command. I want to type what looks like the memory move ("M") command: the first address specifies to the target system where the data should load; the second and third addresses specify the Apple RAM to be sent. I also would like to be able to specify which slot the serial port is in, or where in RAM a subroutine to send bytes to the target system can be found if there is no intelligent interface card.

The program I wrote fulfills those wishes. It loads at $300, and self-installs a control-Y vector for the monitor. Location $0000 and $0001 must then be set to the low- and high-bytes of the port, and the "M"-like control-Y command can be typed. For example:

```
:BRUN B.INTELLEC
:$0:2 0
:$F800<800.FFF^O^Y
```

The port value is 0002, which means slot 2. I wrote the program so that a port value 0001 thru 0007 means a slot number; 0100 thru FFFF means a subroutine address for your own driver; 0000 means send the output where it already is directed when you type the control-Y command. The "^O^Y" on the third line above means "control-O control-Y", which is how you type a control-Y when you are in the S-C Assembler. If you type the command from the monitor (*-prompt), omit the control-O.

I chose to send the data in a form that is compatible with both Intel and Zilog specifications, as I understand them.  I do not have any equipment which expects this format around here, so I cannot test the program with live data.  If you do, and you find I have misinterpreted something, I would sure appreciate some feedback.

There are two types of records sent:  data and end-of-file records. Each record begins with a colon (:) and ends with carriage return linefeed (CRLF, which is $8D8A).  The records consist of five fields.

Record length field:  two hex digits which specify how many bytes of data will be in the data field.  Will be 00 for an end-of-file record. In keeping with Zilog's standard, the maximum length will be 32 bytes.

Address field:  four hex digits which specify the load address in a data record, and the run address in an end-of-file record.

Record type field:  00 for a data record, and 01 for an end-of-file record.

Data field:  two hex digits for each byte of data specified by the record length field.  Empty for an end-of-file record.

Checksum field:  two hex digits which represent the complement of the 8-bit sum of the 8-bit bytes which result from converting each pair of hex digits in the other four fields of this record to 8-bit binary values.

Lines 1250-1330 of the program set up the control-Y vector for the Apple Monitor.  If this is unfamiliar to you, you might like to read all about it in the October 1981 issue of Apple Assembly Line, pages 14-17.

Briefly, once set up, a control-Y command will branch to your own code.  It gives a way to extend the Apple monitor.  You can type up to three addresses before the control-Y, and they will be parsed by the monitor and saved in five two-byte variables (called A1, A2, A3, A4, and A5).  If you type "aaaa<bbbb.cccc" before the control-Y:

```
       aaaa will be saved in A4 and A5
       bbbb will be saved in A1 and A3
       cccc will be saved in A2
```

If you wish to pass more than three items of information to the control-Y routine, you can pre-store them in other locations.  In my routine, you must pre-store the port value in $0000 and $0001.

The whole control-Y routine is shown in just four lines of code: lines 1470-1500.  Of course, these are all subroutine calls.

TURN.ON.OUTPUT.PORT (lines 1510-1650) examines locations $0000 and 0001.  If they contain 0000, then the output port is not changed.  If they contain 0001 thru 00FF, the lower three bits are used to select

**an intelligent interface card in slot 1 through 7. A larger value indicates your own driver routine address.**

**TURN.OFF.OUTPUT.PORT (lines 2010-2030) sets the output back to the Apple screen.**

**SEND.DATA.RECORDS (lines 1660-1890) divides the area to be transmitted into a number of 32-byte blocks. Each block is send as one data record. The final block may be less than 32 bytes.**

**SEND.EOF.RECORD (lines 1900-2000) sends the end-of-file record. The original loading address is assumed to be the run address. If you would rather send 0000 for a run address, you can change lines 1960 and 1980 to "LDA #0".**

**SEND.RECORD (lines 2050-2330) formats and transmits one record of either type, using the count, address, and type information already setup by the caller. It also updates A1 and A4 for the next record.**

**SEND.BYTE (lines 2340-2420) accumulates a byte in the checksum, and then converts it to two hex digits and transmits it.**

**You can use this program with any of the S-C Macro Assemblers or Cross Assemblers, exactly as shown. If you are using some other brand of assembler, you will probably have to leave the assembler environment to load this program, load the object code you wish to transmit, and run the program.**

```
================================================================
DOCUMENT :AAL-8404:Articles:My.Ad.txt
================================================================

S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1 Update...........................$12.50
Source Code for Version 1.1 (on two disks).........................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).....................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984

OBJ.APWRT][F (Don Lancaster, Synergetics)............................$30
Quick-Trace (Anthro-Digital).............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
Amper-Magic (Anthro-Digital).............................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $35)  $30
Routine Machine (Southwestern Data Systems)............(reg. $64.95)  $60
FLASH! Integer BASIC Compiler (Laumer Research).......................$79
Fontrix (Data Transforms).............................................$75
Aztec C Compiler System (Manx Software).................(reg. $199) $180
IACcalc Spreadsheet Program............................(reg. $84.95)  $75
          The one we use every day.  It's better than Visicalc!
Locksmith 5.0 (Omega MicroWare)........................(reg. $99.95)  $90

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                           or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ..........................$20

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($17.95)  $17
    "Incredible Secret Money Machine", Lancaster............($7.95)   $7
```

```
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Bag of Tricks", Worth & Lechner, with diskette.........($39.95)  $36
    "Assembly Lines: The Book", Roger Wagner................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($24.95)  $23
    "What's Where Guide" (updates first edition).............($9.95)   $9
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($17.95)  $17
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

==============================================================================
DOCUMENT :AAL-8404:Articles:New.Source.Code.txt
==============================================================================

Macro Source Code Now Available.............Bob Sander-Cederlof

We have finally become convinced that we should make the source code of our S-C Macro Assembler available for purchase. Many of you have requested, for a long time now. We have resisted, I suppose through a mild case of the same paranoia which causes so many other software publishers to use copy protection and license agreements (which we eschew).

We have absolutely no experiential basis for mistrust. You have all treated our previous offerings of source code in the most honorable fashion, and we expect you will continue to do so.

Effective immediately, registered owners of Version 1.1 of the S-C Macro Assembler can purchase the source code for $100. You will be able to assemble it to obtain a paper listing, study it to learn techniques, and modify it to your own tastes. We hope many of you will make improvements and send them back to us for inclusion in future versions.

The code resides on two nearly-full diskettes. You need at least two drives to assemble it. The source is fully commented, and is organized in a logical easy-to-follow manner.

If you do not yet own Version 1.1, you may purchase or upgrade to it simultaneously with the purchase of the source code, if you wish. If you are one of those who purchased the Version 4.0 source code, we will give you $40 credit toward the purchase of the Macro 1.1 source.

Another product for which we have held back selling source code is the Double Precision Floating Point package for Applesoft (DPFP). From now on that product will be sold WITH source code, at the unforgiveably low price of $50. If you already are a registered owner of DPFP, or can supply other proof-of-purchase, we will send you the source code for $15. In case you never heard of DPFP, it is a 2048-byte &-module that provides 21-digit arithmetic and I/O for Applesoft.

==============================================================================
DOCUMENT :AAL-8404:Articles:Quick.DOS.Updtr.txt
==============================================================================

Quick DOS Updating vs MASTER.CREATE........Bob Sander-Cederlof

When DOS was young, Apples tended to have varying amounts of memory under 48K. Some had 16K, which was the standard purchase at a computer store; others 24K, with one row of 16K and two of 4K; others 32K; and some 48K. Trying to write a DOS image that would fit all of these memories was quite a task.

Apple introduced the concept of a "master" and a "slave" disk. Master disks have a generic image of DOS. The boot process first loads the DOS image as though the machine only has 16K RAM, and then the image is relocated as high as possible in memory. Slave disks have a frozen image, already relocated for a particular memory size. The INIT command always creates a slave disk. In order to make a master disk you either copy and old master using COPYA (or equivalent copy program), or you use the MASTER.CREATE program on the DOS System Master Disk. (For a while the MASTER.CREATE program was called UPDATE 3.3.)

But now! But now you will have a difficult time finding an Apple with less than 48K memory. After all, the chips are only about a dollar apiece, or $8 to $12 for a set of eight. Who needs master disks anymore?

A lot of people think they do, because MASTER.CREATE is there and the reference manual makes such a big deal about it. And this causes a problem. What if I want a master disk with a modified DOS? MASTER.CREATE always reads the DOS image off the system master disk, and it is unmodified. Well, you can use a disk zap program on a copy of the system master.

Or, you can forget all about MASTER.CREATE and use my handy- dandy little patch installer. The program which follows reads the DOS image from the first 3 tracks into memory from $4000 thru $64FF. Then it installs patches from a table of patches; this part is almost identical to the patch installer published in the April 1983 issue of AAL. Finally it writes the patched DOS back on the first three tracks. And it does all this so fast you'll think it never happened.

Once you have coded the patches you want, and have tested them, you can update all your old DOS 3.3 disks almost as fast as you can open and close the drive door. With slight modifications, you could have it write the patched image on successive disks without re-reading and re-patching each time.

Looking at the program, Lines 1200-1240 do the overall job. Just below that, lines 1260-1290 give two entry points to a block of code that sets up an IOB for RWTS and then calls RWTS. The only difference between the two calls is the opcode, either READ or WRITE. Below that point, there is a backwards loop that counts from track 2, sector 4, back to track 0, sector 0. Just for fun, I print out the track and sector numbers just before reading or writing each sector. (If you get tired of the fun, simply delete line 1450, the JSR $F941.)

The DOS image on tracks 0, 1, and 2 is not in exactly the same order as you find it in memory after booting. Therefore the patcher maps patch addresses to the new locations. Lines 1060-1080 define the remapping constants. Addresses which in the running image will be between $B600 and $BFFF will be located from $4000 thru $49FF. If the original was a master, code which does the relocating part of the boot will be found from $4A00 thru $4BFF. The code between $9D00 and $B5FF will be found from $4C00 thru $64FF. The two constants DOS.9D and DOS.B6 are used in figuring the application points of the patches in lines 2110, 2350, and 2540.

For a full explanation of lines 1590-1900, see the April 1983 AAL, pages 24-27. The patch set up to be installed in lines 2020-2580 is the fast LOAD, BLOAD, RUN, BRUN patch from pages 2-8 of the same issue.

==============================================================================
DOCUMENT :AAL-8404:Articles:Woz.Talks.txt
==============================================================================

An Evening with Woz.................................Bill Morgan

Well, maybe not a whole evening, but a good portion of it. And certainly not alone, there were about 150 others in the room. But I did have the opportunity to attend a dinner sponsored by the River City Apple Corps, in Austin, Texas, and hear a speech by Steve Wozniak, the designer of our favorite pastime.

Most of Steve's speech was devoted to the history of his involvement with computers, and the development of the Apple II. That story is pretty well-known by now, so I won't mention too much of it here. The most interesting facets to me were hearing how much of a prankster Woz has always been, and finding out how many features of the Apple II were motivated only by Steve's desire to write a Breakout game in BASIC.

My favorite part of the evening was the question-and-answer session and the informal chats afterward, when we all got our chance to ask about what we really wanted to know. The first question is mine, the rest came from all around the room. These are the items that seem to be of most concern to AAL readers:

How about 65816 machines?

We're heavily involved in a computer based around that chip. But, final computer becoming a full-fledged product is subject to too many other variations, such as: you start working on it and you decide, no, this computer didn't come out right, it's too long, the actual date it will be done, it's not enough, we have to do a different product. So, it may be as soon as a few months, and it may be as long as a couple years before Apple has a product based around that new processor. Fortunately it is 100% compatible with what we've done before. Meaning it's a compatible upgrade, and that's what the Apple II has to do.

When can we expect a portable //e?

It's ... in the works. We're certainly thinking about it and delving into it and unless the project gets cancelled, probably very soon, but you can never say for sure until it's out.

How about color on the Macintosh?

There is no color on the Macintosh. ... Laser printers ... (and) ... LCD displays ... are converging on black and white technology being appropriate for that product line. There is no color for the Macintosh at this time.

Do you expect to see the 3 1/2 inch disks on the //e?

Apple believes that it's time to start moving the entire company toward higher density, better technology, more modern technology disk drives, and the 3 1/2 inch disk drives from Sony that is in the Lisa and Macintosh computers now is the proper direction to move in. It'll be interesting to see how it unfolds over time, as to how the conversion is made and yet extreme compatibility and support taken into account. All the software exists today on 5 1/4 inch disks. How do we get there?

It could be like your second disk can be a nice 3 1/2 inch with a lot more storage capability, but it may be years before it's proper to expect bootable software, to be able to boot on 3 1/2 inch drives. It's a challenge, and it just can't be turned over overnight. We could come out with a product for the Apple II today that uses a 3 1/2 inch drive as your only drive, and you know you can't run any of your software on it.... The sales of such a product would not start until there was a software base established.

What are you personally working on?

I'm interested in the future Apple II families. We're always pursuing higher performance-to-cost versions of the Apple II. And sometimes that's achieved by integrating several chips down into one custom chip, or by looking at accessories that are very commonplace, that almost everyone's going to buy for their //e. You can build one version of it with a lot of those accessories in and save a lot of money in the end, a lot of hassle. There are ways to improve the cost/performance ratio.

The other end, we're always trying to improve the capabilities of the machine. How are we going to eventually, someday, challenge IBM in the multi-megabyte computer world, the high-end? How are we going to improve performance?, increase screen resolution?, all those sort of questions, those sort of enhancements. I've been working very closely on one of those projects in Apple since returning.

... I think we've got to start heading towards a real, more useful home machine that does have a few of the things that we originally pursued, that we now believe is only about 10% of our market. Things such as: speech recognition and speech generation, built in, because they're relatively inexpensive and easy to do now to some level of quality. And it should also have more of the home-ish features, the features that are used in a personal, home environment built in.

So, that's the gist of it. I would like to thank Stuart Greenfield, of the River City Apple Corps, for the invitation to attend their dinner, and of course thank you, Woz, for coming to visit us.

One last note: Steve referred to a portable Apple //e as "probably very soon". Lately we've been hearing about the Apple //c, a 9-pound machine sporting 128K RAM, one disk drive, built-in serial and parallel ports, and no slots. Also no monitor, which sounds a little strange.  Price -- $1200.  The //c announcement is expected in late April.

```
==========================================================================
DOCUMENT :AAL-8404:DOS3.3:S.ApplyDOSPatch.txt
==========================================================================

 1000  *SAVE S.APPLY DOS PATCHES
 1010  *--------------------------------
 1020  PNTR        .EQ $00,01
 1030  PATCH       .EQ $02,03
 1040  SECTOR.CNT .EQ $04
 1050  *--------------------------------
 1060  DOS.IMAGE  .EQ $4000 - $64FF
 1070  DOS.9D      .EQ $9D00-DOS.IMAGE-$0C00
 1080  DOS.B6      .EQ $B600-DOS.IMAGE
 1090  *--------------------------------
 1100  GETIOB      .EQ $3E3
 1110  RWTS        .EQ $3D9
 1120  *--------------------------------
 1130  IOB         .EQ $B7E8
 1140  IOB.VOLUME .EQ IOB+3
 1150  IOB.TRACK  .EQ IOB+4
 1160  IOB.SECTOR .EQ IOB+5
 1170  IOB.BUFADR .EQ IOB+8
 1180  IOB.OPCODE .EQ IOB+12
 1190  *--------------------------------
 1200  PATCH.DOS
 1210         JSR READ.DOS.IMAGE
 1220         JSR PATCHER
 1230         JSR WRITE.DOS.IMAGE
 1240         RTS
 1250  *--------------------------------
 1260  READ.DOS.IMAGE
 1270         LDA #$01     READ OPCODE
 1280         .HS 2C
 1290  WRITE.DOS.IMAGE
 1300         LDA #$02     WRITE OPCODE
 1310         STA IOB.OPCODE
 1320         LDA #0
 1330         STA IOB.BUFADR
 1340         STA IOB.VOLUME
 1350         LDA #DOS.IMAGE/256+16+16+5-1
 1360         STA IOB.BUFADR+1
 1370         LDA #2       TRACK 2
 1380         STA IOB.TRACK
 1390         LDA #4       SECTOR 4
 1400         STA IOB.SECTOR
 1410         LDA #16+16+5
 1420         STA SECTOR.CNT
 1430  .1     LDA IOB.TRACK
 1440         LDX IOB.SECTOR
 1450         JSR $F941
 1460         JSR GETIOB
 1470         JSR RWTS
 1480         LDY IOB.SECTOR
```

```
1490        DEY
1500        BPL .2
1510        LDY #15
1520        DEC IOB.TRACK
1530 .2     STY IOB.SECTOR
1540        DEC IOB.BUFADR+1
1550        DEC SECTOR.CNT
1560        BNE .1
1570        RTS
1580 *--------------------------------
1590 PATCHER
1600        LDA #PATCHES-1
1610        STA PNTR
1620        LDA /PATCHES-1
1630        STA PNTR+1
1640        LDY #0
1650
1660 .1     JSR GET.BYTE LENGTH OF NEXT PATCH
1670        BEQ .4       FINISHED
1680        TAX          SAVE LENGTH IN X
1690        JSR GET.BYTE ADDRESS OF PATCH
1700        STA PATCH
1710        JSR GET.BYTE
1720        STA PATCH+1
1730
1740 .2     JSR GET.BYTE
1750        STA (PATCH),Y
1760        INC PATCH
1770        BNE .3
1780        INC PATCH+1
1790 .3     DEX
1800        BNE .2
1810        BEQ .1       ...ALWAYS
1820
1830 .4     RTS
1840 *--------------------------------
1850 GET.BYTE
1860        INC PNTR
1870        BNE .1
1880        INC PNTR+1
1890 .1     LDA (PNTR),Y
1900        RTS
1910 *--------------------------------
1920 PATCHES
1930 *--------------------------------
1940 *  S.FAST LOAD
1950 *
1960 *      FAST "LOAD" AND "BLOAD"
1970 *
1980 *      INSTALLED IN UNUSED AREAS IN DOS 3.3:
1990 *          $BA69-$BA95   (45 BYTES FREE)
2000 *          $BCDF-$BCFF   (33 BYTES FREE)
2010 *--------------------------------
2020 READ.RANGE          .EQ $AC96
```

```
2030  READ.NEXT.SECTOR    .EQ $B0B6
2040  END.OF.DATA.ERROR   .EQ $B36F
2050  RANGE.LENGTH        .EQ $B5C1,C2
2060  RANGE.ADDRESS       .EQ $B5C3,C4
2070  BUFFER.ADDRESS      .EQ $B5CB,CC
2080  SECTOR.COUNT        .EQ $B5E4,E5
2090  BYTE.OFFSET         .EQ $B5E6
2100  *---------------------------------
2110         .DA #P1.LENGTH,$BA69-DOS.B6
2120         .PH $BA69
2130  PATCH1 LDA BYTE.OFFSET       LAST BYTE OF
2140         BNE GO.READ.RANGE     A SECTOR?
2150         LDA RANGE.LENGTH+1    WHOLE SECTOR LEFT?
2160         BEQ GO.READ.RANGE     NO.
2170         LDA BUFFER.ADDRESS    SAVE BUFFER ADDRESS
2180         PHA
2190         LDA BUFFER.ADDRESS+1
2200         PHA
2210         LDA RANGE.ADDRESS     READ DIRECTLY
2220         STA BUFFER.ADDRESS    INTO RANGE
2230         LDA RANGE.ADDRESS+1
2240         STA BUFFER.ADDRESS+1
2250  READ.LOOP
2260         JSR READ.NEXT.SECTOR
2270         BCS .1
2280         JMP PATCH2
2290  .1     JMP END.OF.DATA.ERROR
2300  GO.READ.RANGE
2310         JMP READ.RANGE
2320  P1.LENGTH .EQ *-PATCH1
2330         .EP
2340  *---------------------------------
2350         .DA #P2.LENGTH,$BCDF-DOS.B6
2360         .PH $BCDF
2370  PATCH2 INC SECTOR.COUNT
2380         BNE .1
2390         INC SECTOR.COUNT+1
2400  .1     INC RANGE.ADDRESS+1   NEXT PAGE
2410         INC BUFFER.ADDRESS+1
2420         DEC RANGE.LENGTH+1
2430         BNE .2
2440         PLA               RESTORE BUFFER
2450         STA BUFFER.ADDRESS+1
2460         PLA
2470         STA BUFFER.ADDRESS
2480         JMP READ.RANGE
2490
2500  .2     JMP READ.LOOP
2510  P2.LENGTH .EQ *-PATCH2
2520         .EP
2530  *---------------------------------
2540         .DA #P3.LENGTH,$ACA5-DOS.9D
2550         .PH $ACA5
2560  PATCH3 JMP PATCH1
```

```
2570  P3.LENGTH .EQ *-PATCH3
2580         .EP
2590  *--------------------------------
2600         .DA #0       END OF PATCHES
```

```
==========================================================================
DOCUMENT :AAL-8404:DOS3.3:S.CRCHansKnecht.txt
==========================================================================

 1000  *SAVE S.CRC GENERATOR (HANSKNECHT)
 1010  *--------------------------------
 1020  BUFFER .EQ $4000
 1030  LIMIT  .EQ $4102
 1040  *--------------------------------
 1050  CRC    .EQ $00,01
 1060  PNTR   .EQ $02,03
 1070  TPTR   .EQ $04,05
 1080  TMASK  .EQ $06
 1090  SPTR   .EQ $07,08
 1100  SMASK  .EQ $09
 1110  *--------------------------------
 1120  PRNTAX .EQ $F941
 1130  CROUT  .EQ $FD8E
 1140  PRBYTE .EQ $FDDA
 1150  COUT   .EQ $FDED
 1160  *--------------------------------
 1170  *      SIMULATE SENDING A BUFFER-FULL
 1180  *--------------------------------
 1190  SEND   JSR SETUP    CLEAR CRC, POINT AT BUFFER
 1200         LDA #0       CLEAR CRC BYTES IN BUFFER
 1210         STA LIMIT-1
 1220         STA LIMIT-2
 1230         JSR CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1240         LDX CRC      STORE CRC INTO LAST 2 BYTES
 1250         LDA CRC+1
 1260         STX LIMIT-1
 1270         STA LIMIT-2
 1280         JSR PRNTAX   DISPLAY THE CRC
 1290         JMP CROUT    <RETURN> AND RETURN
 1300  *--------------------------------
 1310  *      SIMULATE RECEIVING A BUFFER-FULL
 1320  *--------------------------------
 1330  RECV   JSR SETUP    CLEAR CRC, POINT AT BUFFER
 1340         JSR CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1350         LDX CRC      DISPLAY CRC IN HEX
 1360         LDA CRC+1
 1370         JSR PRNTAX
 1380         JMP CROUT
 1390  *--------------------------------
 1400  *      TRY "RECEIVING" THE 258 BYTES
 1410  *      WITH A KNOWN SINGLE-BIT ERROR.
 1420  *--------------------------------
 1430  TEST.SINGLE.BIT.ERRORS
 1440         LDA #BUFFER
 1450         STA TPTR     FOR TPTR = BUFFER TO LIMIT
 1460         LDA /BUFFER
 1470         STA TPTR+1
 1480  .1     LDA TPTR+1        PRINT TPTR"-"
```

```
1490        LDX TPTR
1500        JSR PRNTAX
1510        LDA #"-"
1520        JSR COUT
1530        LDA #$80          FOR TMASK =
1540        STA TMASK             $80,40,20,10,8,4,2,1
1550 .2     LDY #0
1560        LDA (TPTR),Y          INVERT BIT, MAKING ERROR
1570        EOR TMASK
1580        STA (TPTR),Y
1590        JSR SETUP    CLEAR CRC, POINT AT BUFFER
1600        JSR CRC.BUFFER        COMPUTE CRC
1610        LDA #" "              PRINT " "CRC
1620        JSR COUT
1630        LDA CRC+1
1640        LDX CRC
1650        JSR PRNTAX
1660        LDA (TPTR),Y          FIX ERRONEOUS BIT
1670        EOR TMASK
1680        STA (TPTR),Y
1690        LSR TMASK         NEXT TMASK
1700        BNE .2            ...MORE
1710        JSR CROUT         PRINT<CR>
1720        INC TPTR     NEXT TPTR
1730        BNE .3
1740        INC TPTR+1
1750 .3     LDA TPTR
1760        CMP #LIMIT
1770        LDA TPTR+1
1780        SBC /LIMIT+1
1790        BCC .1       ...MORE
1800        RTS
1810 *--------------------------------
1820 TEST.DOUBLE.BIT.ERRORS
1830        LDA #BUFFER
1840        STA SPTR     FOR SPTR=BUFFER TO LIMIT
1850        LDA /BUFFER
1860        STA SPTR+1
1870 *--------------------------------
1880 .1     LDA #$80     FOR SMASK=80,40,20,10,8,4,2,1
1890        STA SMASK
1900 *--------------------------------
1910 .2     LDA #BUFFER  FOR TPTR=BUFFER TO LIMIT
1920        STA TPTR
1930        LDA /BUFFER
1940        STA TPTR+1
1950 *--------------------------------
1960 .3     LDA #$80     FOR TMASK=80,40,20,10,8,4,2,1
1970        STA TMASK
1980 *--------------------------------
1990 .4     LDY #0
2000        LDA (TPTR),Y      MAKE FIRST ERROR
2010        EOR TMASK
2020        STA (TPTR),Y
```

```
2030        LDA (SPTR),Y      MAKE SECOND ERROR
2040        EOR SMASK
2050        STA (SPTR),Y
2060        JSR SETUP    CLEAR CRC, POINT AT BUFFER
2070        JSR CRC.BUFFER    COMPUTE CRC
2080        LDA (SPTR),Y      FIX BOTH ERRORS
2090        EOR SMASK
2100        STA (SPTR),Y
2110        LDA (TPTR),Y
2120        EOR TMASK
2130        STA (TPTR),Y
2140 *---------------------------------
2150        LDA CRC      IF CRC=0, DISPLAY POINTERS
2160        ORA CRC+1
2170        BNE .5       ...CRC .NE. 0, SO CONTINUE
2180        JSR DISPLAY.POINTERS
2190 *---------------------------------
2200 .5     LSR TMASK    NEXT TMASK
2210        BNE .4       ...MORE
2220        INC TPTR     NEXT TPTR
2230        BNE .6
2240        INC TPTR+1
2250 .6     LDA TPTR
2260        CMP #LIMIT
2270        LDA TPTR+1
2280        SBC /LIMIT+1
2290        BCC .3       ...MORE
2300 *---------------------------------
2310        LSR SMASK    NEXT SMASK
2320        BNE .2       ...MORE IN THIS BYTE
2330        INC SPTR     NEXT SPTR
2340        BNE .7
2350        INC SPTR+1
2360 .7     LDA SPTR
2370        CMP #LIMIT
2380        LDA SPTR+1
2390        SBC /LIMIT+1
2400        BCC .1       ...MORE
2410        RTS
2420 *---------------------------------
2430 DISPLAY.POINTERS
2440        LDA TPTR+1   PRINT TPTR"-"TMASK" ";
2450        LDX TPTR
2460        JSR PRNTAX
2470        LDA #"-"
2480        JSR COUT
2490        LDA TMASK
2500        JSR PRBYTE
2510        LDA #" "
2520        JSR COUT
2530        LDA SPTR+1   PRINT SPTR"-"SMASK
2540        LDX SPTR
2550        JSR PRNTAX
2560        LDA #"-"
```

```
                JSR COUT
                LDA SMASK
                JSR PRBYTE
                JMP CROUT
       *--------------------------------
       SETUP    LDA #0       CLEAR CRC
                STA CRC
                STA CRC+1
                LDA #BUFFER  SET UP PNTR TO BUFFER
                STA PNTR
                LDA /BUFFER
                STA PNTR+1
                RTS
       *--------------------------------
       *        COMPUTE CRC FROM (PNTR) THRU LIMIT
       *--------------------------------
       CRC.BUFFER
       .1       LDY #0       SCAN THRU THE BUFFER
                LDA (PNTR),Y
                JSR CRC.BYTE
                INC PNTR     NEXT BYTE
                BNE .2
                INC PNTR+1
       .2       LDA PNTR     CHECK LIMIT
                CMP #LIMIT
                LDA PNTR+1
                SBC /LIMIT
                BCC .1       MORE TO GO
                RTS
       *--------------------------------
       *        COMPUTE CRC ON A SINGLE BYTE
       *--------------------------------
       CRC.BYTE
                LDX #8       DO 8 BITS
       .1       ASL          MSB OF BYTE TO CARRY
                ROL CRC
                ROL CRC+1
                BCC .2       --> 0, GET NEXT BIT
                PHA          --> 1, TOGGLE POLYNOMIAL BITS
                LDA CRC
                EOR #$21     TOGGLE BITS 0 AND 5
                STA CRC
                LDA CRC+1
                EOR #$10     TOGGLE BIT 12
                STA CRC+1
                PLA
       .2       DEX          NEXT BIT
                BNE .1
                RTS
       *--------------------------------
       *   FIND WHICH BIT IS BAD IN BUFFER+CRC
       *
       *        RESULT IS BIT POSITION IN MESSAGE,
       *        WHERE THE FIRST BIT OF THE MESSAGE IS BIT 0
```

```
3110  *      AND (IN THIS CASE) THE LAST CRC BIT IS BIT $80F.
3120  *
3130  *      ALGORITHM BY BRUCE LOVE, NEW ZEALAND
3140  *--------------------------------
3150  BIT.NUMBER .EQ $10,11
3160  DUMMY.CRC  .EQ $12,13
3170  *--------------------------------
3180  FIND.BAD.BIT
3190         LDA #$80F    TOTAL # BITS - 1
3200         STA BIT.NUMBER    (WE WILL COUNT BACKWARDS)
3210         LDA /$80F
3220         STA BIT.NUMBER+1
3230         LDA #$0001   STARTING POINT FOR BIT FINDER
3240         STA DUMMY.CRC
3250         LDA /$0001
3260         STA DUMMY.CRC+1
3270  .1     LDA CRC      COMPARE RECEIVED CRC WITH
3280         CMP DUMMY.CRC        PROCESSED VALUE;
3290         BNE .2       IF THEY MATCH, WE HAVE FOUND THE
3300         LDA CRC+1    BAD BIT.
3310         CMP DUMMY.CRC+1
3320         BEQ .4       ...FOUND BAD BIT!
3330  .2     LDA BIT.NUMBER        DECREMENT BIT COUNTER
3340         BNE .3
3350         DEC BIT.NUMBER+1
3360         BMI .5       WENT TOO FAR
3370  .3     DEC BIT.NUMBER
3380         ASL DUMMY.CRC
3390         ROL DUMMY.CRC+1
3400         BCC .1
3410         LDA DUMMY.CRC
3420         EOR #$21
3430         STA DUMMY.CRC
3440         LDA DUMMY.CRC+1
3450         EOR #$10
3460         STA DUMMY.CRC+1
3470         JMP .1
3480  .4     LDA BIT.NUMBER+1  PRINT THE BIT NUMBER
3490         JSR PRBYTE        (IF $8000, THE ERROR WAS
3500         LDA BIT.NUMBER    NOT A SINGLE BIT)
3510         JSR PRBYTE
3520         JMP CROUT
3530  .5     BRK
3540  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8404:DOS3.3:S.Intellec.Hex.txt
==============================================================================

 1000  *SAVE S.INTELLEC HEX FORMATTER
 1010         .OR $300
 1020  *--------------------------------
 1030  PORT       .EQ $00,01
 1040  CHECK.SUM  .EQ $02
 1050  TYPE       .EQ $03
 1060  COUNT      .EQ $04
 1070  REMAINING  .EQ $05,06
 1080  *--------------------------------
 1090  A1     .EQ $3C,3D
 1100  A2     .EQ $3E,3F
 1110  A3     .EQ $40,41
 1120  A4     .EQ $42,43
 1130  A5     .EQ $44,45
 1140  *--------------------------------
 1150  CTRLY.VECTOR        .EQ $3F8 THRU $3FA
 1160  DOS.REHOOK          .EQ $3EA
 1170  *--------------------------------
 1180  MON.NXTA4           .EQ $FCB4
 1190  MON.CROUT           .EQ $FD8E
 1200  MON.PRHEX           .EQ $FDDA
 1210  MON.COUT            .EQ $FDED
 1220  MON.SETVID          .EQ $FE93
 1230  *--------------------------------
 1240  *      SETUP CONTROL-Y
 1250  *--------------------------------
 1260  SETUP  LDA #SEND.DATA
 1270         STA CTRLY.VECTOR+1
 1280         LDA /SEND.DATA
 1290         STA CTRLY.VECTOR+2
 1300         LDA #$4C
 1310         STA CTRLY.VECTOR
 1320         RTS
 1330  *--------------------------------
 1340  *   *0:XX YY   (LO,HI OF PORT)
 1350  *   *TARGET<START.END<Y>
 1360  *      IF PORT IS 0, DO NOT CHANGE OUTPUT
 1370  *      IF PORT IS 1...7, OUTPUT TO SLOT.
 1380  *         ELSE OUTPUT TO SUBROUTINE
 1390  *      SEND BYTES START...END
 1400  *
 1410  *      1.  TURN ON OUTPUT PORT
 1420  *      2.  SEND DATA RECORDS
 1430  *      3.  SEND EOF RECORD
 1440  *      4.  TURN OFF OUTPUT PORT
 1450  *--------------------------------
 1460  SEND.DATA
 1470         JSR TURN.ON.OUTPUT.PORT
 1480         JSR SEND.DATA.RECORDS
```

```
1490        JSR SEND.EOF.RECORD
1500        JMP TURN.OFF.OUTPUT.PORT
1510 *--------------------------------
1520 TURN.ON.OUTPUT.PORT
1530        LDX PORT+1      HI-BYTE OF PORT SPECIFIED
1540        BNE .1
1550        LDA PORT        LO-BYTE, MUST BE SLOT
1560        AND #$07
1570        BEQ .3         SLOT 0, JUST SCREEN
1580        ORA #$C0
1590        BNE .2         ...ALWAYS
1600 .1     TXA            HI-BYTE OF SUBROUTINE
1610        LDX PORT        LO-BYTE OF SUBROUTINE
1620 .2     STA $37
1630        STX $36
1640        JSR DOS.REHOOK
1650 .3     RTS
1660 *--------------------------------
1670 SEND.DATA.RECORDS
1680        LDA #0
1690        STA TYPE
1700        INC A2         POINT JUST BEYOND THE END
1710        BNE .1
1720        INC A2+1
1730 .1     SEC
1740        LDX #32
1750        LDA A2         SEE HOW MANY BYTES LEFT
1760        SBC A1
1770        STA REMAINING
1780        LDA A2+1
1790        SBC A1+1
1800        STA REMAINING+1
1810        BNE .2         USE MIN(32,A2-A1+1)
1820        CPX REMAINING
1830        BCC .2
1840        LDX REMAINING
1850        BEQ .3         ...FINISHED
1860 .2     STX COUNT
1870        JSR SEND.RECORD
1880        JMP .1         ...ALWAYS
1890 .3     RTS
1900 *--------------------------------
1910 SEND.EOF.RECORD
1920        LDY #0
1930        STY COUNT
1940        INY
1950        STY TYPE
1960        LDA A5         RUN ADDRESS (LO)
1970        STA A4
1980        LDA A5+1       RUN ADDRESS (HI)
1990        STA A4+1
2000        JMP SEND.RECORD
2010 *--------------------------------
2020 TURN.OFF.OUTPUT.PORT
```

```
2030        JSR MON.SETVID
2040        JMP DOS.REHOOK
2050 *--------------------------------
2060 SEND.RECORD
2070        LDA #":"
2080        JSR MON.COUT
2090        LDA #0
2100        STA CHECK.SUM
2110        LDA COUNT
2120        JSR SEND.BYTE
2130        LDA A4+1
2140        JSR SEND.BYTE
2150        LDA A4
2160        JSR SEND.BYTE
2170        LDA TYPE
2180        JSR SEND.BYTE
2190        LDA COUNT
2200        BEQ .2
2210        LDY #0
2220 .1     LDA (A1),Y
2230        JSR SEND.BYTE
2250        JSR MON.NXTA4
2260        DEC COUNT
2270        BNE .1
2280 .2     SEC
2285        LDA #0
2290        SBC CHECK.SUM
2300        JSR SEND.BYTE
2310        JSR MON.CROUT
2320        LDA #$8A     LINEFEED
2330        JMP MON.COUT
2340 *--------------------------------
2350 SEND.BYTE
2360        PHA
2370        CLC
2380        ADC CHECK.SUM
2390        STA CHECK.SUM
2400        PLA
2410        JMP MON.PRHEX
2420 *--------------------------------
```

=============================================================================
DOCUMENT :AAL-8405:Articles:Differences.txt
=============================================================================

Making a Map of Differences................Bob Sander-Cederlof

Many times I have had two versions of the same program, and wondered where the differences might be.

For example, where are the differences between DOS 3.2 and 3.3, or between the various releases of DOS 3.3? And now that Apple has sent out some pre-releases of a new set of CDEF ROMs for the //e, where are the differences between these and the current //e ROMs?

I have always used the monitor V command to find them. By doing it a small piece at a time, I can pinpoint the changes. Then I turn on my printer and use the L command to document the new version wherever there are differences. But the piecemeal use of the V command wastes a lot of time. I wish I had some way of printing a complete map of all the differences....

What if I had a command which would compare two areas of memory, and print a map of differences? I could use a "." to represent matching locations, and a "*" to represent those that do not match. I could print either 32 or 64 per line: 32 on a 40-column screen, 64 on an 80-column screen or printer. Then I could tell at a glance where all changes had occurred!

I looked at the October 1981 issue of AAL to find out how to use the control-Y monitor command to add a new monitor feature. Then I looked in the listing of the monitor ROM (in my old "red" Apple Reference Manual) at the code for the V command and the command which prints a range of memory.

The program on the next page is the result.

Lines 1150-1190 set up the monitor control-Y vector. Booting DOS stores a branch which effectively makes the control-Y command do nothing. Storing the address of a real program there allows you to add your own commands to the monitor. Once installed, typing a control-Y into the monitor will execute the program named DIFFERENCES.

When we get there, if we typed a full length monitor command of the form "address1<address2.address3^Y" (by "^Y" I mean control-Y), all three of the addresses will have been converted to binary and stored in some standard locations. Address1 will be in $42 and $43, address2 in $3C and $3D, and address3 in $3E and $3F. We will interpret the addresses to mean to compare the block of memory beginning at address1 with the block running from address2 through address3.

Line 1220 prints a carriage return, the current address value in $3C and $3D, and a dash. Lines 1230-1280 compare the bytes at corresponding positions in the two blocks of memory, and select either a "." or a "*" accordingly. Line 1290 prints the selected character.

Lines 1300-1310 increment the two base addresses to point to the next byte in both memory blocks. The new address2 is also compared to address3 to see if we are finished yet.
Lines 1320-1350 check to see if we have printed all 32 on the current screen line. If not, back to .1 to print the next one. Otherwise, all the way back to print a new address and dash, starting a new line. If you want 64 bytes per line, change the mask in line 1330 from #$1F to #$3F. You might want to have the program check to see whether 80-columns is turned on or not, and automatically select #$1F or #$3F accordingly. You could also check to see if the output hook at $36, 37 is pointing at a printer, and use the longer lines.

Experiment. You'll learn a lot and have a lot of fun at the same time!

==========================================================================
DOCUMENT :AAL-8405:Articles:DP18.Part.1.txt
==========================================================================

Decimal Floating Point Arithmetic..........Bob Sander-Cederlof

Perhaps you have wondered why PRINT INT(14.9 * 10) in Applesoft prints 148. This and many other such seeming bugs are a very common idiosyncrasy in the computer world.

Applesoft use binary floating point format for storing numbers and doing arithmetic. The number 14.9 is very clean in decimal, but it is an awful mess in binary. If you look at what is stored in RAM after doing X=14.9, you will find 84 6E 66 66 66. The first byte, 84, means the remaining four should be understood as four bits of binary integer (the "14" of "14.9") and 28 bits of binary fraction (the ".9" part). The first bit of the second byte is zero, which means the number is positive. Applesoft stores the sign in this bit position, knowing that ALL values other than 0.0 will have a 1-bit in this position of the magnitude.

Just before doing any arithmetic on the value above, Applesoft will unpack it, separating the sign, binary exponent, and the rest. The fancy name for the rest is the "mantissa". Writing out the mantissa for 14.9 we see EE 66 66 66. The first "E" means 14, and the .E666666 is APPROXIMATELY equal to .9. It is actual less than .9 by .000000066666666...forever. Since the number is not quite 14.9, multiplying by 10 gives not quite 149. And taking the INT of not-quite-149 gives the CORRECT answer of 148.

CORRECT, but not what you WANTED or EXPECTED. Right, Ethan? That is why you will find business software written in Applesoft is full of little fudge factors. We always need to multiply by enough 10's to make all pennies into integers, and then round up, and then truncate.

An alternative is to use DECIMAL arithmetic. And guess what: the 6502 has built-in decimal arithmetic. The only trouble is that Applesoft does not know about it.

I wrote an Applesoft extension package called DPFP which gives Applesoft 21-digit precision, rather than the normal 9. But it is still binary, so you still get those round-off and truncation problems with clearcut decimal fractions. About two and a half years ago I wrote another Applesoft extension package called DP18. This one is DECIMAL, and gives 18-digit precision. Bobby Deen helped me flesh it out with full support for arithmetic expressions and all the math functions.

Well, it has been hiding on my shelf long enough! I am going to start publishing it in AAL, a piece at a time. In this issue you will find the routines for addition and subtraction.

First a word about the way DP18 stores numbers. Since Applesoft uses five bytes for each floating point value, and since it is relatively easy to connect to Applesoft using multiples of five bytes, I use ten bytes for each DP18 value. The first byte holds the sign and exponent for the value. The remaining nine bytes hold 18 decimal digits, in BCD format. That is, each digit takes four bits.
The first bit of the first byte is the sign bit. Zero means plus, one means minus. If the whole first byte is zero, the whole number is zero. The remaining seven bits of the first byte are the decimal exponent, excess $40. The value $40 means ten to the zero power. $41 means 10, $42 means 100, and so on. $3F means .1, $3E means .01, and so on. Thus the exponent range is from $01 through $7F, meaning from 10^-63 through 10^64.

The mantissa bytes are considered to be a decimal fraction. The number is stored so that the most significant digit is always in the first nybble of the first byte, and the exponent is adjusted accordingly. Let's look at a few examples:

```
       42 14 90 00 00 00 00 00 00 00  =  14.9
       41 31 41 59 26 53 58 97 93 23  =  pi
       38 50 00 00 00 00 00 00 00 00  =  .000000005
       B8 50 00 00 00 00 00 00 00 00  =  -.000000005
```

Since listing the whole program at once is impossible, I have jumped right down to the lowest level so you can see how the elementary functions of addition and subtraction work. I put the origin at $0800 for this listing, but of course the final package will run wherever you assemble it for. Later we will get into I/O conversions, multiply and divide, math functions, print using, conversions between Applesoft and DP18 values, handling expressions with precedence and parentheses, and the linkage between DP18 and Applesoft.

The listing shown below has two main entry points, DSUB and DADD. You can guess what they mean! The two values to be operated on will already be unpacked into DAC and ARG by the time DSUB or DADD is called. Note that there is one extra byte for each accumulator, so that series of calculations will carry around an extra two digits of precision to avoid rounding errors. Unpacking a value into DAC involves storing the exponent byte in DAC.SIGN and then stripping the sign bit from DAC.EXPONENT.

DSUB and DADD both begin with the easiest cases, in which at least one of the values is zero. DSUB complements the value in DAC by merely toggling the sign bit, and then falls into DADD. In other words, ARG-DAC is the same as ARG+(-DAC).

DADD then determines which of the two values has the larger exponent. If necessary, it swaps ARG and DAC: the object is to have the value with the larger exponent in DAC (unless they are the same). Then the value in ARG is shifted right N digits, where N is the difference in the exponents. This what our teachers called "lining up the decimal points".

The subroutine which shifts ARG right N digits is rather smart. First, it will just fill ARG with zeros if the shift is 20 or more. Next, if the shift count is odd, it shifts right one digit position, or four bits. Then it does a direct move to shift the rest of the digits by N/2 bytes, and fills in with zero bytes on the left.

Addition is divided into two cases: either both arguments have the same sign, or they are different. If they are both the same, a simple addition loop is used. If the result carries into the next digit, DAC is shifted right one digit and a "1" is installed in the leftmost digit.

Otherwise, ARG is subtracted from DAC. If both ARG and DAC had the same exponents, it is possible that the value in ARG is larger than the value in DAC. In this case the subtracion loop will end with a "borrow" status, so the result needs to be complemented. I complement by subtracting from zero. Note that the three loops just described are all performed with the 6502 in decimal mode (the SED opcode at line 1490). CLD later reverts back to binary mode. After the mantissas are combined, the result may have one or more zero digits on the left. Therefore we go to a NORMALIZE subroutine.

NORMALIZE shifts the mantissa left until a non-zero digit is in the leftmost digit position. It also decrements the exponent for each digit-shift. I tried to do the shifting involved as intelligently as possible.

```
=============================================================================
DOCUMENT :AAL-8405:Articles:Front.Page.txt
=============================================================================
```

$1.80

Volume 4 -- Issue 8                         May, 1984

In This Issue...



This month we are beginning a series of articles describing a double-precision decimal arithmetic package for Applesoft. Imagine 18-digit arithmetic with none of the screwy rounding errors we are used to seeing in Applesoft's binary arithmetic.

You will also find quick looks at the new Apple //c and a forthcoming set of revised ROMs for the //e. We finally have the solution to a three-year-old mystery! You old-timers might remember that in August of 1981 we published a peculiar little "what does this code do?" item from John Broderick. Well he has revealed answer at long last.

Oops!

There are a couple of bugs in the Intellec Hex Converter we published last month. To correct the program you should delete line 2240 (the INY) and add a LDA #0 at line 2285. That will take care of it! Our thanks to Chaim Palman, of Calcomp, for pointing out the problems.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
================================================================================
DOCUMENT :AAL-8405:Articles:My.Ad.txt
================================================================================

S-C Macro Assembler Version 1.0.......................................$80
S-C Macro Assembler Version 1.1....................................$92.50
Version 1.1 Update.................................................$12.50
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code)........................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks...............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984

OBJ.APWRT][F (Don Lancaster, Synergetics).............................$30
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $35)  $30
Routine Machine (Southwestern Data Systems)............(reg. $64.95)  $60
FLASH! Integer BASIC Compiler (Laumer Research).......................$79
Fontrix (Data Transforms).............................................$75
Aztec C Compiler System (Manx Software)..................(reg. $199) $180
IACcalc Spreadsheet Program............................(reg. $84.95)  $75
          The one we use every day.  It's better than Visicalc!
Locksmith 5.0 (Omega MicroWare)........................(reg. $99.95)  $90

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                           or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ...........................$20

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler.................($22.95)  $21
    "Understanding the Apple II", Sather...................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster...........($15.95)  $15
```

```
              Second edition, with //e information.
    "Incredible Secret Money Machine", Lancaster............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Bag of Tricks", Worth & Lechner, with diskette.........($39.95)  $36
    "Assembly Lines: The Book", Roger Wagner................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($24.95)  $23
    "What's Where Guide" (updates first edition).............($9.95)   $9
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($17.95)  $17
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

==========================================================================
DOCUMENT :AAL-8405:Articles:New.IIe.ROMs.txt
==========================================================================

Apple //e ROM Revision.....................Bob Sander-Cederlof

Dated March 21, 1984, I received a pair of 2764 ROMs and 12-page writeup. These are preliminary versions of a new set predicted to be in general distribution by early next year.

The new //e ROMs are substantially better than the current ones. Changes include:

Applesoft: modified to work in 80-column mode, and with lower case.

Monitor:

* modified to work with new Mouse ICON characters;
* modified to accept lowercase input;
* location $1F no longer used;
* miniassembler is back;
* search command added;
* IRQ handling substantially modified.

Video Firmware (after PR#3):

* fixed many bugs;
* no more jagged scrolling, now smooth and 30% faster;
* two new escape commands to enable/disable printing of control characters;
* SETVID ($FE93) now turns off 80-column mode;
* escape-R removed.

The new IRQ handler should finally make interrupts actually usable on the Apple. The old problem with location $45 is fixed. The settings of the various soft-switches which control memory mapping are saved and the machine is put into a cononical state. The standard IRQ return sequence will restore the interrupted state of all those switches.

The total overhead from IRQ-event to your IRQ-subroutine will run from 250 to 300 microseconds, depending on the soft-switch settings. If you are in a ProDOS environment, you will have to add all the overhead caused by ProDOS.

Of course, there will be new problems. ProDOS bent over backwards in a very strange way to solve the $45 problem with interrupts. Now that it is not necessary, ProDOS should be changed. But it can't be changed for the new and still work in the old, so.... The new IRQ and BRK handler also clobbers locations $100 and $101, which is BAD! Both those locations are used by Applesoft and many other programs!

If you think these changes will impact your work, or want to be involved in shaking out bugs, you might contact Developer Relations at Apple (408) 996-1010 and discuss the Certified Apple Developer program. I think it is because I am one of those that I received this material.

```
==============================================================================
DOCUMENT :AAL-8405:Articles:Random.Numbers.txt
==============================================================================
```

Random Numbers for Applesoft...............Bob Sander-Cederlof

The RND function in Applesoft is faulty, and many periodicals have loudly proclaimed its faults. "Call APPLE", Jan 83, pages 29-34, tells them in "RND is Fatally Flawed", and presents an alternative routine which can be called with the USR function.

First, the flaws: 1) the initialization code fails to preset all five bytes of the seed value (only the first four of five are loaded); 2) the RND code uses a poor algorithm, and depends on "tweaks" to make the numbers more random; 3) the RND code does not properly implement the algorithm it appears to be aiming at.

BAD INITIALIZATION. The initialization code is at $F150 in the Applesoft ROMs. This loop moves the CHRGET subroutine down to $B1-C8, and is also supposed to copy the random number seed into $C9-CD. The last byte does not get copied, due to a bug. Changing $F151 from $1C to $1D would fix it. Most of us don't really care about this bug, because we are trying to get random numbers for games and the like, and the more random the better: not copying the last byte could make the numbers generated a little more random from one run to the next. However, some applications in simulation programs require REPEATABLE sequences of random numbers, so the effect of model changes can be seen independent of the random number generator.

POOR ALGORITHM. Most generators use an algorithm which makes the next random number by multiplying the previous one by a constant, and adding another constant. The result is reduced by dividing by a third constant and saving the remainder as the next random number. More on this later. The proper choice of the three constants is critical. I am not sure whether the Applesoft authors just made poor choices, or whether the bugs mentioned below drove them to tweaking. Tweaking the generated value is often thought to produce even more random results. In fact, according to authorities like Donald Knuth, they almost always ruin the generator. Applesoft tweaks the generated value by reversing the middle two bytes of the 32-bit value. Guess what: it ruins the generator, assuming it was good to start with.

BUGGY ALGORITHM. The congruency algorithm described in words above will only work properly when integer arithmetic is used. Applesoft uses floating point arithmetic. Further, Applesoft arithmetic routines expect five-byte operands. For some reason the constants used in RND are only four bytes long each. It appears that the exponents may have been omitted, in the expectation that integer arithmetic was going to be used. You can see the code for RND at $EFAE.

If you want to see some non-random features using RND, type in and RUN the following program:

```
10 HGR:HCOLOR=3
20 X=RND(1)*280:Y=RND(1)*160
30 HPLOT X,Y
40 GO TO 20
```

You will see the Hi-Res screen being sprinkled with dots. After about seven minutes, but long before the screen is full, new dots stop appearing. RND has looped, and is replotting the same sequence of numbers. Another test disclosed that the repetition starts at the 37,758th "random" number.

Mathematicians have developed many sophisticated tests for random number generators, but Applesoft fails even these simple ones! Depending on the starting value, you can get the Applesoft generator in a loop. You never get anywhere near the theoretically possible 4 billion different values.

The Call APPLE article proposes a new algorithm. It comes with impressive claims and credentials, but I have not found it to be better than a properly implemented congruential algorithm. The algorithm multiplies the previous seed by 8192, and takes the remainder after dividing by 67099547. This is a congruency algorithm:

```
X(n+1) = ( a * X(n) + c ) mod m

with a=8192, c=0, m=67099547
```

I re-implemented the Call APPLE algorithm, and my listing follows. The Call APPLE version would not quite fit in page 3, but mine does with a little room to spare. I also dug into some other references and came up with another algorithm, from Knuth. It is also a congruency, but with a=314159269, c=907633386, and m=2^32. This turns out to be easier to compute, and according to Knuth it should be "better". "Better" is in quotes because it is really hard to pin down what are the most important properties. Anyway this one should have very good characteristics.

The RND function does three different things, depending on the argument. You write something like R=RND(X). If X=0, you get the same number as the previous use of RND produced. If X<0, the absolute value of X becomes the new seed value. This allows you to control the sequence when you wish, and also to randomize it somewhat by using a "random" seed. If X>0, you get the next random number. The value will always be a positive number less than 1. If you want to generate a number in a range, you multiply by the width of the range and add the starting value. For example, to generate a random integer between 1 and 10:

```
R = INT( RND(1)*10 ) + 1
```

The programs I have written build a little on the options available with RND. They all begin with a little routine which hooks in the USR vector. After executing this, you can write R=USR(X), in other words substitute USR(X) anywhere you would have used RND(X). But I have added, following the Call APPLE article, the option to automatically generate integers in a range based at 0. If 0<X<2, you will get the next random fraction. If X is 2 or greater than 2, you will get a random integer between 0 and X-1. Thus you can make a random integer between 1 and 10 like this:

```
       R = USR(10) + 1
```

as well as with:

```
       R = INT (USR(1)*10) + 1
```

I wrote a third program which makes a 16-bit random value. This one uses the seed at $4E and $4F which the Apple increments continuously whenever the standard monitor input loop is waiting for an input keystroke. Integer BASIC uses this seed, and as a result is quite valuable in writing games. My new program gives you all the options stated above, and is significantly quicker than any of the others. It uses a=19125, c=13843, and m=2^16 in a standard congruency algorithm.

If you are seriously interested in random numbers, you need to read and study Donald Knuth. Volume 2 of his series "The Art of Computer Programming" is called "Seminumerical Algorithms". Chapter 3, pages 1-160, is all about random numbers. (There is only one other chapter in this volume, all about arithmetic in nearly 300 pages!) Knuth started the series back in the 60's, with the goal of seven volumes covering most of what programmers do. He finished the first three by 1972, went back and revised the first one, and then evidently got sidetracked into typesetting (several books around a typesetting language he calls "Tex").

Speaking of being sidetracked...!

Knuth ends his chapter with a list of four rules for selecting a, c, and m for congruency algorithms. Let me summarize those rules here:

1. The number m is conveniently taken as the word size. In Applesoft, the floating point mantissa is 32 bits; hence, I chose m=2^32.

2. If m is a power of 2 (and mine is), pick "a" so that "a mod 8 = 5". This, together with the rules on choosing c below, ensure that all m values will produced before the series repeats.

3. Pick "a" between m/100 and m-sqrt(m). The binary digits should NOT have a simple, regular pattern. Knuth recommends taking some haphazard constant, such as a=3131492621.

4. "c" should be odd, and preferable near "m*.2113248654".

Now for the program listings.

The first listing is for my rendition of Call APPLE's algorithm. Lines 1220-1280 link in the USR vector. Lines 1370-1450 branch according to the value of the argument of the USR function. If the argument is negative, lines 1550-1620 set up its absolute value as the new seed. If the argument is zero, the old seed is used without change, lines 1420-1450. If positive non-zero, lines 1470-1490 set up the argument as the RANGE.

Lines 1640-1690 calculate the new seed, which will be 8192 times the old seed, modulo 67099547. 8192 is 2^13, so we can multiply be 13 left shifts. After each shift, if the result is bigger than 67099547, we subtract that value and keep the remainder. The final result will be some number smaller than 67099547.

Lines 1700-1770 save the new seed, and then divide it by 67099547 to get a fraction for the USR function result. Lines 1780-1860 check the initial argument to see if you wanted a fraction between 0 and 1, or an integer between 0 and arg-1. If the latter, the fraction is multiplied by the range and reduced to an integer.

The subroutine named MODULO subtracts 67099547 from the seed value if it would leave a positive remainder, and then renormalizes the result into floating point.

Line 2270 defines the initial seed after loading the program to be 1.0. If you want some other seed, change this line or be sure to seed it with R=USR(-seed) in your Applesoft program.

<<<<listing of S.USRND S-C>>>>>

The second listing is for my 32-bit algorithm based on Knuth's rules. Again, lines 1210-1270 set up the USR linkage. Lines 1360-1400 decide what kind of argument has been used. If negative, lines 1470-1590 prepare a new seed value. If zero, the previous value is re-used. If positive, the argument is the range.

In this version the seed is maintained as a 32-bit integer. Lines 1470-1590 convert from the floating point form of the argument in FAC to the integer form in SEED. If the argument happens to be bigger than 2^32, I simply force the exponent to 2^32.

Lines 1600-1690 form the next seed by multiplying by 314159269 and adding 907633386. The calculation is done in a somewhat tricky way. Essentially it involves loading 907633386 into the product register, and then adding the partial products of 314159269*seed to that register. The tricks allow me to do all that with a minimum of program and variable space, and I hope with plenty of speed. I understood it all this morning, but it is starting to get hazy now. If you really need a detailed explanation, call me some day. The modulo 2^32 part is automatic, because bits beyond 32 are thrown away.

Lines 1700-1780 load the seed value into FAC and convert it to a floating point fraction.

Lines 1790-1870 check the range requested. If less than 2, the fraction is returned as the USR result. If 2 or more, the fraction is multiplied by the range and integerized.

<<<S.RANDOM KNUTH here>>>

The third listing is cut down from the second one, to produce a 16-bit random number. The code is very similar to the program above, so I will not describe it line-by-line. If you want an optimized version of this, the multiply especially could be shortened.

<<<S.RANDOM KEYIN>>>

What do you do if you want even more randomness than you can get from one generator? You can use two together. The best way (for greatest randomness) is to use one to select values from a table produced by the other. First generate, say 50 or 100, random values with one generator. The generate a random value with the second generator and use it to pick one of the 50 or 100 values. That picked value is the first number to use. Then replace the picked value with a new value from the first generator. Pick another value randomly using the second generator, and so on. This is analogous to two people working together. The first person picks a bowlful at random from the universe. The second person picks items one at a time from the bowl. The first person keeps randomly picking from the universe to replace the items removed from the bowl by the second person.

You could use the 16-bit generator to pick values from a "bowl" kept full by my 32-bit generator.

Now back to those tests mentioned at the beginning. I am happy to report that all three of the algorithms listed above completely fill the hi-res screen, no holes left, eventually.

By the way, the August 1981 AAL contained an article about the Integer BASIC RND function, and how to use it from assembly language.

```
==============================================================================
DOCUMENT :AAL-8405:Articles:S.IIc.65C02.txt
==============================================================================
```

65C02 vs the older Apples...................Bob Sander-Cederlof

A few months ago we reported that apparently 2-MHz versions of the 65C02 chip worked in Apple IIs and II Plusses. (Even 1-MHz versions work in //e's.) Bob Stout was our source: he tried it, it worked, and he told us so.

Based on Bob's good luck, Stephen Bach tried it, it did not work, and he told us so. Steve and Bob got together, and it seems that the 2-MHz parts work in some IIs and II Plusses, but not all. "Try it and see" seems to be the only definitive answer.

By the way, you can get the 65C02 from Hamilton/Avnet and several other distributors for under $15 each. The 1MHz version is under $10 from Western Design Center. There is no incentive for dealers to get into the distribution of chips like this, because quantity price breaks depend on volumes in the thousands.

If you are having trouble finding a distributor, call Rockwell International's sales office; they might sell to you directly, point you to a distributor, or even give you a free sample. If not Rockwell, then try GTE or NCR, who also manufacture the 65C02, albeit without the extra 32 instructions Rockwell inserted. Here are some phone numbers for Rockwell:

```
       California:  (714) 833-4655
       Texas:       (214) 996-6500
       Illinois:    (312) 297-8862
       New Jersey:  (609) 596-0090
       Tokyo:       (03)  265-8806
       West Germany:(089) 857-6016
       England:     (01)  759-9911
```

You might possibly find these chips at Apple dealers or repair centers in the near future, because it is being used in the Apple //c. Apple is apparently not using the Rockwell version, because the BYTE article about the //c says the chip has 27 new opcodes. This is the total count of new opcodes including the new addressing modes added by the 65C02 offered by NCR, GTE, Western Design, and others. The Rockwell version adds an additonal 32. Those 32 are NOT in the 65802 or 65816, so chasing after them will lead you into dead-end streets.

If you are able to wait, the 65802 and 65816 far surpass the 65C02. You can order samples from Western Design Center, (602)962-4545, at $95 each. Originally expected in January, they are now targeting June 15th.

The Apple //c..............................Bob Sander-Cederlof

In August 1977 I walked into CompuShop with checkbook in hand, hoping to fill a void in my life by (finally) buying my own personal computer. I didn't know one brand from another, but there was a 4K Apple II running a color demo in lo-res graphics that caught my eye. I bought it. My toy, because I certainly could think of no possible way to consider it more than a toy. The serial number is 219, and I am using it to write this article. By the way, the other brands that were at CompuShop in 1977 are now all out of business.

The price for 4K was $1298; I got 4K extra RAM and paid $1348 plus sales tax. No software. No CRT. No floating point BASIC. No slick manuals. About 45 pages of mimeographed notes was the total documentation package. I had to build a modulator kit that afternoon so I could hook it up to my TV set. The only other connection which seemed of any use was the cassette tape, which several hundred of you may remember. The store gave me a cassette containing the color demo and Woz's Breakout game. That was all there was! Eight empty slots, and absolutely nothing on the market to plug into them. Not even enough memory for hi-res graphics, which I did not even know existed. Absolutely no software for sale from any vendor.

I have spent a lot of time on this Apple. And money. And it is not JUST a toy any more! It has Applesoft on the motherboard, with 48K RAM. Slot 0 has an STB 128K RAM card (the best, in my opinion). All the other slots are full, but with what depends on the work for the day.

Now there is the Apple //c. $1295 buys you 128K RAM, Applesoft BASIC, a disk drive, and ProDOS! Probably over 10,000 programs on the market which will run in it, and many more to come. Built-in interfaces including two serial ports, mouse, disk controller, 80-columns, many video options, and more. The most often purchased interfaces are all there, enough to fill five slots in an older Apple. They added a headphone jack and volume control, too; it is recessed under the left edge. Using it will let you work later at night without disturbing light sleepers. You still get a "game" port, but it is a 9-pin D-socket and doubles as the mouse port. Sorry, no more Cassette port. A second disk drive can be added, and it costs significantly less than a second //e drive.

There are two new switches beside the RESET switch, labeled 40/80 and Keyboard. The first switches between 40 and 80 columns. The second selects QWERTY or Dvorak keyboard arrangement. Think a while of the implications to future generations of including THAT switch. The 40/80 switch is really just connected to what used to be cassette input $C060. You can read the switch position like the firmware does, by looking at the sign bit of that byte.

Until now all Apple game ports had four analog inputs, four switch outputs, and three switch inputs. The //c has only two analog inputs, and no switch outputs. The three switch inputs remain, with switch two dedicated to the mouse button. The other two analog input addresses are used as single bits to read the mouse X and Y direction. The four output bits are now used to control various interrrupt modes.

An interesting new softswitch input is at $C077. If bit 7 of the byte is 1, the current line being stroked on the screen is graphics; if 0, it is text. People like Bob Bishop, Don Lancaster, and Bill Budge probably already have figured out fantastic new tricks using this bit.

The power supply is now in a little box that is part of the power cord. 115 volts AC in, 12 volts DC out. The rest of the supply voltages derived inside the case. There will be a battery pack option later. And how about an adapter for running in the car?

The video output capability is phenomenal. Now you get all the American and European options built in. One connector gives you the NTSC we are all used to. Another gives you RF-modulated form for an American TV set. You also get RGB and various European standards. The 15-pin video connector also gives you an audio signal and various timing signals.

The ROM in the //c is VERY different. The differences include serial port and mouse firmware, better interrupt handling, the improvements made in the new //e ROMs, no more self-test program, and extensions to the disassembler (monitor L-command) for the 65C02 chip.

It is getting to be quite a chore for software to distinguish which kind of Apple II it is in. Here is a chart showing Apple's official ID bytes:

| $FBB3 | $FB1E | $FBC0 | Environment |
|---|---|---|---|
| $38 | | | Old (Original) Apple ][ |
| $EA | $AD | | Apple ][ Plus Autostart |
| $EA | $8A | | Apple /// Emulation |
| $06 | | $EA | Apple //e |
| $06 | | $E0 | New Apple //e ROM |
| $06 | | $00 | Apple //c |

Interrupts are used extensively by the mouse firmware. A keyboard interrupt plus firmware implements a 128-character type-ahead buffer.

All this talk about mouse support leads me to make one clarification. You don't get a mouse unless you pay an extra $100. The firmware and interface are built-in, but the actual device is optional.

By the way, besides the 16 memory chips there are and only 21 other chips. More special chips, including IWM (Integrated Woz Machine, the disk controller); GLU (General Logical Unit); and TMG (Timing Generator). Compare the total 37 chips with about 50 in the Macintosh, and more than 90 in the IBM PCjr. Most of the chips are soldered in, but a few still sit in sockets.

==============================================================================
DOCUMENT :AAL-8405:Articles:That.Code.Did.txt
==============================================================================

What That Code Did.........................Bob Sander-Cederlof

Way back in August 1981 I published a short article by John Broderick titled "What Does This Code Do?" Well, John never did tell us. But in the May 1984 Nibble, page 115, he finally has let the cat out of the bag. I think this article has probably been banging around the Nibble office for some time now, because John hasn't done anything with Apple's in quite a while. He developed a super fast accounting program in Apple II assembly language, then re-wrote the whole thing for the Sage 68000-based system. Last I heard he was in the IBM world.

The code he gave us three years ago was five bytes long:

```
             BRK
             PLA
             PLA
             PLA
             RTS
```

As published in Nibble, it is a little longer:

```
       BREAK BRK
             NOP
             PLA
             PLA
             JSR $FF3F
             RTS
```

Boiling it all down, John used this code during debugging sessions. By putting a JSR to the 8-byte program he can effect a clean breakpoint. Clean, in that he can use the monitor "G" command to continue execution after the BRK.

When JSR BREAK is executed, the BRK opcode will send Apple into the monitor and display the five registers. Their contents will have been saved at $45 thru $49. The address of the first PLA will also be saved. Typing the monitor "G" command will continue execution at that PLA. The two PLA's will pop off the return address the G command put on the stack, leaving it as it was before the BRK. The JSR $FF3F will restore the A-register, which the two PLA's clobbered. The the RTS will return right after the JSR BREAK which started this paragraph.

The original five-byte version was both confusing and erroneous. Confusing, because the PLA immediately after the BRK is never executed. BRK seems like a two-byte opcode to the 6502, so the saved address skips over the following byte. Erroneous, because the A-register has been changed by the time the RTS is executed. I think I would amend both of his versions to this:

```
BREAK  BRK
       NOP
       PLA
       PLA
       LDA $45
       RTS
```

```
===============================================================================
DOCUMENT :AAL-8405:Articles:Wagner.News.txt
===============================================================================
```

Some Interesting News

Roger Wagner:  well known to most of us as owner of Southwestern Data Systems, author of "Assembly Lines:  The Book", author of several popular programs in early Apple days, speaker at AppleFests, and so forth.  Roger is branching out.

Last month he incorporated and changed the name of SDS to Roger Wagner Publishing.  Along with the name change, the product packaging has been changed.  After a poll of dealers, they decided to replace the plush padded binders with a new package design which allows customers to browse through the manuals, while the diskette and other package contents are securely kept intact.  No more shrink wrap!  Simpler packaging is less expensive, so the prices of some products have been lowered.  And one step further, even more significant:  no more copy protection!

We applaud Roger for taking this step.  As I remember it, Roger was one of the first publishers to use any kind of software protection, back in the 70's.  His scheme included a program on the master disk which allowed you to make a limited number of back up copies.  Now Roger joins us and a handful of other publishers who refuse to shackle users with protected software.

Roger has also joined forces with Val Golding (founder and long-time editor of Call-A.P.P.L.E.) to form Emerald City Publishing, Inc. Their first project is "The Apple's Apprentice", a magazine aimed at Apple-teens.

================================================================================
DOCUMENT :AAL-8405:DOS3.3:ANOTHER.TEST.txt
================================================================================

R–’(…1)M–40:ÜC(M)"ó=-R–’(M):N–N»1:ñ1:¢1:∫N;M(C(R)–C(R)»1d2X –
”(RÀ10):Y–R…X 10|<¢Y»2:ñ10 X»1:∫C(R);ÑF´30

```
=================================================================
DOCUMENT :AAL-8405:DOS3.3:Lic.Plate.Game.txt
=================================================================
```

```
N–0:ÜNC(279):ó$T–0:ÅL–0¡25D-≠L–’(26)ŒL–’(26)ŒL–’(26)ƒ50R(T–
T»1:´30u2∫Á(L»65);:Ç:T–TÀ2:≠Tœ279ƒT–279¨<NC(T)–NC(T)»1:N–N»1:∫" -- "N"
"T:≠,(…16384)—128ƒ20,Fπ…16368,0:M–0:F–279:L–0:ÅJ–0¡279:≠NC(J)œMƒM –
NC(J)˜K≠NC(J)œ0ÕJœLƒL–J
```

```
P≠NC(J)œ0ÕJ—FƒF–J!   ZÇJ:ë:π49234,0:í34   _ì0,191¡279,191Z
dÅJ–F¡L:ìJ,191 (1…NC(J)ÀM)¡J,191:Çq n≠,(…16384)—128ƒ110Ö
xπ…16368,0:â:´20
```

```
==============================================================================
DOCUMENT :AAL-8405:DOS3.3:More.Rnd.Tests.txt
==============================================================================
```

```
ë:í3:π49234,0*ì’(280),’(192):´204dà:†15Jnç’(40),’(40):´110V»R1–
’(1)u“N–N»1:ñ1:∫N;:≠’(1)—œR1ƒ210Ñ‹∫:∫R1,’(0)
```

```
==========================================================================
DOCUMENT :AAL-8405:DOS3.3:S.DIFFERENCES.txt
==========================================================================
```

```
*SAVE S.DIFFERENCES
*--------------------------------
*       DISPLAY MAP OF DIFFERENCES
*       IN TWO MEMORY REGIONS
*
*       ADR1<ADR2.ADR3^Y
*
*--------------------------------
A1      .EQ $3C,3D
A4      .EQ $42,43
*--------------------------------
MON.NXTA4  .EQ $FCB4
MON.PRA1   .EQ $FD92
MON.COUT   .EQ $FDED
*--------------------------------
SETUP   LDA #DIFFERENCES
        STA $3F9
        LDA /DIFFERENCES
        STA $3FA
        RTS
*--------------------------------
DIFFERENCES
        JSR MON.PRA1      PRINT CR, ADDRESS AND "-"
.1      LDY #0            COMPARE TWO BYTES
        LDA (A1),Y
        CMP (A4),Y
        BEQ .2            SAME, SELECT FIRST CHAR
        INY               DIFF, SELECT 2ND CHAR
.2      LDA CHARS,Y       GET DISPLAY CHAR
        JSR MON.COUT      PRINT SAME OR DIFF CHAR
        JSR MON.NXTA4     NEXT ADDRESS AND TEST
        BCS .3            ...FINISHED
        LDA A1            CHECK FOR FULL LINE
        AND #$1F          OF 32
        BNE .1            ...FULL YET
        BEQ DIFFERENCES   ...FULL
.3      RTS
*--------------------------------
CHARS   .AS -/.*/         SAME AND DIFF CHARS
*--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8405:DOS3.3:S.DP18.ADD.SUB.txt
==========================================================================

 1000   .LIF
 1010  *SAVE S.DP18 ADD & SUB
 1020  *---------------------------------
 1030  *      18-DIGIT DECIMAL FLOATING POINT
 1040  *      ADDITION AND SUBTRACTION
 1044  *---------------------------------
 1046  AS.OVRFLW  .EQ $E8D5
 1050  *---------------------------------
 1060  DAC           .BS 12
 1070  DAC.EXPONENT  .EQ DAC
 1080  DAC.HI        .EQ DAC+1
 1090  DAC.EXTENSION .EQ DAC+10
 1100  DAC.SIGN      .EQ DAC+11
 1110  *---------------------------------
 1120  ARG           .BS 12
 1130  ARG.EXPONENT  .EQ ARG
 1140  ARG.HI        .EQ ARG+1
 1150  ARG.EXTENSION .EQ ARG+10
 1160  ARG.SIGN      .EQ ARG+11
 1170  *---------------------------------
 1180  SWAP.ARG.DAC
 1190         LDY #11      SWAP 12 BYTES
 1200  .1     LDA ARG,Y
 1210         LDX DAC,Y
 1220         STA DAC,Y
 1230         TXA
 1240         STA ARG,Y
 1250         DEY
 1260         BPL .1
 1270         RTS
 1280  *---------------------------------
 1290  *      SUBTRACT DAC FROM ARG
 1300  * DAC = ARG - DAC
 1310  *---------------------------------
 1320  DSUB   LDA DAC.EXPONENT
 1330         BEQ SWAP.ARG.DAC  ARG-0=ARG
 1340         LDA DAC.SIGN
 1350         EOR #$80
 1360         STA DAC.SIGN
 1370  *---------------------------------
 1380  *      ADD ARG TO DAC
 1390  * DAC = ARG + DAC
 1400  *---------------------------------
 1410  DADD   LDA ARG.EXPONENT
 1420         BEQ .3       DAC+0=DAC
 1430  .1     SEC          COMPARE EXPONENTS
 1440         LDA DAC.EXPONENT
 1450         BEQ SWAP.ARG.DAC  ARG+0=ARG
 1460         SBC ARG.EXPONENT
```

```
       BMI .8       ARG IS LARGER
       JSR SHIFT.ARG.RIGHT.N
       SED          SET DECIMAL MODE
       LDA DAC.SIGN COMPARE SIGNS
       EOR ARG.SIGN
       BMI .4       OPPOSITE SIGNS
*---SAME SIGNS------------------
       CLC          SAME SIGNS, JUST ADD VALUES
       LDY #9       TEN BYTES
.2     LDA DAC.HI,Y
       ADC ARG.HI,Y
       STA DAC.HI,Y
       DEY
       BPL .2
       CLD          BINARY MODE
       BCC .3       NO CARRY
       JSR SHIFT.DAC.RIGHT.ONE
       LDA DAC.HI
       ORA #$10
       STA DAC.HI
.3     RTS
*---DIFFERENT SIGNS-------------
.4     SEC          SUBTRACT ARG FROM FAC
       LDY #9       TEN BYTES
.5     LDA DAC.HI,Y
       SBC ARG.HI,Y
       STA DAC.HI,Y
       DEY
       BPL .5
       BCS .7       NO BORROW
       SEC          BORROW, SO COMPLEMENT
       LDY #9
.6     LDA #0
       SBC DAC.HI,Y
       STA DAC.HI,Y
       DEY
       BPL .6
       LDA ARG.SIGN
       STA DAC.SIGN
.7     CLD
       JMP NORMALIZE.DAC
*---SWAP ARG & DAC, TRY AGAIN----
.8     JSR SWAP.ARG.DAC
       JMP .1
*--------------------------------
*      SHIFT DAC RIGHT ONE DECIMAL DIGIT
*--------------------------------
SHIFT.DAC.RIGHT.ONE
       INC DAC.EXPONENT
       BMI .2
       LDY #4       4 BITS RIGHT
.1     LSR DAC.HI
       ROR DAC.HI+1
       ROR DAC.HI+2
```

```
2000        ROR DAC.HI+3
2010        ROR DAC.HI+4
2020        ROR DAC.HI+5
2030        ROR DAC.HI+6
2040        ROR DAC.HI+7
2050        ROR DAC.HI+8
2060        ROR DAC.HI+9 EXTENSION
2070        DEY
2080        BNE .1
2090        RTS
2095 .2     JMP AS.OVRFLW
2100 *--------------------------------
2110 *      SHIFT ARG RIGHT N DIGITS
2120 *--------------------------------
2130 SHIFT.ARG.RIGHT.N
2140        LDY #9       SET UP FOR 10 BYTES
2150        CMP #20      DON'T BOTHER IF OFF END
2160        BCS .4       JUST ENTER ZERO INTO ARG
2170        LSR          TEST SHIFT COUNT ODD OR EVEN
2180        BCC .2       EVEN
2190        JSR SHIFT.ARG.RIGHT.ONE
2200 .2     TAY          # BYTES TO SHIFT
2210        BEQ .6       NONE
2220        EOR #$FF     -(#BYTES+1)
2230        CLC
2240        ADC #10      9-#BYTES
2250        TAX
2260        LDY #9
2270 .3     LDA ARG.HI,X
2280        STA ARG.HI,Y
2290        DEY
2300        DEX
2310        BPL .3
2320 .4     LDA #0
2330 .5     STA ARG.HI,Y
2340        DEY
2350        BPL .5
2360 .6     RTS
2370 *--------------------------------
2380 *      NORMALIZE VALUE IN DAC
2390 *--------------------------------
2400 NORMALIZE.DAC
2410        LDY #-1
2420 .1     INY          NEXT BYTE
2430        CPY #10
2440        BCS .7       ...NO MORE BYTES
2450        LDA DAC.HI,Y
2460        BEQ .1       ...STILL ZEROES
2500 *--------------------------------
2510 .2     TYA          TEST BYTE COUNT
2520        BEQ .5       FIRST BYTE IS NON-ZERO
2530        LDX #0       POINT X AT FIRST BYTE
2540 .3     LDA DAC.HI,Y
2550        STA DAC.HI,X
```

```
        INX
        INY
        CPY #10
        BCC .3
*--------------------------------
        LDA #0       FILL REST OF DAC WITH ZEROES
.4      STA DAC.HI,X
        DEC DAC.EXPONENT  ADJUST EXPONENT
        DEC DAC.EXPONENT  FOR SHIFT DISTANCE
        INX
        CPX #10
        BCC .4
*--------------------------------
.5      LDA DAC.HI   SEE IF NEED ONE-DIGIT SHIFT
        AND #$F0
        BNE .6       NO NYBBLE SHIFT NEEDED
        DEC DAC.EXPONENT
        JSR SHIFT.DAC.LEFT.ONE
.6      LDA DAC.EXPONENT
        BPL .8
.7      LDA #0
        STA DAC.EXPONENT
        STA DAC.SIGN
.8      RTS
*--------------------------------
SHIFT.DAC.LEFT.ONE
        LDY #4
.1      ASL DAC.EXTENSION
        ROL DAC.HI+8
        ROL DAC.HI+7
        ROL DAC.HI+6
        ROL DAC.HI+5
        ROL DAC.HI+4
        ROL DAC.HI+3
        ROL DAC.HI+2
        ROL DAC.HI+1
        ROL DAC.HI
        DEY
        BNE .1
        RTS
*--------------------------------
*       SHIFT ARG RIGHT ONE DECIMAL DIGIT
*--------------------------------
SHIFT.ARG.RIGHT.ONE
        LDY #4
.1      LSR ARG.HI
        ROR ARG.HI+1
        ROR ARG.HI+2
        ROR ARG.HI+3
        ROR ARG.HI+4
        ROR ARG.HI+5
        ROR ARG.HI+6
        ROR ARG.HI+7
        ROR ARG.HI+8
```

```
3050        ROR ARG.HI+9  EXTENSION
3060        DEY
3070        BNE .1
3080        RTS
3090  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8405:DOS3.3:S.RANDOM.KEYIN.txt
==============================================================================
```

```
 1000  *--------------------------------
 1010  *SAVE S.RANDOM KEYIN
 1020  *--------------------------------
 1030  *      ALLOWS ACCESS TO THE KEYIN RANDOM VALUE
 1040  *--------------------------------
 1050         .OR $300
 1060         .TF B.RANDOM KEYIN
 1070  *--------------------------------
 1080  NORMALIZE.FAC       .EQ $E82E
 1090  FMUL.FAC.BY.YA      .EQ $E97F
 1100  STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
 1110  AS.QINT             .EQ $EBF2
 1120  AS.INT              .EQ $EC23
 1130  *--------------------------------
 1140  USER.VECTOR         .EQ $0A THRU $0C
 1150  FAC                 .EQ $9D THRU $A2
 1160  FAC.SIGN            .EQ $A2
 1170  FAC.EXTENSION       .EQ $AC
 1180  KEY.SEED            .EQ $4E,4F
 1190  *--------------------------------
 1200  LINK   LDA #$4C     "JMP" OPCODE
 1210         STA USER.VECTOR
 1220         LDA #RANDOM
 1230         STA USER.VECTOR+1
 1240         LDA /RANDOM
 1250         STA USER.VECTOR+2
 1260         RTS
 1270  *--------------------------------
 1280  *      R = USR (X)
 1290  *      IF X < 0 THEN RESEED WITH ABS(X)
 1300  *      IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
 1310  *      IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
 1320  *                     0 <= R < 1
 1330  *      IF X >= 2 THEN R = INT(RND*X)
 1340  *--------------------------------
 1350  RANDOM
 1360         LDA FAC.SIGN CHECK FOR RESEEDING
 1370         BMI .1       ...YES
 1380         LDA FAC      CHECK FOR X=0
 1390         BEQ .6       ...YES, REUSE LAST NUMBER
 1400  *---X --> RANGE-----------------
 1410         LDX #RANGE
 1420         LDY /RANGE
 1430         JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
 1440         JMP .4
 1450  *---PREPARE SEED----------------
 1460  .1     LDA #0       MAKE SEED POSITIVE
 1470         STA FAC.SIGN
 1480         LDA FAC      LIMIT SEED TO 2^16-1
```

```
1490        CMP #$90
1500        BCC .2
1510        LDA #$90
1520        STA FAC
1530 .2     JSR AS.QINT   $EBF2
1540        LDA FAC+3
1550        STA KEY.SEED
1560        LDA FAC+4
1570        STA KEY.SEED+1
1580 *---SEED*19125+13843-------------
1590 .4     LDX #0
1600 .5     LDA KEY.SEED,X
1610        STA MULTIPLIER
1620        LDA C,X
1630        STA KEY.SEED,X
1640        JSR MULTIPLY
1650        INX
1660        CPX #2
1670        BCC .5
1680 *---LOAD SEED INTO FAC-----------
1690 .6     LDA #0
1700        STA FAC+3
1710        STA FAC+4
1720        STA FAC.SIGN
1730        STA FAC.EXTENSION
1740        LDA #$80
1750        STA FAC
1760        LDA KEY.SEED
1770        STA FAC+1
1780        LDA KEY.SEED+1
1790        STA FAC+2
1800        JSR NORMALIZE.FAC
1810 *---SCALE TEST-------------------
1820        LDA RANGE
1830        CMP #$82     IS RANGE BETWEEN ZERO AND ONE?
1840        BCC .8       ...YES
1850 *---SCALE------------------------
1860        LDA #RANGE
1870        LDY /RANGE
1880        JSR FMUL.FAC.BY.YA   $E97F
1890        JSR AS.INT  $EC23
1900 *---RETURN-----------------------
1910 .8     RTS
1920 *--------------------------------
1930 MULTIPLY
1940        STX BYTE.CNT
1950        LDY #1
1960 .1     LDA A,Y
1970        STA MULTIPLICAND,X
1980        DEY
1990        DEX
2000        BPL .1
2010        LDY #8
2020        BNE .2       ...ALWAYS
```

```
2030  *--------------------------------
2040  .5     CLC          DOUBLE THE MULTIPLICAND
2050  .6     ROL MULTIPLICAND,X
2060         DEX
2070         BPL .6
2080  .2     LSR MULTIPLIER
2090         BCC .4
2100         LDX BYTE.CNT
2110         CLC
2120  .3     LDA MULTIPLICAND,X
2130         ADC KEY.SEED,X
2140         STA KEY.SEED,X
2150         DEX
2160         BPL .3
2170  .4     LDX BYTE.CNT
2180         DEY
2190         BNE .5
2200         RTS
2210  *--------------------------------
2220  RANGE          .HS 81.00000000
2230  A              .DA /19125,#19125
2240  C              .DA /13843,#13843
2250  MULTIPLIER     .BS 1
2260  MULTIPLICAND   .BS 2
2270  BYTE.CNT       .BS 1
2280  *--------------------------------
```

```
===================================================================
DOCUMENT :AAL-8405:DOS3.3:S.RANDOM.KNUTH.txt
===================================================================

 1000  *--------------------------------
 1010  *SAVE S.RANDOM KNUTH
 1020  *--------------------------------
 1030  *      FROM KNUTH'S "THE ART OF COMPUTER PROGRAMMING"
 1040  *                   VOLUME 2, PAGES 155-157.
 1050  *--------------------------------
 1060         .OR $300
 1070         .TF B.RANDOM KNUTH
 1080  *--------------------------------
 1090  NORMALIZE.FAC       .EQ $E82E
 1100  FMUL.FAC.BY.YA      .EQ $E97F
 1110  STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
 1120  AS.QINT             .EQ $EBF2
 1130  AS.INT              .EQ $EC23
 1140  *--------------------------------
 1150  USER.VECTOR         .EQ $0A THRU $0C
 1160  FAC                 .EQ $9D THRU $A2
 1170  FAC.SIGN            .EQ $A2
 1180  FAC.EXTENSION       .EQ $AC
 1190  AS.SEED             .EQ $CA THRU $CD
 1200  *--------------------------------
 1210  LINK   LDA #$4C     "JMP" OPCODE
 1220         STA USER.VECTOR
 1230         LDA #RANDOM
 1240         STA USER.VECTOR+1
 1250         LDA /RANDOM
 1260         STA USER.VECTOR+2
 1270         RTS
 1280  *--------------------------------
 1290  *      R = USR (X)
 1300  *      IF X < 0 THEN RESEED WITH ABS(X)
 1310  *      IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
 1320  *      IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
 1330  *                    0 <= R < 1
 1340  *      IF X >= 2 THEN R = INT(RND*X)
 1350  *--------------------------------
 1360  RANDOM
 1370         LDA FAC.SIGN CHECK FOR RESEEDING
 1380         BMI .1       ...YES
 1390         LDA FAC      CHECK FOR X=0
 1400         BEQ .6       ...YES, REUSE LAST NUMBER
 1410  *---X --> RANGE------------------
 1420         LDX #RANGE
 1430         LDY /RANGE
 1440         JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
 1450         JMP .4
 1460  *---PREPARE SEED-----------------
 1470  .1     LDA #0       MAKE SEED POSITIVE
 1480         STA FAC.SIGN
```

```
       LDA FAC      LIMIT SEED TO 2^32-1
       CMP #$A0
       BCC .2
       LDA #$A0
       STA FAC
.2     JSR AS.QINT   $EBF2
       LDX #3       COPY FAC INTO SEED
.3     LDA FAC+1,X
       STA SEED,X
       DEX
       BPL .3
*---SEED*314159269+907633386-----
.4     LDX #0
.5     LDA SEED,X
       STA MULTIPLIER
       LDA C,X
       STA SEED,X
       JSR MULTIPLY
       INX
       CPX #4
       BCC .5
*---LOAD SEED INTO FAC-----------
.6     LDX #5
.7     LDA FLT.SEED,X
       STA FAC,X
       DEX
       BPL .7
       LDA #0
       STA FAC.EXTENSION
       JSR NORMALIZE.FAC
*---SCALE TEST-------------------
       LDA RANGE
       CMP #$82     IS RANGE BETWEEN ZERO AND ONE?
       BCC .8       ...YES
*---SCALE------------------------
       LDA #RANGE
       LDY /RANGE
       JSR FMUL.FAC.BY.YA   $E97F
       JSR AS.INT  $EC23
*---RETURN-----------------------
.8     RTS
*--------------------------------
MULTIPLY
       STX BYTE.CNT
       LDY #3
.1     LDA A,Y
       STA MULTIPLICAND,X
       DEY
       DEX
       BPL .1
       LDY #8
       BNE .2       ...ALWAYS
*--------------------------------
.5     CLC          DOUBLE THE MULTIPLICAND
```

```
2030  .6     ROL MULTIPLICAND,X
2040         DEX
2050         BPL .6
2060  .2     LSR MULTIPLIER
2070         BCC .4
2080         LDX BYTE.CNT
2090         CLC
2100  .3     LDA MULTIPLICAND,X
2110         ADC SEED,X
2120         STA SEED,X
2130         DEX
2140         BPL .3
2150  .4     LDX BYTE.CNT
2160         DEY
2170         BNE .5
2180         RTS
2190  *--------------------------------
2200  RANGE          .HS 81.00000000
2210  FLT.SEED       .HS 80
2220  SEED           .HS 00.00.00.00
2230                 .HS 00         SIGN
2240  A              .HS 12.B9.B0.A5   314159269
2250  C              .HS 36.19.62.EB   907633386
2260  MULTIPLIER     .BS 1
2270  MULTIPLICAND   .BS 4
2280  BYTE.CNT       .BS 1
2290  *--------------------------------
```

```
======================================================================
DOCUMENT :AAL-8405:DOS3.3:S.USRND.S.C.txt
======================================================================

 1000  *--------------------------------
 1010  *SAVE S.USRND S-C
 1020  *--------------------------------
 1030  *       FROM CALL APPLE, JAN 1983, PAGE 29-34
 1040  *--------------------------------
 1050         .OR $300
 1060         .TF B.USRND
 1070  *--------------------------------
 1080  NORMALIZE.FAC       .EQ $E82E
 1090  FMUL.FAC.BY.YA      .EQ $E97F
 1100  LOAD.ARG.FROM.YA    .EQ $E9E3
 1110  FDIV.ARG.BY.YA      .EQ $EA5C
 1120  LOAD.FAC.FROM.YA    .EQ $EAF9
 1130  STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
 1140  COPY.FAC.TO.ARG     .EQ $EB66
 1150  AS.INT              .EQ $EC23
 1160  *--------------------------------
 1170  USER.VECTOR         .EQ $0A THRU $0C
 1180  FAC                 .EQ $9D THRU $A2
 1190  FAC.SIGN            .EQ $A2
 1200  CNTR                .EQ $A5
 1210  *--------------------------------
 1220  LINK   LDA #$4C     "JMP" OPCODE
 1230         STA USER.VECTOR
 1240         LDA #RANDOM
 1250         STA USER.VECTOR+1
 1260         LDA /RANDOM
 1270         STA USER.VECTOR+2
 1280         RTS
 1290  *--------------------------------
 1300  *      R = USR (X)
 1310  *      IF X < 0 THEN RESEED WITH ABS(X)
 1320  *      IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
 1330  *      IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
 1340  *                    0 <= R < 1
 1350  *      IF X >= 2 THEN R = INT(RND*X)
 1360  *--------------------------------
 1370  RANDOM
 1380         LDA FAC.SIGN CHECK FOR RESEEDING
 1390         BMI .2       ...YES
 1400         LDA FAC      CHECK FOR X=0
 1410         BNE .1       ...NO, X=RANGE
 1420         LDA #SEED
 1430         LDY /SEED
 1440         JSR LOAD.ARG.FROM.YA
 1450         JMP .5
 1460  *---X --> RANGE-----------------
 1470  .1     LDX #RANGE
 1480         LDY /RANGE
```

```
       JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
*---SEED --> FAC-----------------
       LDA #SEED
       LDY /SEED
       JSR LOAD.FAC.FROM.YA  $EAF9
*---PREPARE SEED-----------------
.2     LDA #0       MAKE SEED POSITIVE
       STA FAC.SIGN
       LDA FAC      LIMIT SEED TO 67099547
       CMP #$9A
       BCC .3
       LDA #$9A
       STA FAC
       JSR MODULO
*---(8192*SEED) MOD 67099547-----
.3     LDA #13
       STA CNTR
.4     INC FAC
       JSR MODULO
       DEC CNTR
       BNE .4
*---SEED/67099547----------------
       LDX #SEED
       LDY /SEED
       JSR STORE.FAC.AT.YX.ROUNDED
       JSR COPY.FAC.TO.ARG  $EB66
.5     LDA #FLT67
       LDY /FLT67
       JSR FDIV.ARG.BY.YA $EA5C
*---SCALE TEST-------------------
       LDA RANGE
       CMP #$82     IS RANGE BETWEEN ZERO AND ONE?
       BCC .6       ...YES
*---SCALE------------------------
       LDA #RANGE
       LDY /RANGE
       JSR FMUL.FAC.BY.YA   $E97F
       JSR AS.INT  $EC23
*---RETURN-----------------------
.6     RTS
*--------------------------------
MODULO
       LDY #0
       LDA FAC
       CMP #$9A
       BCC .3       < 67099547
       BEQ .1       67099547...
       LDY #4
.1     SEC
       LDA FAC+4    LSB
       SBC MAN67+3,Y
       PHA
       LDA FAC+3
       SBC MAN67+2,Y
```

```
2030        PHA
2040        LDA FAC+2
2050        SBC MAN67+1,Y
2060        PHA
2070        LDA FAC+1
2080        SBC MAN67+0,Y
2090        PHA
2100        BCC .2       <67099547
2110        PLA
2120        STA FAC+1
2130        PLA
2140        STA FAC+2
2150        PLA
2160        STA FAC+3
2170        PLA
2180        STA FAC+4
2190        JMP NORMALIZE.FAC  $E82E
2200 .2     PLA
2210        PLA
2220        PLA
2230        PLA
2240 .3     RTS
2250 *--------------------------------
2260 RANGE  .HS 81.00000000
2270 SEED   .HS 81.00000000
2280 FLT67  .HS 9A.7FF6E6C0    = 67,099,547
2290 MAN67  .HS FFF6E6C0
2300        .HS 7FFB7360
2310 *--------------------------------
```

```
===============================================================================
DOCUMENT :AAL-8405:DOS3.3:TEST.USRND.txt
===============================================================================
```

```
ê:í3:X–’(…1))ì’(280),’(192):´20
```

==============================================================================
DOCUMENT :AAL-8406:Articles:Andromeda.Board.txt
==============================================================================

Fixing the Andromeda 16K Card.....................Bob Bernard

In the April 1984 Call-APPLE there was a letter from John Wallace regarding a problem with the Andromeda 16K RAM card. As this card was the second on the market, first after Apple's (which was bundled with Pascal), there are probably still tens of thousands in use. Yet the Andromeda is anathema to some hardware and software.

In particular, it played havoc with John Wallace's copy of Apple PIE (a popular word processor from yesteryear), and my Lobo 8" floppy drive controller (another relic, I suppose). Bob S-C tells of running into the problem too:

"I have an Andromeda board, and I ran into this problem with early versions of ES-CAPE. Using a STA (or other store) opcode to any soft switches on the Andromeda card write-protected the card. Using two stores in a row to try to write-enable the card does no good either. I had to change all stores to loads or BITs to make it work. Apple's board accepts either stores or loads, as do all other memory cards I have tested."

There are probably lots of interfaces and programs out there which stumble over Andromeda. Wallace details a hardware modification to the Andromeda board which makes it work the same as all other memory boards. I found a slightly simpler way, and I recommend that all Andromeda owners fix their boards as soon as possible.

Remove the 74LS08 chip at board location U13. Bend pin 10 out so that it sticks straight out, and plug the chip back into its socket so that pin 10 is on the outside. Solder a small wire to pin 10 (carefully), and solder the other end of the wire to pin 14 of the same chip. Or, you can solder to a solder pad pin 14 is connected to, as shown in the drawing below. (Pin 14 is connected to Vcc, the +5 volts line.) That's all there is to it.

John Wallace suggests using a 1K resistor rather than a wire, but I found a wire is sufficient.

With the wire installed, both reads and writes can be used to switch the card, just like Apple intended it.

```
=====================================================================
DOCUMENT :AAL-8406:Articles:Barkovitch.Mntn.txt
=====================================================================
```

The Barkovitch Utilities

Did you notice Dave Barkovitch's ad last month? He has written a very handy set of utilities for us serious Applers, and sells 'em cheap! Be prepared to puzzle your way through his admittedly skimpy documentation, but it is all there.

The I/O Tracer comes in EPROM on a little card that plugs into any slot 1-7 for only $40.50 (including shipping). I/O Tracer is essentially a powerful disk ZAP utility, allowing you to read/write/edit any DOS 3.3 sector. You see an entire sector at once on the screen, in either hex or ASCII, along with all status information.

Dave's Single-Step Trace program will help you debug assembly language. He likes it better than the other commercial varieties of debuggers, and sells it for only $35.

Any questions, call Dave at (201) 499-0636.

==============================================================================
DOCUMENT :AAL-8406:Articles:CRC.Bad.Bit.txt
==============================================================================

Finding the Erroneous Bit Using CRC.................Bruce Love
Hamilton, New Zealand

The April 1984 AAL article about using Cyclic Redundancy Codes posed the question, "How do you find out which bit was in error, assuming only one was wrong?" I found a way.

My algorithm assumes that there was one and only one bit wrong in the entire 258-byte message (256 bytes of original message plus 2 bytes of CRC). The bits are numbered left-to-right, or most significant bit of first byte received through the least significant bit of the CRC, 0 through $80F (or 2063, if you prefer decimal).

After receiving the data and CRC, the RECV program has computed a composite CRC and the result will be $0000 if there were no errors. If the result is non-zero, it uniquely determines which bit was wrong. Here is a summary of my algorithm for finding which bit:

```
       let bit.number = 2063
       let dummy.crc  = 1
    -->if dummy.crc = crc, then we found the bit
   |   decrement bit.number
   |   shift dummy.crc left 1 bit
   |   if carry set, EOR with $1021
    ---loop
```

[ The following comments are by Bob Sander-Cederlof. ]

The program listing which follows is an addendum to the listing in the April issue of AAL. Lines 3070 through the end should be appended to the program in that issue. If you buy the AAL Quarterly Disk, it will already be there.

The sequence I used for testing the program went like this. First I assembled the whole program, April's plus the one below. Then I typed "$4000<F800.F8FFM" to move a copy of the monitor's first page into the test buffer. I thought this would be "interesting" data to play with. Then these steps:

```
       :MGO SEND       (fakes sending the buffer)
       1F45            (SEND prints out the CRC for BUFFER)
       :$4000          (see what is there)
       4A              (it was $4A)
       :$4000:CA       (make a fake error in the 1st bit)
       :MGO RECV
       xxxx            (some non-zero value)
       :MGO FIND.BAD.BIT
       0000            (the bad bit was the first bit)
       $4000:4A        (restore the correct bit
```

Then I tried the same steps on various other bit positions, with accurate results in every case.

=============================================================================
DOCUMENT :AAL-8406:Articles:DOSology.txt
=============================================================================

DOSology and DOSonomy......................Bob Sander-Cederlof

The other day I was thinking about the way Apple spells ProDOS. They jealously guard the spelling, having trademarked the idea of upper-case "P" and "DOS" with lower-case "ro".

Of course, we all know that "Pro" is a standard prefix, with origins in the Greek language. In Greek it means "before". I think Apple derived it from the English word "professional", so that ProDOS means "professional DOS". Nevertheless, the "pro" even in the word professional means before, according to the etymologies in dictionaries.

I took some Greek courses at Dallas Theological Seminary back in 1973 and 1974. I remember very little now, but one thing stuck with me: prepositions. "Pro" is one, but there are a lot more. What other interesting DOSses can we invent?

By the way, the preferred pronunciation of DOS rhymes with "boss", not "gross". If you insist on rhyming with the latter, your pronunciation has a decided Spanish accent. For you we have invented "UnoDOS", which is of course two-thirds of a popular product on the IBM-PC, uno-dos-tres by Lotus. Ha!

The first that came to mind was "ParaDOS". We like it so well, we've decided to trademark it! It could relate to either paradox or pair-of-dice or paradise, take your pick. A shrewdly written DOS could appear as all three at different times to different people.

Bill and I then started to brainstorm, and we can't stop. We've got a blackboard full of neat names, just waiting for some one to write code for. We may have stumbled on to some previously-used names, like SoliDOS and ProntoDOS, but for the most part I think we have cornered the market.

```
AmbiDOS   MisoDOS   PhiloDOS  BiblioDOS ViviDOS   DiaDOS
PaleoDOS  MesoDOS   NeoDOS    PsychoDOS MoriDOS   Dial-a-DOS
ChromoDOS BlancoDOS TechniDOS SomatoDOS DulciDOS  AnoDOS
AcriDOS   FeloniDOS BaloniDOS FormiDOS  MiniDOS   CathoDOS
MicroDOS  MidiDOS   MilliDOS  MegaDOS   NanoDOS   VagaDOS
TeraDOS   UniDOS    BioDOS    StupiDOS  TorriDOS  FabriDOS
SemiDOS   PeriDOS   AntiDOS   AnteDOS   ProsDOS   ExoDOS
HypoDOS   HyperDOS  OvaDOS    PupaDOS   PropoDOS  EnDOS
ArcheDOS  StatiDOS  DynamoDOS DynaDOS   ProtoDOS  EschatoDOS
OsteoDOS  MultiDOS  PuroDOS   CardioDOS PyroDOS   PrimaDOS
FrigiDOS  InterDOS  AndroDOS  GynoDOS   GymnoDOS  PseudoDOS
HieroDOS  SpiroDOS  HelioDOS  CycloDOS  AutoDOS   AggreDOS
ManoDOS   ChiroDOS  PetroDOS  LithoDOS  AeroDOS   PosiDOS
PlanoDOS  LiquiDOS  MarbleDOS PedoDOS   GraviDOS  NegaDOS
```

```
PedaDOS   GeriaDOS  NutriDOS  FlexiDOS  PleniDOS  NecroDOS
VisiDOS   InvisiDOS FluoriDOS FloriDOS  FaunaDOS  PensaDOS
ThanaDOS  AgriDOS   NaviDOS   NovaDOS   SpuriDOS  MensaDOS
StereoDOS VerbiDOS  VermiDOS  CineDOS   GeoDOS    TragiDOS
MonoDOS   DuoDOS    CobraDOS  FerroDOS  OxyDOS    AfroDOS
EuroDOS   NippoDOS  FrancoDOS IndoDOS   CanaDOS   HispanoDOS
```

Get the idea?

=============================================================
DOCUMENT :AAL-8406:Articles:DP18.Part.2.txt
=============================================================

18-Digit Arithmetic, Part 2................Bob Sander-Cederlof

Feedback on installment one of this series came from as far away as Sweden. Paul Schlyter, with others, pointed out the omission of three very important letters. PRINT (14.9*10) indeed prints 149, as expected. What I meant to say was that PRINT INT(14.9*10) prints 148.

I noticed another error at the top of page 21. The exponent range runs from 10^-63 thru 10^63, not 10^64.

Paul pointed out that my routines did not check for underflow and overflow. I did have such checks in another part of the code, as yet unlisted, but I now agree with him that some checks belong in the routines printed last month.

The subroutine SHIFT.DAC.RIGHT.ONE is called when a carry beyond the most significant bit is detected in DADD, at line 1620. If the exponent is already 10^63, or $7F, this shift right will cause overflow. That means the sum formed by DADD is greater than 10^63, and we need to do either of two things. My usual choice, assuming the routines are being used from Applesoft, is to JMP directly to the Applesoft ROM overflow error routine, at $E8D5. Another option is to set the DAC exponent to $7F, and the mantissa to all 9's. To implement it my way, add these lines:

```
       1945       BMI .2

       2085 .2    JMP $E8D5
```

Underflow needs to be tested in the NORMALIZE.DAC subroutine. Underlofw happens when the exponent falls below 10^-63. The normal procedure upon underflow is to set the result to zero. Zero values in DP18 are indicated by the exponent being zero, regardless of the mantissa value. Delete lines 2400-2480 and line 2730, and enter the following lines

```
       2400        LDY #-1
       2410 .1     INY
       2420        CPY #10
       2430        BCS .7
       2440        LDA DAC.HI,Y
       2450        BEQ .1

       2730 .6     LDA DAC.EXPONENT
       2731        BPL .8
       2732 .7     LDA #0
       2733        STA DAC.EXPONENT
       2734        STA DAC.SIGN
       2735 .8     RTS
```

All these changes will be installed on Quarterly Disk 15.

This month I want to present several pack and unpack subroutines, and one which rounds the value in DAC according to the value in the extension byte.

Note that I have just LISTed the subroutines below, rather than showing the assembly listing, because the program parts need to all be assembled together before they are meaningful.

There are two "unpack" subroutines, MOVE.YA.DAC and MOVE.YA.ARG. They perform the "load accumulator" function. There is one "pack" subroutine, MOVE.DAC.YA, which performs the "store accumulator" function.

The MOVE routines use a page-zero pair at $5E and $5F. Assuming the DP18 package will be called from Applesoft via the &-vector, there will be no page-zero conflicts here.

The subroutines DADD and DSUB from last month, and DMULT and DDIV to come, all expect two arguments in DAC and ARG and leave the result in DAC. Assuming there are two packed DP18 value at VAL.A and VAL.B, and that I want to add them together and store the result in VAL.C, I would do it this way:

```
       LDA #VAL.A
       LDY /VAL.A
       JSR MOVE.YA.DAC
       LDA #VAL.B
       LDY /VAL.B
       JSR MOVE.YA.ARG
       JSR DADD
       LDA #VAL.C
       LDY /VAL.C
       JSR MOVE.DAC.YA
```

Note that MOVE.DAC.YA calls ROUND.DAC before storing the result. ROUND.DAC checks the extension byte. If the extension byte has a value less than $50, no rounding need be done. If it is $50 through $99, the value in DAC should be rounded up. If the higher digits are less than .999999999999999999, then there will be no carry beyond the most significant digit, and no chance for overflow. However, if it is all 9's we will get a final carry and we will need to change the number to 100000000000000000 and add one to the exponent. In tiny precision, this is like rounding .995 up to 1.00. If the exponent was already 10^63, rounding up with a final carry causes overflow, so I jump to the Applesoft error handler.

<<<<  MOVE listings here>>>>

None of the pack/unpack code is especially tricky, but the same cannot be said for DMULT. Multiplication is handled "just like you do it with pencil and paper", but making it happen at all efficiently makes things look very tricky.

Call DMULT after loading the multiplier and multiplicand into DAC and ARG (doesn't matter which is which, because multiplication is commutative). Then JSR DMULT to perform the multiply. The result will be left in DAC.

Looking at the DMULT code, lines 1040-1070 handle the special cases of either argument being 0. Anything times zero is zero, and zero values are indicated by the exponent being zero, so this is real easy.

Lines 1090-1130 clear a temporary register which is 20 bytes long. This register will be used to accumulate the partial products. Just in case some of the terminology is losing you, here are some definitions:

```
       12345  <-- multiplicand
     x 54321  <-- multiplier
   ---------
       12345  <-- 1st partial product
      24690   <-- 2nd partial product
     37035    <-- 3rd    "       "
    49380     <-- 4th    "       "
   61725      <-- 5th    "       "
   ---------
   670592745  <-- product
```

Lines 1150-1180 form the 20-digit product of the two 10-digit arguments. I wanted to reduce the number of times the individual digits have to be isolated, or the accumulators shifted by 4-bits, so I used a trick. Line 1150 calls a subroutine which multiplies the multiplicand (in ARG) by all the low-order digits in each byte of the multiplier (in DAC). In other words, I accumulate only the odd partial products at this time. Then I shift DAC 4-bits right, which places the other set of digits in the low-order side of each byte. I also have to shift the result register, MAC, right 4-bits, and then I call the MULTIPLY.BY.LOW.DIGITS subroutine again.

Lines 1200-1270 form the new exponent, which is the sum of the exponents of the two arguments. Since both exponents have the value $40 added to make them appear positive, one of the $40's has to be subtracted back out. But before that, if the sum is above $C0 then we have an overflow condition. After subtracting out one of the $40's, if the result is negative we have an underflow condition. Note that since the carry status was clear at line 1250, I subtracted $3F; for one more byte, I could have done it the normal way and used SEC, SBC #$40.

Lines 1290-1310 form the sign of the product, which is the exclusive-or of the signs of the two arguments. Lines 1330-1370 copy the most significant 10 bytes of the product from MAC to DAC.

The result may have a leading zero digit in the left half of the first byte, so I call NORMALIZE.DAC at line 1390. If The leading digit was zero, normalizing will shift DAC left one digit position, leaving room for another significant digit on the right end. Lines 1400-1490 handle installing the extra digit if necessary.

MULTIPLY.BY.LOW.DIGITS picks up the low-order digit out of each byte of the multiplier, one-by-one, and calls MULTIPLY.ARG.BY.N.

MULTIPLY.ARG.BY.N does the nitty-gritty multiplication. And here is where I lost all my ingenuity, too. The multiplier digit is stored in DIGIT, and used to count down a loop which adds ARG to MAC DIGIT times. Surely this can be done more efficiently! How about it Paul? Or Charlie? Anyone?

Well, that's all for this month. Next month expect some simple I/O routines and the divide subroutine.

================================================================
DOCUMENT :AAL-8406:Articles:Front.Page.txt
================================================================

$1.80

Volume 4 -- Issue 9                              June, 1984

In This Issue...



More on ProDOS and Nonstandard Apples

In the March issue we published Bob Stout's note on how to make ProDOS boot in a Franklin computer. The current issue, (No. 9) of Hardcore Computist points out that the address given in that note didn't work for the ProDOS version dated 1-JAN-84. Apparently Bob was referring to an earlier version. The correct address for the NOPs is $265B.

In a similar vein, inside this issue Jan Eugenides points out that ProDOS will also fail in an Apple with a modified Monitor ROM. He then gives a slightly different patch to defeat the check code.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8406:Articles:LancastersStuff.txt
==============================================================================

OBJ.APWRT][F updated to AWIIe Toolkit...........Don Lancaster

I have packed even more goodies on eight disk sides, combining the HACKER and USER packages into one powerful Toolkit. The price is only slightly higher... They were $29.50 each, now only $39.50 together.

Now that we have yet another Apple monitor, vastly different yet purportedly compatible, guess what! Applewriter //e is not QUITE compatible with the //c. Surprise, surprise! The status line display gets turned into garbage. One of the patches included in the new AWIIe Toolkit solves the problem admirably. This AWIIE CLARIFIER Applesloth program modifies your Applewriter IIe backup diskettes to eliminate trashing of the IIc status display line. Here it is now, more than slightly compressed for AAL, to tease you into getting the whole Toolkit:

```
100  REM *---------------------------------------*
200  REM *  COPYRIGHT 1984 BY DON LANCASTER AND  *
220  REM *  SYNERGETICS,  BOX 1300, THATCHER AZ  *
240  REM *  85552      Phone:  (602) 428-4073    *
260  REM *   ALL  COMMERCIAL RIGHTS RESERVED     *
280  REM *---------------------------------------*
380  TEXT : HOME : HIMEM: 8000
400  HTAB 8: PRINT "Applewriter IIe CLARifier": PRINT
600  REM Check Validity
660  PRINT CHR$(4)"BLOAD OBJ.APWRT][F,A$2300
670  IF  PEEK (14815) <  > 188 THEN 880
680  IF  PEEK (15052) <  > 41 THEN 880
690  IF  PEEK (15096) <  > 59 THEN 880
695  REM Install Patches
700  POKE 14815,60: POKE 14816,36: POKE 14817,207:
      POKE 14818,16: POKE 14819,2: POKE 14820,169:
      POKE 14821,62
710  POKE 15052,208: POKE 15053,42
720  POKE 15062,96
730  POKE 15096,41: POKE 15097,127: POKE 15098,201:
      POKE 15099,96: POKE 15100,176: POKE 15101,208:
      POKE 15102,201: POKE 15103,64
740  POKE 15104,144: POKE 15105,204: POKE 15106,41:
      POKE 15107,63: POKE 15108,176: POKE 15109,200
750  PRINT CHR$(4)"UNLOCK OBJ.APWRT][F"
760  PRINT CHR$(4)"BSAVE OBJ.APWRT][F,A$2300,L$30D3"
770  PRINT CHR$(4)"LOCK OBJ.APWRT][F"
870  PRINT "IT WORKED!" : END
880  PRINT "Will not verify as AWIIe; patch ABORTED" : END
```

Gotchas: Fixes only the status line. Rare and brief changes in the flashing cursor symbol will remain.

==============================================================================
DOCUMENT :AAL-8406:Articles:Making65C02Work.txt
==============================================================================

Making a 65C02 Work in my Apple II Plus........William O'Ryan

I am writing this on my Apple II Plus running a 2 MHz 65C02 (GTE G65SC02PI-2). All is well now, but it took some doing.

A few days after pluggin in the chip I started noticing problems. Applesoft found itself unable to process numeric literals, and the version of FORTH I have been developing began acting weird.

Following the tip in AAL that the timing of the fetch-process- save instructions might be responsible, I ran some tests on them. The 65C02 worked flawlessly. Apparently the problem is elsewhere.

After further checking, especially in my FORTH, I found that a certain BNE instruction sitting in the first byte of a page and branching backward into the prior page frequently branched back one byte less than it should.

I'm not a hardware person, but I figured debugging is debugging and I really wanted that chip to work, so I began staring at the circuit diagram in the Apple Reference manual. After several hours I concluded that I stood for input, O for output, D for data, and A for address.

The easiest hypothesis to check seemed to be that data was not getting back from the RAMs to the microprocessor in time. So I wrote down some chip numbers and went downtown to see if I could buy some faster varients. Well, the first two chips I replaced solved the problem.

They were 74LS257 chips at B6 and B7. These chips multiplex the output of RAM with the output of the keyboard and send the result to the 65C02. I replaced them with 74F257 chips. I understand these consume less power, respond faster, and are more susceptible to electrostatic damage.

Anyway, my 65C02 is happy now. I would like to hear whether this modification works in other Apples, and with other 65C02s. Drop a line to Bob and Bill at S-C if you have any word on this.

==============================================================================
DOCUMENT :AAL-8406:Articles:More.Rnd.Stuff.txt
==============================================================================

More Random Number Generators..............Bob Sander-Cederlof

I published my "Random Numbers for Applesoft" article last month just in time. The June issue of Micro includes a 9.5 page article called "A Better Random Number Generator", by H. Cem Kaner and John R. Vokey. The article reports on work funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors give an excellent overview of the various methods used to test random number generators, and the methods they used during their seven years of research to produce three "better" generators. It is worth buying a copy of Micro to get a copy of this article.

The authors used the same linear congruential algorithm I discussed last month, but with different parameters. My favorite last month was:

```
       R(new) = ( R(old) * A + C ) mod 2^32
       where A = 314159269
         and C = 907633386
```

Kaner and Vokey decided to use a 40-bit seed and use mod 2^40 in calculating each successive value. After extensive analysis and testing, they came up with three generators based on the following values for "A" and "C":

```
       RNG 1:  A = 31415938565
               C = 26407

       RNG 2:  A = 8413453205
               C = 99991

       RNG 3:  A = 27182819621
               C = 3
```

There are an unusually large number of typos in the article, and some of them are hard to decipher. The value 26407 above was written in the comment field as 24607; however, in the hexadecimal constant assembly code it was 0000006727, which is 26407. Even worse, in the listing there are missing lines of code and missing characters here and there. All of the immediate mode instructions are missing a "#" character. Four or five labels are never defined in the listing.

Since the program as printed cannot possibly be successfully loaded, assembled, POKEd, or executed, I have chosen to re-write it for inclusion here, after my own style. I believe my version is also significantly improved in coding and speed.

The reason given for choosing to work with 40 bits rather than 32, even though Applesoft only stores 32 and using 40 takes longer, was that it is important to provide more values between 0.0 and 2^-32. I tend to disagree on the importance of this, since most uses of random numbers on the Apple are for games, and simulate such simple things as dealing cards or throwing dice. Perhaps more serious simulations need more precise randomness. Of course they also increase by a factor of 256 the number of numbers generated before the sequence repeats.

Buried in the middle of the program is a well-optimized 40-bit multiplication loop. You might enjoy puzzling out how it works!

The program uses USR(x), where x selects which of the three generators to use. There is no provision for setting the seed or for selecting a range other than 0...1, such as I included in my programs last month. Some enterprising individual will marry the shell of my USR subroutine with the RNG of Kaner and Vokey to produce a really great Applesoft Random Number Generator.

==============================================================
DOCUMENT :AAL-8406:Articles:Moto.Formatter.txt
==============================================================

Converting to Motorola S-Format............Bob Sander-Cederlof

Last April I told how to convert object code to the Intellec Hex Format (AAL pages 14-18, April, 1984). Both Intel and Zilog use that format. Motorola, on the other hand, has its own format for object code. It is similar, but it is significantly different. If you are programming for a Motorola chip, or using a PROM burner that uses Motorola format, then the following program is for you.

The Motorola S-Type format has three kinds of records: header, data, and end-of-file. Each record begins with the letter "S" and ends with a carriage return linefeed (CRLF). According to the samples I have seen, all of the bytes in a record are in ASCII code with the high bit zero. (Apple peripherals tend to like the high bit = 1, so I made this an option.) The maximum length including the "S" and up to but not including the CRLF is 64 "frames". Between the "S" and CRLF, each record consists of five fields:

Record format field: ASCII 0, 1, or 9 (hex $30, $31, or $39) for header, data, or end-of-file records respectively.

Byte count field: the count expressed as two ASCII digits of the number of bytes (half the number of frames) from address field through the checksum field. The minimum is 3, and the maximum is 60 decimal or $3C hexadecimal.

Address field: four frames representing the four digits of the load address for data bytes in a data record, or the run address in an end-of-file record. All four digits will be "0" in a header record.

Data field: two hex digits for each byte of data. The number of bytes will be 3 less than the number specified in the byte count field, because that count includes two bytes for the address and one byte for the checksum.

Checksum field: two hex digits representing the 1's complement of the binary sum of all the bytes in the previous four fields.

If you compare the S-Type format with the Intellec format, you will note several differences:

* records start with "S" instead of ":"
* the fields are in a different order
* there was no header record for Intellec
* the byte count covers three fields instead of only the data field
* the checksum is computed by a different algorithm and covers different data.

I tried to use as much as possible of the Intellec program when writing the Motorola program. You will find a lot of similarities if you compare the two. Both are designed to be used with the monitor's control-Y instruction. Both expect you to enter the output slot number or address in zero-page bytes 0 and 1.

The Motorola program requires two additional pieces of information. It needs a byte at 0002 which will be either $00 or $80, denoting whether to set the high bit to 0 or 1 on every output byte. It also needs an eight character name for the header record. This should be entered in zero-page locations 0003 through 000A.

For example, assume the object code I want to format is in the Apple between $6000 and $67FF. In the target processor it will load at address $1000. The name of the program is "SAMPLE". I want to send the data with the high bit = 0. The device I want to send it to is connected to an intelligent peripheral card in slot 2. Here is what I type:

```
]BRUN B.MOTOROLA FORMATTER     (or :BRUN B.MOTOROLA FORMATTER
]CALL -151                     (or :MNTR)
*0:2 0                         (send to slot 2)
*:0                            (hi-bit = 0)
*:53 41 4D 50 4C 45 20 20      ("SAMPLE")
*1000<6000.67FF^Y              (^Y means control-Y)
```

I recommend comparing this program and my description of it with the Intellec program and article in the April AAL. Here is the Motorola formatter:

```
===================================================================
DOCUMENT :AAL-8406:Articles:My.Ad.txt
===================================================================

S-C Macro Assembler Version 1.0.......................................$80
S-C Macro Assembler Version 1.1....................................$92.50
Version 1.1 Update.................................................$12.50
Source Code for Version 1.1 (on two disk sides).......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code)........................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $35)  $30
Routine Machine (Southwestern Data Systems)............(reg. $64.95)  $60
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36
FLASH! Integer BASIC Compiler (Laumer Research).......................$79
Fontrix (Data Transforms).............................................$75
Aztec C Compiler System (Manx Software).................(reg. $199) $180

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                           or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ...........................$20

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Incredible Secret Money Machine", Lancaster............($7.95)   $7
```

```
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Assembly Lines: The Book", Roger Wagner................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($24.95)  $23
    "What's Where Guide" (updates first edition).............($9.95)   $9
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($17.95)  $17
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

==============================================================================
DOCUMENT :AAL-8406:Articles:PDos.Mod.Mtr.txt
==============================================================================

Booting ProDOS with a Modified Monitor ROM........Jan Eugenides

You may have already figured this out, but ProDOS won't boot if you have installed S. Knouse's modified ROM in your Apple. This can easily be fixed, as follows:

On track 1, sector C, change bytes B4-B6 from AE B3 FB to A2 EA EA. This tells ProDOS your machine is a II+. If it's a //e, make B5 an A0 instead.
On track 1, sector 9, change bytes 60-61 from A9 00 to A5 0C. This defeats the ROM check routine.

Ta daaa! Now ProDOS works just fine with your modified ROM.

```
===========================================================================
DOCUMENT :AAL-8406:Articles:PRT.Command.txt
===========================================================================
```

Using the PRT Command...............................Bill Morgan

New users of the S-C Macro Assembler have asked for examples of how to use some of the customizing features. For example, just now I had a call from a gentleman who needed to know how to set up the PRT vector to turn on his printer and send the special control strings it requires.

It happens that I had the same problem just a few weeks ago. I just picked up an OkiData 92 printer, which I am quite happy with, except for a couple of small warts. Setting Elite spacing (12 characters/inch, 8 lines/inch) on that printer requires these hex codes: 9C 9B B8. The catch is that 9C, which corresponds to Control-backslash. I can't type CTRL-\ on my Apple II+! Besides, by the time I type in the commands to turn on the printer, set Elite mode, and set a left margin, I have entered 15 keystrokes. That's too many for my lazy, dyslexic fingers, so I came up with a PRT command to do the whole job.

The addresses in this routine are set up for the 40-column Version 1.1 Language Card assembler. If you are using another version, check to make sure that the patch space is indeed all zeroes. All $D000 versions of the assembler have some blank space before $E000. If you are using a $1000 version, look to see if there is some space available between the end of the assembler and the beginning of the Symbol Table and set PATCH.SPACE to that address. You will also have to set PRT.VECTOR to $1009.

Here are the exact steps to use this patch:

Start the assembler.

```
$C083 C083
$D01C:0 D0 0 F8

$AA60.AA61

LOAD S.PRT

ASM

$D01C:0 0 0 0
$C080

BSAVE <assembler>,A$D000,L$XXXX
```

The $AA60.AA61 line gives you the length that you will need to use for the BSAVE command. Substitute the filename of the version you use for <assembler> in the above command.

If you are using Version 1.0 of the assembler, things are a little different. You should omit the $D01C entries in the above commands, delete lines 1090 and 1100, and add this line to the program:

```
     1125        .TA $800
```

Then, after the assembly, install the patch with $DF00<800.81EM and $D009: 4C 00 DF. These extra steps are necessary because Version 1.0 lacks the ability to override memory protection during assembly.

Lines 1270-1290 are where you should install the codes your printer needs.

```
================================================================
DOCUMENT :AAL-8406:Articles:Revisit.48.0.txt
================================================================
```

Revisiting $48:0...........................Bob Sander-Cederlof

Remember all those warnings about storing 0 in $48 after DOS had a whack at your zero page? Maybe not, but let me remind you.

Apple's monitor uses locations $45 through $49 in a very special way. Ignoring this, the writers of DOS also used them. When you start execution from the monitor (using the G, S, or T commands) The data in these locations gets loaded into the registers: $45 into A, $46 into X, $47 into Y, $48 into P (status), and $49 into S (stack pointer). When a program hits a BRK opcode, or the S command has finished executing a single opcode, the monitor saves these five registers back into $45...$49.

No serious problem, unless you like to enter the monitor and issue the G, S, or T commands. Even less of a problem, because the S and T commands were removed from the monitor ROM when the Apple II Plus came out. And if you don't care what is in the registers anyway....

But the P-register is rather special, too. One of its bits, called "D", controls how arithmetic is performed. If "D" is zero, arithmetic will be done in the normal binary way; if D=1, arithmetic is done in BCD mode. That is, adding one to $49 will produce $50 rather than $4A. If the program you are entering doesn't expect to be in decimal mode, and tries arithmetic, you will get some rather amusing results.

Hence the warning: before using the G command from the monitor, type 48:0 to be sure decimal mode is off. Later versions of DOS store 0 into $48 after calling those routines which use $48. And the monitor stores 0 into $48 whenever you hit the RESET key (or Control-RESET).

```
******************************************************
*                                                    *
*   Now I am here to tell you that storing 0 into    *
*   $48 is ALL WRONG!  It took Bill and me 5 hours   *
*   to unravel the mystery caused by storing zero    *
*   there!                                           *
*                                                    *
******************************************************
```

You should put into $48 a sensible value. Better, DOS should never use $45 through $48; if it must use them, save and restore them. There are eight bits in the P-register, and in the 6502 seven of them are important. One of them, we discovered, is VERY important.

The bit named "I" controls the IRQ interrupt. If I=1, IRQ interrupts will not be accepted. If I=0, IRQ interrupts will be accepted. So...who cares about interrupts?

Hardly anyone uses interrupts in Apple II's, because of all the hidden problems. But there are some very nice boards for the Apple that are designed to be used with interrupts. Most of them are safe, because RESET disables their interrupt generators.

Need I say that we discovered a board that does not disable the interrupt generators when you hit RESET? The Novation Cat Modem (a very excellent product) leaves at least one of its potential IRQ sources in an indeterminate state. IRQ's don't immediately show up, though, because they are trapped until you have addressed any of the soft switches on the card. But, for example, if that card is in slot 2 and I read or write any location from $C0A0 through $C0AF, IRQ's start coming. Still no problem, because I=1 in the P-register.

UNTIL WE USE THE MONITOR G COMMAND!

If I use the monitor G command, location $48, containing 0, is loaded into the P-register. Then an IRQ gets through and sends the 6502 vectoring through an unprepared vector at $3FE,3FF and BANG!

Our solution was to put SEI instructions in various routines, and to make sure that $48 contains 4, not 0, before using the G command.

From now on, whenever you hear that you need to be sure $48 contains zero, think four.

```
==========================================================================
DOCUMENT :AAL-8406:DOS3.3:S.CRCBadBidFndr.txt
==========================================================================

 1000  *SAVE S.CRC BAD BIT FINDER
 1010  *--------------------------------
 1020  BUFFER .EQ $4000
 1030  LIMIT  .EQ $4102
 1040  *--------------------------------
 1050  CRC    .EQ $00,01
 1060  PNTR   .EQ $02,03
 1070  TPTR   .EQ $04,05
 1080  TMASK  .EQ $06
 1090  SPTR   .EQ $07,08
 1100  SMASK  .EQ $09
 1110  *--------------------------------
 1120  PRNTAX .EQ $F941
 1130  CROUT  .EQ $FD8E
 1140  PRBYTE .EQ $FDDA
 1150  COUT   .EQ $FDED
 1160  *--------------------------------
 3060  *--------------------------------
 3070  *   FIND WHICH BIT IS BAD IN BUFFER+CRC
 3080  *
 3090  *      RESULT IS BIT POSITION IN MESSAGE,
 3100  *      WHERE THE FIRST BIT OF THE MESSAGE IS BIT 0
 3110  *      AND (IN THIS CASE) THE LAST CRC BIT IS BIT $80F.
 3120  *
 3130  *      ALGORITHM BY BRUCE LOVE, AUSTRIALIA.
 3140  *--------------------------------
 3150  BIT.NUMBER .EQ $10,11
 3160  DUMMY.CRC  .EQ $12,13
 3170  *--------------------------------
 3180  FIND.BAD.BIT
 3190         LDA #$80F    TOTAL # BITS - 1
 3200         STA BIT.NUMBER    (WE WILL COUNT BACKWARDS)
 3210         LDA /$80F
 3220         STA BIT.NUMBER+1
 3230         LDA #$0001   STARTING POINT FOR BIT FINDER
 3240         STA DUMMY.CRC
 3250         LDA /$0001
 3260         STA DUMMY.CRC+1
 3270  .1     LDA CRC      COMPARE RECEIVED CRC WITH
 3280         CMP DUMMY.CRC        PROCESSED VALUE;
 3290         BNE .2       IF THEY MATCH, WE HAVE FOUND THE
 3300         LDA CRC+1    BAD BIT.
 3310         CMP DUMMY.CRC+1
 3320         BEQ .4       ...FOUND BAD BIT!
 3330  .2     LDA BIT.NUMBER        DECREMENT BIT COUNTER
 3340         BNE .3
 3350         DEC BIT.NUMBER+1
 3360         BMI .5       WENT TOO FAR
 3370  .3     DEC BIT.NUMBER
```

```
3380        ASL DUMMY.CRC
3390        ROL DUMMY.CRC+1
3400        BCC .1
3410        LDA DUMMY.CRC
3420        EOR #$21
3430        STA DUMMY.CRC
3440        LDA DUMMY.CRC+1
3450        EOR #$10
3460        STA DUMMY.CRC+1
3470        JMP .1
3480 .4     LDA BIT.NUMBER+1  PRINT THE BIT NUMBER
3490        JSR PRBYTE        (IF $8000, THE ERROR WAS
3500        LDA BIT.NUMBER    NOT A SINGLE BIT)
3510        JSR PRBYTE
3520        JMP CROUT
3530 .5     BRK
3540 *--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8406:DOS3.3:S.DP18.MULTIPLY.txt
==================================================================

 1000  *SAVE S.DP18 MULTIPLY
 1010  *--------------------------------
 1020  * DAC = ARG * DAC
 1030  *--------------------------------
 1040  DMULT  LDA DAC.EXPONENT  IF DAC=0, EXIT
 1050         BEQ .3
 1060         LDA ARG.EXPONENT  IF ARG=0, SET DAC=0 AND EXIT
 1070         BEQ .4
 1080  *---CLEAR RESULT REGISTER--------
 1090         LDA #0
 1100         LDY #19
 1110  .1     STA MAC,Y
 1120         DEY
 1130         BPL .1
 1140  *---FORM PRODUCT OF FRACTIONS----
 1150         JSR MULTIPLY.BY.LOW.DIGITS
 1160         JSR SHIFT.MAC.RIGHT.ONE
 1170         JSR SHIFT.DAC.RIGHT.ONE
 1180         JSR MULTIPLY.BY.LOW.DIGITS
 1190  *---ADD THE EXPONENTS------------
 1200         LDA DAC.EXPONENT
 1210         CLC
 1220         ADC ARG.EXPONENT
 1230         CMP #$C0     CHECK FOR OVERFLOW
 1240         BCS .5       ...OVERFLOW
 1250         SBC #$3F     ADJUST OFFSET
 1260         BMI .4       ...UNDERFLOW
 1270         STA DAC.EXPONENT
 1280  *---FORM SIGN OF PRODUCT---------
 1290         LDA DAC.SIGN
 1300         EOR ARG.SIGN
 1310         STA DAC.SIGN
 1320  *---MOVE MAC TO DAC--------------
 1330         LDY #9
 1340  .2     LDA MAC,Y
 1350         STA DAC.HI,Y
 1360         DEY
 1370         BPL .2
 1380  *---NORMALIZE DAC----------------
 1390         JSR NORMALIZE.DAC
 1400         LDA MAC      IF LEADING DIGIT=0,
 1410         AND #$F0     THEN GET ANOTHER DIGIT
 1420         BNE .3
 1430         LDA MAC+10
 1440         LSR
 1450         LSR
 1460         LSR
 1470         LSR
 1480         ORA DAC.HI+9
```

```
1490        STA DAC.HI+9
1500 .3     RTS
1510 .4     LDA #0
1520        STA DAC.SIGN
1530        STA DAC.EXPONENT
1540        RTS
1550 .5     JMP AS.OVRFLW
1560 *---------------------------------
1570 *      MULTIPLY BY EVERY OTHER DIGIT
1580 *---------------------------------
1590 MULTIPLY.BY.LOW.DIGITS
1600        SED          DECIMAL MODE
1610        LDX #9
1620        LDY #19
1630 .1     LDA DAC.HI,X
1640        AND #$0F     ISOLATE NYBBLE
1650        BEQ .2       0, SO NEXT DIGIT
1660        JSR MULTIPLY.ARG.BY.N
1670 .2     DEY          NEXT MAC POSITION
1680        DEX          NEXT DAC DIGIT
1690        BPL .1       DO NEXT DIGIT
1700        CLD          BINARY MODE
1710        RTS          DONE
1720 *---------------------------------
1730 MULTIPLY.ARG.BY.N
1740        STA DIGIT    N = 1...9
1750        STY TEMP     SAVE Y
1760        STX TEMP+1   SAVE X
1770 .1     LDX #9       INDEX INTO ARG
1780        CLC
1790 .2     LDA ARG.HI,X
1800        ADC MAC,Y    ADD IT
1810        STA MAC,Y
1820        DEY          NEXT MAC
1830        DEX          NEXT ARG
1840        BPL .2       NEXT DIGIT
1850        BCC .4       NO CARRY
1860 .3     LDA #0       PROPAGATE CARRY
1870        ADC MAC,Y
1880        STA MAC,Y
1890        DEY
1900        BCS .3       MORE CARRY
1910 .4     LDY TEMP     GET POSITION IN MAC
1920 .5     DEC DIGIT    NEXT DIGIT
1930        BNE .1
1940        LDX TEMP+1
1950        RTS          DONE
1960 *---------------------------------
1970 SHIFT.MAC.RIGHT.ONE
1980        LDY #4       4 BITS RIGHT
1990 .0     LDX #1       20 BYTES
2000        LSR MAC
2010 .1     ROR MAC,X
2020        INX          NEXT BYTE
```

```
2030        PHP
2040        CPX #20
2050        BCS .2       NO MORE BYTES
2060        PLP
2070        JMP .1
2080 .2     PLP
2090        DEY          NEXT BIT
2100        BNE .0
2110        RTS
2120 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8406:DOS3.3:S.DP18.Pack.Un.txt
==============================================================================

 1000  *SAVE S.DP18 PACK & UNPACK
 1010  *-------------------------------
 1020  *     ADDRESSES INSIDE APPLESOFT
 1030  *-------------------------------
 1040  AS.OVRFLW .EQ $E8D5    OVERFLOW ERROR
 1050  *--------------------------------
 1060  *      PAGE ZERO USAGE
 1070  *-------------------------------
 1080  PNTR   .EQ $5E,5F
 1090  *-------------------------------
 1100  *      MOVE (Y,A) INTO DAC AND UNPACK
 1110  *--------------------------------
 1120  MOVE.YA.DAC
 1130         STA PNTR
 1140         STY PNTR+1
 1150         LDY #9       MOVE 10 BYTES
 1160  .1     LDA (PNTR),Y
 1170         STA DAC,Y
 1180         DEY
 1190         BPL .1
 1200         INY          Y=0
 1210         STY DAC.EXTENSION
 1220         LDA DAC.EXPONENT
 1230         STA DAC.SIGN
 1240         AND #$7F
 1250         STA DAC.EXPONENT
 1260         RTS
 1270  *--------------------------------
 1280  *      MOVE (Y,A) INTO ARG AND UNPACK
 1290  *--------------------------------
 1300  MOVE.YA.ARG
 1310         STA PNTR
 1320         STY PNTR+1
 1330         LDY #9       MOVE 10 BYTES
 1340  .1     LDA (PNTR),Y
 1350         STA ARG,Y
 1360         DEY
 1370         BPL .1
 1380         INY          Y=0
 1390         STY ARG.EXTENSION
 1400         LDA ARG.EXPONENT
 1410         STA ARG.SIGN
 1420         AND #$7F
 1430         STA ARG.EXPONENT
 1440         RTS
 1450  *--------------------------------
 1460  *      PACK AND MOVE DAC TO (Y,A)
 1470  *--------------------------------
 1480  MOVE.DAC.YA
```

```
1490        STA PNTR
1500        STY PNTR+1
1510        JSR ROUND.DAC
1520        LDA DAC.EXPONENT
1530        BIT DAC.SIGN
1540        BPL .1       POSITIVE
1550        ORA #$80     NEGATIVE
1560 .1     LDY #0
1570 .2     STA (PNTR),Y
1580        INY
1590        LDA DAC,Y
1600        CPY #10
1610        BCC .2
1620        RTS
1630 *---------------------------------
1640 *      ROUND DAC BY EXTENSION
1650 *---------------------------------
1660 ROUND.DAC
1670        LDA DAC.EXTENSION
1680        CMP #$50     COMPARE TO .5
1690        BCC .3       NO NEED TO ROUND
1700        LDY #8
1710        SED          DECIMAL MODE
1720 .1     LDA #0
1730        ADC DAC.HI,Y
1740        STA DAC.HI,Y
1750        BCC .2       NO NEED FOR FURTHER PROPAGATION
1760        DEY
1770        BPL .1       ...MORE BYTES
1780        INC DAC.EXPONENT
1790        BMI .4       ...OVERFLOW
1800        LDA #$10     .999...9 ROUNDED TO 1.000...0
1810        STA DAC.HI
1820 .2     CLD
1830 .3     LDA #0
1840        STA DAC.EXTENSION
1850        RTS
1860 .4     CLD
1870        JMP AS.OVRFLW
1880 *---------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8406:DOS3.3:S.KANER.VOKEY.txt
==========================================================================

   1   .LIF
1000  *SAVE S.KANER & VOKEY
1010  *--------------------------------
1020  *      BASED ON PROGRAM PRINTED IN MICRO MAGAZINE
1030  *      JUNE 1984, PAGES 33,34, BY H. CEM KANER
1040  *                           AND JOHN R. VOKEY
1050  *--------------------------------
1060  USER.VECTOR .EQ $0A,0B,0C
1070  FAC         .EQ $9D THRU $A1
1080  FAC.SIGN    .EQ $A2
1090  FAC.EXT     .EQ $AC
1100  *--------------------------------
1110  NORMALIZE.FAC.2 .EQ $E82E
1120  *--------------------------------
1130         .OR $300
1140         .TF B.KANER & VOKEY
1150  *--------------------------------
1160  LINK   LDA #$4C     "JMP" OPCODE
1170         STA USER.VECTOR
1180         LDA #RANDOM
1190         STA USER.VECTOR+1
1200         LDA /RANDOM
1210         STA USER.VECTOR+2
1220         RTS
1230  *--------------------------------
1240  Z.C    .HS 00.00.00.67.27  26407
1250  Z.A    .HS 07.50.89.2E.05  31415938565
1260  Z.OLD  .HS 00.00.00.00.00
1270  *--------------------------------
1280  Y.C    .HS 00.00.01.86.97  99991
1290  Y.A    .HS 01.F5.7B.1B.95  8413453205
1300  Y.OLD  .HS 00.00.00.00.00
1310  *--------------------------------
1320  X.C    .HS 00.00.00.00.03  3
1330  X.A    .HS 06.54.38.E9.25  27182819621
1340  X.OLD  .HS 00.00.00.00.00
1350  *--------------------------------
1360  GROUP      .BS 1
1370  MULTIPLIER .BS 5
1380  OLD        .BS 5
1390  *--------------------------------
1400  RANDOM LDY #Z.C-Z.C+4
1410         LDA FAC.SIGN
1420         BMI .1       SELECT Z
1430         LDY #Y.C-Z.C+4
1440         LDA FAC
1450         BEQ .1       SELECT Y
1460         LDY #X.C-Z.C+4   SELECT X
1470  .1     STY GROUP    SAVE FOR LATER
```

```
1480  *---LOAD SELECTED GROUP----------
1490         LDX #4       MOVE 5 BYTES
1500  .2     LDA Z.C,Y
1510         STA FAC+1,X
1520         LDA Z.A,Y
1530         STA MULTIPLIER,X
1540         LDA Z.OLD,Y
1550         STA OLD,X
1560         DEY
1570         DEX
1580         BPL .2
1590  *---MULTIPLY INTO FAC------------
1600         LDX #4
1610  .3     STX FAC.EXT  USE FOR BYTE COUNT
1620         LDA MULTIPLIER,X
1630         STA FAC      SAVE FOR 8-BIT MULITPLY
1640         LDY #7       COUNT BITS
1650  .4     LSR FAC      GET RIGHTMOST BIT INTO CARRY
1660         BCC .6       =0, SO WE DO NOT ADD THIS TIME
1670         CLC          =1, SO WE BETTER ADD
1680  .5     LDA FAC+1,X
1690         ADC OLD,X
1700         STA FAC+1,X
1710         DEX
1720         BPL .5
1730  .6     ASL OLD+4    SHIFT MULTIPLICAND LEFT
1740         ROL OLD+3
1750         ROL OLD+2
1760         ROL OLD+1
1770         ROL OLD
1780         LDX FAC.EXT  GET BYTE COUNT AGAIN
1790         DEY          NEXT BIT
1800         BPL .4
1810         DEX          REDUCE BYTE COUNT
1820         BPL .3       ...MORE BYTES
1830  *---SAVE NEW SEED IN OLD---------
1840         LDX #4
1850         LDY GROUP
1860  .7     LDA FAC+1,X
1870         STA Z.OLD,Y
1880         DEY
1890         DEX
1900         BPL .7
1910  *---NORMALIZE NEW VALUE----------
1920         LDY #$80     ASSUME A FRACTION
1930  .8     LDA FAC+1    LOOK AT LEADING BIT
1940         BMI .9       ...FINISHED NORMALIZING
1950         LSR FAC.SIGN
1960         ROR FAC+4
1970         ROR FAC+3
1980         ROR FAC+2
1990         ROR FAC+1
2000         DEY
2010         CPY #$58
```

```
2020        BCS .8
2030        LDY #0      LESS THAN 2^-40 IS ZERO
2040  .9    STY FAC     EXPONENT
2050        LDA #0
2060        STA FAC.SIGN MAKE IT POSITIVE
2070        RTS
```

```
=========================================================================
DOCUMENT :AAL-8406:DOS3.3:S.MotoSType.Obj.txt
=========================================================================

 1000  *SAVE S.MOTOROLA S-TYPE OBJECT
 1010         .OR $800
 1020  *--------------------------------
 1030  PORT       .EQ $00,01
 1040  HI.BIT     .EQ $02
 1050  NAME       .EQ $03 ... $0A
 1060  CHECK.SUM  .EQ $12
 1070  TYPE       .EQ $13
 1080  COUNT      .EQ $14
 1090  REMAINING  .EQ $15,16
 1100  START      .EQ $17,18
 1110  END        .EQ $19,1A
 1120  TARGET     .EQ $1B,1C
 1130  *--------------------------------
 1140  A1     .EQ $3C,3D
 1150  A2     .EQ $3E,3F
 1160  A3     .EQ $40,41
 1170  A4     .EQ $42,43
 1180  A5     .EQ $44,45
 1190  *--------------------------------
 1200  CTRLY.VECTOR         .EQ $3F8 THRU $3FA
 1210  DOS.REHOOK           .EQ $3EA
 1220  *--------------------------------
 1230  MON.NXTA4            .EQ $FCB4
 1240  MON.CROUT            .EQ $FD8E
 1250  MON.PRHEX            .EQ $FDDA
 1260  MON.COUT             .EQ $FDED
 1270  MON.SETVID           .EQ $FE93
 1280  *--------------------------------
 1290  *      SETUP CONTROL-Y
 1300  *--------------------------------
 1310  SETUP  LDA #SEND.DATA
 1320         STA CTRLY.VECTOR+1
 1330         LDA /SEND.DATA
 1340         STA CTRLY.VECTOR+2
 1350         LDA #$4C
 1360         STA CTRLY.VECTOR
 1370         RTS
 1380  *--------------------------------
 1390  *   *0:XX YY   (LO,HI OF PORT)
 1400  *   *:ZZ       (00 OR 80, FOR ASCII HI-BIT)
 1410  *   *:C1 C2 C3 C4 C5 C6 C7 C8   ASCII VALUES FOR
 1420  *                    THE 8 CHARACTERS OF THE NAME
 1430  *   *TARGET<START.END<Y>
 1440  *      IF PORT IS 0, DO NOT CHANGE OUTPUT
 1450  *      IF PORT IS 1...7, OUTPUT TO SLOT.
 1460  *         ELSE OUTPUT TO SUBROUTINE
 1470  *      SEND BYTES START...END
 1480  *
```

```
1490  *       1.  TURN ON OUTPUT PORT
1500  *       2.  SEND ID RECORD
1510  *       3.  SEND DATA RECORDS
1520  *       4.  SEND EOF RECORD
1530  *       5.  TURN OFF OUTPUT PORT
1540  *--------------------------------
1550  SEND.DATA
1560         JSR SAVE.PARAMETERS
1570         JSR TURN.ON.OUTPUT.PORT
1580         JSR SEND.ID.RECORD
1590         JSR RESTORE.PARAMETERS
1600         JSR SEND.DATA.RECORDS
1610         JSR SEND.EOF.RECORD
1620         JMP TURN.OFF.OUTPUT.PORT
1630  *--------------------------------
1640  SAVE.PARAMETERS
1650         LDX #1
1660  .1     LDA A1,X
1670         STA START,X
1680         LDA A2,X
1690         STA END,X
1700         LDA A4,X
1710         STA TARGET,X
1720         DEX
1730         BPL .1
1740         RTS
1750  *--------------------------------
1760  RESTORE.PARAMETERS
1770         LDX #1
1780  .1     LDA START,X
1790         STA A1,X
1800         LDA END,X
1810         STA A2,X
1820         LDA TARGET,X
1830         STA A4,X
1840         DEX
1850         BPL .1
1860         RTS
1870  *--------------------------------
1880  TURN.ON.OUTPUT.PORT
1890         LDX PORT+1     HI-BYTE OF PORT SPECIFIED
1900         BNE .1
1910         LDA PORT       LO-BYTE, MUST BE SLOT
1920         AND #$07
1930         BEQ .3       SLOT 0, JUST SCREEN
1940         ORA #$C0
1950         BNE .2       ...ALWAYS
1960  .1     TXA          HI-BYTE OF SUBROUTINE
1970         LDX PORT       LO-BYTE OF SUBROUTINE
1980  .2     STA $37
1990         STX $36
2000         JSR DOS.REHOOK
2010  .3     RTS
2020  *--------------------------------
```

```
2030  SEND.ID.RECORD
2040         LDA #'0'     TYPE = "0"
2050         STA TYPE
2060         LDA #8       COUNT = 8
2070         STA COUNT
2080         LDA #0       ADDR=0
2090         STA A4
2100         STA A4+1
2110         STA A1+1
2120         STA A2+1
2130         LDA #NAME
2140         STA A1
2150         LDA #NAME+7
2160         STA A2
2170         JMP SEND.RECORD
2180  *--------------------------------
2190  SEND.DATA.RECORDS
2200         LDA #'1'     TYPE = "1"
2210         STA TYPE
2220         INC A2       POINT JUST BEYOND THE END
2230         BNE .1
2240         INC A2+1
2250  .1     SEC
2260         LDX #20
2270         LDA A2       SEE HOW MANY BYTES LEFT
2280         SBC A1
2290         STA REMAINING
2300         LDA A2+1
2310         SBC A1+1
2320         STA REMAINING+1
2330         BNE .2       USE MIN(20,A2-A1+1)
2340         CPX REMAINING
2350         BCC .2
2360         LDX REMAINING
2370         BEQ .3       ...FINISHED
2380  .2     STX COUNT
2390         JSR SEND.RECORD
2400         JMP .1       ...ALWAYS
2410  .3     RTS
2420  *--------------------------------
2430  SEND.EOF.RECORD
2440         LDA #0       # OF DATA BYTES = 0
2450         STA COUNT
2460         LDA #'9'     TYPE="9"
2470         STA TYPE
2480         LDA TARGET   RUN ADDRESS (LO)
2490         STA A4
2500         LDA TARGET+1 RUN ADDRESS (HI)
2510         STA A4+1
2520         JMP SEND.RECORD
2530  *--------------------------------
2540  TURN.OFF.OUTPUT.PORT
2550         JSR MON.SETVID
2560         JMP DOS.REHOOK
```

```
2570  *--------------------------------
2580  SEND.RECORD
2590         LDA #'S'     LETTER "S"
2600         JSR SEND.FRAME
2610         LDA TYPE     TYPE "0", "1", OR "9"
2620         JSR SEND.FRAME
2630         LDA #0       INIT CHECKSUM
2640         STA CHECK.SUM
2650         CLC
2660         LDA COUNT    SEND BYTE COUNT
2670         ADC #3       ...INCLUDING ADDR AND CSUM
2680         JSR SEND.BYTE
2690         LDA A4+1     SEND ADDRESS
2700         JSR SEND.BYTE
2710         LDA A4
2720         JSR SEND.BYTE
2730         LDA COUNT    ANY DATA?
2740         BEQ .2       ...NO
2750         LDY #0       ...YES, SEND DATA
2760  .1     LDA (A1),Y
2770         JSR SEND.BYTE
2780         JSR MON.NXTA4
2790         DEC COUNT
2800         BNE .1
2810  .2     LDA CHECK.SUM     SEND CHECK SUM
2820         EOR #$FF
2830         JSR SEND.BYTE
2840         LDA #$0D     SEND CRLF
2850         JSR SEND.FRAME
2860         LDA #$0A     LINEFEED
2870         JMP SEND.FRAME
2880  *--------------------------------
2890  SEND.BYTE
2900         PHA          SAVE BYTE
2910         CLC
2920         ADC CHECK.SUM     ACCUMULATE CHECKSUM
2930         STA CHECK.SUM
2940         PLA          RECOVER BYTE
2950         PHA          SAVE ANOTHER COPY
2960         LSR          READY FIRST DIGIT
2970         LSR
2980         LSR
2990         LSR
3000         JSR SEND.DIGIT
3010         PLA          RECOVER BYTE
3020         AND #$0F     READY SECOND DIGIT
3030  SEND.DIGIT
3040         ORA #$30     CHANGE TO ASCII
3050         CMP #$3A
3060         BCC SEND.FRAME
3070         ADC #6       CHANGE TO A...F
3080  SEND.FRAME
3090         ORA HI.BIT   $00 OR $80
3100         JMP MON.COUT
```

```
3110  *--------------------------------
3120         .OR $300
3130  SAMPLE
3140         .HS 86.44.B7.01.00.41.42.43
3150         .HS 44.45.46.47.48.49.4A.4B
3160         .HS 4C.4D.4E
```

```
==============================================================================
DOCUMENT :AAL-8406:DOS3.3:S.PRT.COMMAND.txt
==============================================================================

 1000  *---------------------------------
 1010  *
 1020  *   SAMPLE PRT COMMAND
 1030  *
 1040  *---------------------------------
 1050  PRT.VECTOR  .EQ $D009
 1060  PATCH.SPACE .EQ $DF00
 1070  MON.COUT    .EQ $FDED
 1080  *---------------------------------
 1090         .OR PRT.VECTOR
 1100         JMP PRT         JUMP TO HANDLER
 1110  *---------------------------------
 1120         .OR PATCH.SPACE
 1130
 1140  PRT    LDX #0
 1150  .1     LDA STRING,X    OUTPUT THE
 1160         BEQ .2          CONTROL
 1170         JSR MON.COUT    STRING
 1180         INX
 1190         BPL .1
 1200
 1210  .2     RTS
 1220  *---------------------------------
 1230  STRING .HS 8D84        <CR><^D>
 1240         .AS -/PR#1/
 1250         .HS 8D          <CR>
 1260
 1270         .HS 9C9BB8      ELITE SPACING
 1280         .HS 9BA5C3      LEFT MARGIN
 1290         .HS B0B9B0      90 DOT SPACES
 1300         .HS 00          END MARKER
```

```
==============================================================================
DOCUMENT :AAL-8407:Articles:DisasmNameTable.txt
==============================================================================
```

Building Label Tables for DISASM...........Bob Sander-Cederlof

RAK-Ware's DISASM has the nice feature of being able to used a list of pre-defined labels when you are disassembling a block of code.  I needed to turn the //c monitor ROM ($F800-$FFFF) into source code, and Apple sent me a list of all their labels in this area.

The format of the label table, or name table, is very simple.  Each entry takes eight bytes:  the first two are the value, high byte first; the remaining six are the label name, in ASCII with high bit set.  If the name is less than six characters long, zeroes are used to fill out the entry.

Very simple to explain, but how do you enter things like that in the S-C Macro Assembler?  The example on the DISASM disk does it this way:

```
       1000      .HS FDED
       1010      .AS -/COUT/
       1020      .HS 00000000
       1030      .HS FDF0
       1040      .AS -/COUT1/
       1050      .HS 000000
       and so on.
```

That works, but it is so error prone and time wasting that I gave up before I started.  However, there is an easy way using macros and abbreviations.

Start by defining a macro which will build one entry:

```
       1000    .MA LBL
       1010    .HS ]1
       1020    .AS -/]2/
       1030    .BS *+7/8*8-*
       1040    .EM
```

The macro is named LBL, and will be used like this:

```
       1050    >LBL FDED,COUT
       1060    >LBL FDF0,COUT1
```

Line 1030 is the tricky one.  This .BS will pad out an entry to an even multiple of 8 bytes.  Now, assuming the origin started at an even multiple of 8, and assuming you are writing the table on a target file, that macro builds the kind of entries DISASM wants.  Instead of just assuming, lets add:

```
       0900    .OR $4000
       0910    .TF B.NAMETBL
```

I also mentioned abbreviations above. I even get tired of typing "tab>LBL ", you know. Usually when I have a lot of lines to type that have a common element, I use some special character that is easy to type and not present in the lines I plan to type. Then after all the lines are in, I use the REPLACE command to substitute the longer string for the single-character abbreviation I have used. Thus, I can type:

```
       1050 .FDED,COUT
       1060 .FDF0,COUT1
       et cetera
```

and after many lines type

```
       REP /./ >LBL /1050,A
```

I was about up to FA90 when it dawned on me that I could break the symbols into blocks within a page, and include the page value in my abbreviation:

```
       1050 .ED,COUT
       1060 .F0,COUT1
       REP /./ >LBL FD/1050,A
```

With all these shortcuts, I was able to enter over 400 label names and definitions in less than an hour.

Let the computer work FOR you!

```
=============================================================================
DOCUMENT :AAL-8407:Articles:DP18.Part.3.txt
=============================================================================
```

18-digit Arithmetic, Part 3................Bob Sander-Cederlof

Plowing ahead, this installment will offer the division and input conversion subroutines.

You will remember that we covered addition and subtraction in the May 1984 issue, and multiplication in June.  Now it's time for division, which completes the fundamental arithmetic operations.  All four of these routines are designed to operate on two arguments stored in DAC and ARG, leaving the result in DAC.  Addition and subtraction leave "garbage" in ARG.  Multiplication leaves ARG unchanged.  Division leaves in ARG what was in DAC.

Division is simple enough in concept, but no one would call it simple in implementation.  "How many groups of X are in Y?"  "If I deal an entire deck of 52 cards to 4 people, how many will each person get?" "If I scramble a dozen eggs, and serve them in equal-size portions to 7 people, how many eggs will each eat?"  (Really, I am good cook!)

Suppose I have a pile of pennies, and want to find out how many dollars they represent.  I will count out piles of 100 pennies, moving them into separate piles.  Then I will count the little piles.  Now, suppose I have two 18-digit numbers in my computer and want to divide the one in ARG by the one in DAC....  I will subtract the value in DAC from the one in ARG over and over, until I finally cross zero.  Then if I was wise enough to count how many times I did the subtraction, I have the answer.

Let's look at the problem in more detail now.  What I want to do is divide the value in ARG by the value in DAC:

```
        numerator (in ARG)
       -------------------  =  quotient (in DAC)
       denominator (in DAC)
```

Numbers in DP18 can be positive or negative, so we have to remember the rules of signed division.  If the signs of the numerator and denominator are the same, the quotient will be positive; if they are different, the quotient will be negative.

Numbers in DP18 are coded as 18-digit fractions with a power- of-ten exponent.  Remembering algebra:

```
        .f * 10^m      f
        ---------  =  --- * 10^(m-n)
        .g * 10^n      g
```

The 18-digit fractions are normalized so that there are no leading zeroes. That is, the value will either be all zero, or it will be between .1 and .999999999999999999 (inclusive).

I think it is time now to start looking at the program. In the listing which follows there are references to subroutines and variables which we defined in the previous two installments of this series.

Line 4250 swaps the contents of ARG and DAC. I did it this way because it leaves something possibly useful in ARG after the division is finished. If you wanted to form the reciprocal quotient, DAC=DAC/ARG, you can enter at DDIVR, which skips the swapping step.

Lines 4260-4270 check for the illegal case of division by zero. If I divide something into zero-size parts, I get an infinite number of these parts. That's fine, but the DP18 has no representation for infinity; therefore we say it is illegal to divide by zero, just like Applesoft does. Some computers and some software arithmetic packages do represent infinity, but DP18 does not. Zero values are represented by having an exponent byte of zero, so we only have to check one byte here.

Lines 4280-4310 form the sign of the quotient. This is the same as lines 1280-1310 of the DMULT listing given last month, and so we could make them into a subroutine. The subroutine would take 10 bytes, and the two JSR's make another 6. That's 16 bytes, against the 18 bytes for the two versions of in-line code. Saves a total of 2 bytes, at a cost of adding 12 cpu cycles to both multiply and divide. (Small digression into the kind of trade-offs I am continually making....)

Lines 4330-4390 compute the exponent of the quotient, and check for overflow and underflow cases. The special case of the numerator being zero is also caught here, line 4350. Line 4380 restores the bias of $40. Bias? Remember, the exponent is kept in memory with $40 added to it, so that the range -63 through +63 is represented by $01 through $7F.

If the new exponent is still in the range $00 through $7F, we will go ahead and do the division. If not, the quotient is either too small (underflow) or too large (overflow). For example, 10^-40 / 10^40 results in 10^-80, which is too small for DP18. Lines 4410-4470 catch these cases, and change the quotient to zero. If the new exponent is between $80 and $BF, it represents 10^64 or larger, and so we call on the Applesoft OVERFLOW error.

Lines 4500-4550 set up the loop which does the actual division of the fractions. The 6502's decimal mode will be used during this loop. Ten bytes in MAC (defined in DMULT last month) will be used to hold the quotient until we are through with DAC. The X-register will be used to count out the 20 digits. The other end of the loop is in lines 4920-4930, where X is decremented and tested.

The body of the loop is really a lot simpler than it looks. Basically, ARG is subtracted from DAC until DAC goes negative. The number of subtractions is counted in MAC+9. Then ARG is added back to DAC to make it positive again, and MAC+9 decremented. The result is a quotient digit in MAC+9, and a remainder in DAC. One extra digit is needed, extending DAC on the left end. This digit is carried in the stack. See it pushed at line 4710, pulled at line 4790.

After each digit of the quotient is determined, both MAC and DAC are shifted left one digit place. This might shift a significant digit out of DAC (the remainder), so it is lifted out first and saved on the stack (lines 4570-4630). If the first two digits of the remainder (happen to be "00", then we know without subtracting that the quotient digit in this position will also be "0". (Remember that the leading digit of the denominator in ARG is NEVER zero.) This fact can speed up divisions, so it is tested for at line 4580, with lines 4670-4680.

After all 20 digits are formed, the loop terminates. Line 4950 then returns us to binary mode. Line 4960 adds one to the quotient exponent, adjusting for the normalization step. (.9/.1 = 9, but we want to represent it as .9*10^1.) If the exponent now is negative ($80), it may be still in range if the leading digit of the quotient is zero (.1/.9 = 0.1111...). This test takes place at lines 4970-5000.

Lines 5020-5060 copy the quotient from MAC to DAC. These are the same as lines 1330-1370 in DMULT, so they could be made into a subroutine. Two other candidates for subroutines are lines 4720-4780, which are identical to lines 1680-1740 of DADD (May 1984); and lines 4830-4890, which are the same as 1530-1590 of DADD.

Finally, DDIV exits by jumping to NORMALIZE.DAC.

Doesn't all this take a lot of time? You bet it does! I timed it in the full DP18 package with a program that looked like this:

```
       &DP:INPUT X(0) : INPUT X(2)
       FOR I = 1 TO 100
       &DP:X(4) = X(0)/X(2)
       NEXT
```

I determined the loop overhead by entering a value zero for X(0). Since this case skips around nearly everything in DDIV, I called its time the loop overhead time. After subtracting out the loop overhead, the times look like this:

```
       0/anything      0
       x/x             12 msec
       1/9=.1111...    23 msec
       8/9=.8888...    49 msec
       1/7=.142857...  35 msec
```

It looks like the maximum time, which would be for a quotient with all 20 digits = 9, would be about 53 msec. The average time, about 35 msec. This compares with an average Applesoft 9-digit division time of about 7 msec.

<<<listing here.>>>

DP18 Input Conversion

The input conversion subroutine processes characters from memory to produce a value in DAC. This is analogous to what the equivalent subroutine in Applesoft ROMs does.

It is so analogous, in fact, that I even depend upon CHRGET and CHRGOT to fetch successive characters from memory. It is a lot faster than Applesoft conversion, however, because it is BCD coded rather than binary. This means that, stripping away the frills such as sign, exponent part, and decimal point, it even easier than an ASCII to hex conversion.

Of course, we need all those frills. Look ahead to the program listing which follows: Lines 1200-1220, just those three little lines, handle the conversion of digits. All the rest of the page is for frills! Well, to be honest about it, two of the three lines call subroutines, but still, the frills predominate.

The acceptable format of numbers is basically the same as that which normal Applesoft accepts. A leading sign is optional. The numeric portion can be more than 20 digits long, but only the first 20 will be accumulated (not counting leading zeroes). A decimal point is optional anywhere in the numeric portion. An exponent part can be appended to the numeric portion, and consists of the letter "E", and optional sign, and one or two digits. The exponent can be up to 81, just so the final number evaluates between .1*10^-63 and .9999...9*10^63. Numbers smaller than .1*10^-63 will be changed to zero, and numbers larger than .9999...9*10^63 will cause an OVERFLOW ERROR.

Looking at the program, lines 1040-1080 clear a working area which comprises DAC and four other variables: SGNEXP, EXP, DGTCNT, and DECFLG. SGNEXP will be used to hold the sign of the exponent part; EXP will hold the value of the exponent part; DGTCNT will count the digits in the numeric portion; and DECFLG will flag the occurrence of a decimal point. DAC includes DAC.SIGN. Note that the X-register will be left with $FF, which fact is important at line 1170 below.

Lines 1090-1100 preset the DAC.EXPONENT to $40, which indicates 10^0. This will be incremented along with DGTCNT until a decimal point is encountered.

Lines 1110-1180 handle the optional leading sign. DAC.SIGN has already been cleared above, indicating the positive case. If a minus sign is in front of the number, line 1170 sets DAC.SIGN negative. Note that calling CHRGOT and CHRGET to retrieve characters automatically eliminates (ignores) blanks. CHRGOT/CHRGET also checks whether the character retrieved is a digit or not, and indicates digits by carry clear. If the first non-blank character is a digit, we immediately jump to the numeric loop at line 1200. If not, the subroutine FIN.SIGN checks for a + or - character. The + or - may or may not be tokenized, depending on whether the string is from an INPUT statement or is a constant embedded in a program, so we have to check for both the character and the token form of both signs. FIN.SIGN handles this checking.

If that first character is neither a digit nor a sign, it may be a letter "E" or a decimal point; so, we go down to lines 1240-1270 to check for those two cases. If neither of these either, we must be at the end of the number. If it is a decimal point, lines 1630-1650 record the fact that a decimal point was found and also check whether this is the first one found or not. If the first, back we go to continue looking for digits. If not the first, it must be the end of the number, so we fall into the final processing section at line 1670.

Exponents are more difficult, because the value actually must be converted from ASCII to binary. Lines 1290-1610 do the work, including handling of the optional sign, and range checking.

Lines 1670-1730 compute the final exponent value. This is the number of digits before the decimal point (not counting any leading zeroes you may have typed to confuse me) plus the exponent computed in the optional "E" field. If the result is negative, between $C0 and $FF, it indicates underflow; in this case, the value is changed to zero. If there were no non-zero digits in the numeric portion, the value is set to zero regardless of any "E" field. If the resulting exponent is between $80 and $BF, it indicates OVERFLOW.

Lines 1840-2130 accumulate individual digits. DGTCNT is used to index into the nybbles of DAC, and the digit is stored directly into place. Leading zeroes on the numeric field are handled here (lines 2090-2120). Leading zeroes before a decimal point are entirely ignored, while leading zeroes after a decimal point cause the DAC.EXPONENT to be decremented. The incrementation of DAC.EXPONENT for each significant digit on the left of the decimal point is also taken care of here (lines 2020-2070).

This complete the third installment of DP18. We are well on the way to a working subset of the entire package. We still need output conversion and some sort of linkage to Applesoft before we can begin to see it all run. The entire DP18 package really exists, and works, now. It includes PRINT USING, very fancy input screen handling, full expression parsing, and all the math functions. Several of you have been very anxious to get the whole package for use in projects of your own, so we have offered a source code license to DP18 on an "as is" basis for only $200.

=====================================================================
DOCUMENT :AAL-8407:Articles:Front.Page.txt
=====================================================================

$1.80

Volume 4 -- Issue 10                    July, 1984

In This Issue...



Feedback on our DOSonomy

Our little dossier of DOS names was well received. It may be we will soon have so many names we will need a dosser (a large basket that can be carried on the back) to hold them all. On the other hand, if we keep writing about this our fortunes may reverse, forcing to finding new quarters in a doss house. What is the critical dosage?

Dan Pote offers "Kinda-Sorta-DOS". Which led Bill to coin "MaybeDOS". Randy Horton reminded us of "Ante-DiluviDOS". Chris Balthrop enters MacroDOS and "What's Up DOS". (I think the latter is "Buggy". Or "Bugsy"? Oh, it's not bunny anymore...) If you can take all this, you may be too docile.

Don Lancaster Strikes Again

We just have a little space and a little time to mention Don's new Assembly Cookbook for the Apple II/IIe, which just arrived. It looks like another winner! Look for a full review next month, or check our ad on page 3 for ordering info.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8407:Articles:IIc.Notes.txt
==============================================================================

Our //c came in, and we love it.  However...

The //c package does not include any DOS 3.3 master.  Everything is ProDOS.  Of course you do get a DOS 3.3 with most software you purchase.  And of course ProDOS includes a disk copier that is supposed to be able to copy DOS 3.3 disks when you need to back up your DOS-based software.  However...

The ProDOS disk copier which is being shipped with the //c has a serious bug.  When you are copying a DOS-based disk it ignores the volume number recorded on the source disk, and forces the copy to be volume 254.  That is fine if the source just happened to be volume 254 also, but chances are it isn't.  I have many disks around here which are volume 1.  The DOS image and the VTOC both think the disk copied by //c ProDOS is volume 1, but RWTS discovers it is volume 254 and refuses to cooperate any further.

I guess the solution is to use the old faithful COPYA from your DOS 3.3 System Master.  Since that doesn't come with a //c system, we are including licensed copies of COPYA and FID on our Macro 1.1 disks now.

More gotchas....  Apple decided it was time to rewrite large chunks of the monitor.  Necessarily so, because the disassembler now has to cope with 27 new opcodes and address modes.  The removed four entries from the monitor command table, and changed its starting point.  This throws off the "$" command in the S-C Macro Assemblers, all versions.

If you have Macro 1.1, the //e version is the one you should be running in your //c.  You can fix the "$" command with these patches:

| $1000 version | $D000 version | old value | new value |
|---|---|---|---|
| $147B | $D47B | $17 | $13 |
| $1486 | $D486 | $CC | $CD |
| $148B | $D48B | $15 | $11 |

A more elegant patch is possible, which automatically adjusts for whether you are in a //e or //c.  If you want this, and have a 1.1 version prior to serial # 675, send us $5 for an update.

We have tried RAK-Ware's DISASM 2.2e on our //c, and it works fine. It even picks up the 27 new opcodes and address modes automatically, because DISASM links to the monitor disassembler.  Older versions of DISASM will not run on a //e or //c.

```
=====================================================================
DOCUMENT :AAL-8407:Articles:My.Ad.txt
=====================================================================

S-C Macro Assembler Version 1.0......................................$80
S-C Macro Assembler Version 1.1...................................$92.50
Version 1.1 Update................................................$12.50
Source Code for Version 1.1 (on two disk sides).....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks..............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $35)  $30
Routine Machine (Southwestern Data Systems)............(reg. $64.95)  $60
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36
FLASH! Integer BASIC Compiler (Laumer Research)......................$79
Fontrix (Data Transforms)............................................$75
Aztec C Compiler System (Manx Software).................(reg. $199) $180

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                           or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ..........................$20

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
```

```
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Assembly Lines: The Book", Roger Wagner................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "What's Where Guide" (updates first edition).............($9.95)   $9
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($17.95)  $17
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

==============================================================================
DOCUMENT :AAL-8407:Articles:Orphans.Widows.txt
==============================================================================

Orphans and Widows.........................Bob Sander-Cederlof

James, a brother of Jesus Christ, wrote: "Pure religion and undefiled before God and the Father is this, to visit the fatherless and widows in their affliction, and to keep himself unspotted from the world." (chapter 1, verse 27, King James Version)

Of course, he was referring to real life and to real people with real needs, but it still serves to introduce this little announcement.

"Orphans" and "widows" are also terms used in word processing to describe the lamentable situation of one line of a paragraph being left all alone on one page, while the rest is on another page. If that one line is the last line of a paragraph which won't quite fit, "she" is forced to the top of the next page, and is a widow. If the lonely line is the first line of a paragraph, dwelling at the bottom of a page, bereft of the rest of its family on the following page, he or she is indeed an orphan.

High class word processors give you the option of automatically "visiting" orphans and widows "in their affliction". Thanks to Bobby Deen, this feature is now (as of June 29th) included in the S-C Word Processor (whether high class or not). When the feature is selected (by the "!or1" directive), orphans get moved to the next page and widows get squeezed onto the current page.

Bobby is also working on, and he says it is now functional but somewhat unfinished, a version that fully uses the 80-column display on the Apple //e. We already had 80-column preview, but he is developing 80-column text display during edit/entry mode.

==========================================================================
DOCUMENT :AAL-8407:Articles:Quick.Mem.Test.txt
==========================================================================

Quick Memory Testing.......................Bob Sander-Cederlof

What do you do when a friend brings his Apple over with an intermittent memory failure? You KNOW you have a memory test program somewhere, but WHERE?

Here is a quick way to test normal RAM between $7D0 and $BFFF. (RAM in //e hyperspace or banked into ROM space is another matter.) Turn on your friend's computer, and hit reset to abort the booting sequence. We don't need or want DOS around while we are testing memory. Type HOME and CALL-151 to get into the monitor. Then type the following monitor command:

```
*N 7D0:00 N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:55 N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:AA N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:FF N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   34:0
```

The "*" is the monitor prompt, so don't you type it. There are no carriage returns in the line above, it just wraps around the 40-column screen that way. There must be one trailing blank after the "34:0" at the end. This makes the monitor repeat the whole command line forever.

I started the test at $7D0 so there will be some visible feedback, but most of the screen will stay clear. If you begin testing at a lower address, any errors displayed on the screen might be overwritten as soon as they show up.

When you type the RETURN key you will see a line of inverse at-signs at the bottom of the screen. After a few seconds, this will change to flashing U. Then *, and then some other character, depending on what kind of Apple you have. Then the cycle will start over again.

Until a memory error is detected. Any error will cause two lines to be printed, giving the address before the error with its contents and the contents of the error byte, and the address of the error byte with its actual contents and should-be contents. For example, if you were in the "AA" phase, and $8123 came up with $AB, you would see:

```
       8122-AA (AB)    byte before error
       8123-AB (AA)    error byte
```

If any error lines start printing, note which bit is bad and which 16K bank it is in. Then you can point directly to the bad chip.

```
                       7  6  5  4  3  2  1  0
        7D0...3FFF    C10 C9 C8 C7 C6 C5 C4 C3
```

```
4000...7FFF    D10 D9 D8 D7 D6 D5 D4 D3
8000...BFFF    E10 E9 E8 E7 E6 E5 E4 E3
```

=====================================================================
DOCUMENT :AAL-8407:Articles:Sieve.6502.txt
=====================================================================

Updating the 6502 Prime Sifter.............Bob Sander-Cederlof

I spent a half day applying Peter's algorithm improvements to the November 1982 6502 version, and refining the program as much as I could. It now runs in 175 milliseconds per iteration, or 1000 iterations in 175 seconds. Still way behind the 68000, of course. On the other hand, a 6MHz 6502, with fast enough RAM for no wait states, would be faster than a 12.5 MHz 68000. And it remains to be seen what the 65802 could do.

In the process of running various versions and various tests, I discovered that the innermost loop, at lines 1820-1850, is executed 10277 times. This means that, while marking out the odd non-primes between 1 and 16383, a total of 10277 such marks are made. Since only odd numbers are assigned slots in the working array, giving only 8192 such slots, you can see that some numbers get stricken more than once. These are the numbers with more than one prime factor. The most-stricken number is 3*5*7*11*13 = 15015, which gets five strikes. The loop takes 11 cycles as written, and I don't see any way to shorten it any further or to reduce the number of times it is used. Do you?

The loop time is 11*10277 is 121297 cycles, or about 120 msec out of the total 175. The array clearing accounts for another 41 msecs, leaving only 14 msec for all the rest of the program. Not bad!

Here is a little Applesoft program which will make a nice neat listing of primes from the working array, assuming it runs from $6000 through $7FFF.

```
100 HIMEM:24576
110 FOR A = 24576 TO 32767
120 IF PEEK (A) = 0 THEN
    PRINT RIGHT$("      "+STR$((A-24576)*2+1,7);:
    N = N + 1
130 IF N = 10 THEN PRINT : N = 0
140 NEXT
```

=============================================================================
DOCUMENT :AAL-8407:Articles:Sieve.68000.txt
=============================================================================

68000 Sieve Benchmark.......................Peter J. McInerney
New Zealand

Here are two versions of the Sieve of Eratosthenes for the MC68000. They provide ample justification for the power claimed for this chip.

The first version is a fairly straightforward translation of the algorithm as presented in the November 1982 AAL, by Tony Brightwell. Tony's best time in the 6502 was 183 seconds for 1000 repetitions; in my 12.5 MHz DTACK GROUNDED attached processor, 1000 repetitions took only 40 seconds.

Compare the 68000 code with the 6502 code, and I'm sure you will agree the 68000 version is much easier to understand. Note the use of long instructions in the array clearing loop and the two-dimensional indexing in lines 1230 and 1310. Other nice things are the shift left by 3 (multiply by 8) in line 1270 and the decrement & branch instructions in lines 1120 and 1400. Also very useful is the postincrement address mode, which automatically increments the address kept in the referenced register by 1, 2, or 4 depending on the size of the operation. This is used for popping off (downward growing) stacks or as here to advance through memory. There is also a predecrement mode but I did not use it in these example programs.

The second version uses a modified algorithm. The changes I made should apply to the 6502 version also, improving it in about the same proportion.

* Since we are ignoring even numbers, we may as well leave them out of the array entirely, thus halving the array size.

* We can therefore simplify the formula for odd squares from S*8+1 to S*4.

* We can even do away with the *4 part by adding 4 each time rather than 1.

* The initial array clearing loop can be made faster by using more than one CLR instruction per loop.

This modified version does 1000 iterations in only 33 seconds! It is only slightly harder to follow than the first version, and only slightly larger. In fact, if we forego the final modification above, the code is actually shorter. I think most of the speedup comes from halving the array size.

If you have a Macintosh, and can manage to load machine code into it, you should find everything running about half as fast as my DTACK GROUNDED board.

[ We tried the program on our QWERTY Q-68 board, and it took roughly 10 times as long as Peter's DG board.  Understandable, since it was using Apple memory at .5MHz rate for all work.  (Bill&Bob) ]

==========================================================================
DOCUMENT :AAL-8407:Articles:Speed.Vs.Space.txt
==========================================================================

Speed vs. Space.............................Bob Sander-Cederlof

There are always tradeoffs. If you have plenty of memory, you can write faster code. If you have plenty of time, you can write smaller code. In an "academic" situation you may have plenty of both, so you can write "creative" code, stretching the frontiers of knowledge. In a "real" world it seems there is never enough time or memory, so jobs have to be finished on a very short schedule, fit in a tiny ROM or RAM, and run like greased lightning.

A case in point is last month's segment of the DP18 series: the SHIFT.MAC.RIGHT.ONE subroutine on page 8 takes about 1827 clock cycles, and fits in 25 bytes. Upon reflection, I see a way to write a 34-byte version that takes only 1029 cycles. If I can use nine more bytes, I can shave about 800 microseconds off each and every multiply. (Maybe a total of a whole minute per day!) That might be important, or it might not; but seeing the two techniques side-by-side is probably valuable.

```
1970 SHIFT.MAC.RIGHT.ONE
1980      LDY #4     4 BITS RIGHT
1990 .0   LDX #1     20 BYTES
2000      LSR MAC
2010 .1   ROR MAC,X
2020      INX        NEXT BYTE
2030      PHP
2040      CPX #20
2050      BCS .2     NO MORE BYTES
2060      PLP
2070      JMP .1
2080 .2   PLP
2090      DEY        NEXT BIT
2100      BNE .0
2110      RTS

1970 SHIFT.MAC.RIGHT.ONE
1980      LDX #0     FOR X=0 TO 19
1990      TXA        NEW 1ST NYBBLE = 0
2000 .1   STA TEMP   SAVE FOR HI NYBBLE
2010      LDA MAC,X  MOVE LOW NYBBLE
2020      ASL            TO HI SIDE
2030      ASL
2040      ASL
2050      ASL
2060      PHA        SAVE ON STACK
2070      LDA MAC,X  MOVE HI NYBBLE
2080      LSR            TO LOW SIDE
2090      LSR
2100      LSR
```

```
2110       LSR
2120       ORA TEMP   MERGE WITH NEW
2130       STA MAC,X      HI NYBBLE
2140       PLA        HI NYBBLE OF NEXT BYTE
2150       INX        NEXT X
2160       CPX #20
2170       BCC .1
2180       RTS
```

The smaller method uses two nested loops. The inner loop shifts all 20 bytes of MAC right one bit. The outer loop does the inner loop four times. If I counted cycles correctly, the time is 4*(19*23+18)+7. The faster method uses one loop to scan through the twenty bytes one time. The timing works out as 20*51+9.

Upon still further reflection, it dawned on me that a 38 byte version could run in 840 cycles! This version processes the bytes from right to left instead of left to right; eliminates the PHA-PLA and STA-ORA TEMP of the second version above; and loops only 19 times rather than 20. The timing is 19*43+23.

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDX #19      FOR X = 19 TO 1 STEP -1
1990 .1    LDA MAC,X    SHIFT HI- TO LO-
2000       LSR
2010       LSR
2020       LSR
2030       LSR
2040       STA MAC,X    SAVE IN FORM 0X
2050       LDA MAC-1,X  GET LO- OF HIGHER BYTE
2060       ASL
2070       ASL
2080       ASL
2090       ASL
2100       ORA MAC,X    MERGE THE NYBBLES
2110       STA MAC,X
2120       DEX          NEXT X
2130       BNE .1       ...UNTIL 0
2140       LDA MAC      PROCESS HIGHEST BYTE
2150       LSR          INTRODUCE LEADING ZERO
2160       LSR
2170       LSR
2180       LSR
2190       STA MAC
2200       RTS
```

Of course an even faster approach is to emulate the loops I wrote for shifting 10-bytes left or right 4-bits. The program would look like this:

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDY #4
1990 .1    LSR MAC
2000       LSR MAC+1
            .
```

```
          .
          .
2180      LSR MAC+19
2190      DEY
2200      BNE .1
2210      RTS
```

This version takes 2+3*20+4 = 66 bytes. Yet the timing is only (4*6+5)*20+7 = 587 clock cycles. And by writing out the four loops all the way, we use 4*3*20 = 240 bytes; the time would be 4*6*20 or 480 cycles.
How about another example? The MULTIPLY.ARG.BY.N subroutine on the same page last month was nice and short, but very slow. The subroutine is called once for each non-zero digit in the multiplier, or up to 20 times. What it does is add the multiplicand to MAC the number of times corresponding to the current multplier digit. If we assume the distribution of digits is random, with equal probablility for any digit 1...9 in any position, the average number of adds will be 5. Actually there will be zero digits too, so the average will be 4.5 instead of 5, with the subroutine not even being called for zero digits.

For 20 digits, 4.5 addition loops per digit, that is an average of 90 addition loops. And a maximum, when all digits are 9, of 180 addition loops.

Now, if there is enough RAM around, we can pre-calculate all partial products from 1 to 9 of the multiplicand and save them in a buffer area. Each partial product will take 11 bytes. We already have the first one in ARG, so for 2...9 we will need 8*11 or 88 bytes of storage. It will take 8 addition loops to form these partial products. Once they are all stored, the MULTIPLY.ARG.BY.N subroutine will always do exactly one addition loop no matter what the non-zero digit is. Therefore the maximum number of addition loops is 8+20 or 28, compared to 180! And the average (assuming there will be 2 zero digits out of 20 on the average) will be 26 addition loops.

The inner loop in MULTIPLY.ARG.BY.N, called "addition loop" above, takes 20 cycles. If we implement this new method, we will have shortened the average case from 1800 to 520 cycles, and the maximum from 3600 to 560 cycles. Of course the whole DMULT routine includes more time-consuming code, but this subroutine was the biggest factor. Taking the SHIFT.MAC.RIGHT.ONE improvements also, we have shortened the overall time in the average case by 2078 cycles, or 2 milliseconds per multiply. In the maximum case, the savings is nearly 4 milliseconds.

Of course, it takes more code space as well as the 88-byte partial product buffer for the new method. And it will take more time to write such a program. You have to make tradeoffs.
1

```
===============================================================================
DOCUMENT :AAL-8407:Articles:Swap.Sort.txt
===============================================================================
```

Sorting and Swapping.......................Bob Sander-Cederlof

Jack McDonald, writing in the July 1984 Software News, posed a puzzle for programmers: using nothing more than a series of calls to a SWAP, sort five items into ascending order. SWAP compares two items according to the indexes supplied, and exchanges the items if they are out of order. For example, calls on SWAP which follow the pattern of a "Bubble Sort" would look like this:

```
    SWAP (1,2)    SWAP (1,2)    SWAP (1,2)    SWAP (1,2)
    SWAP (2,3)    SWAP (2,3)    SWAP (2,3)
    SWAP (3,4)    SWAP (3,4)
    SWAP (4,5)
```

That is ten swaps, which is more than necessary. You can do it in nine, which was McDonalds Puzzle. He gave an answer, and I found another. It was fun writing some quick code to test various swap-lists.

First I wrote a macro named "S" which loaded the two index numbers into X and Y, and called a subroutine named SWAP. See it in lines 1030-1070.

Then I coded SWAP (lines 1200-1290), which compared two bytes at BASE,X and BASE,Y; if they were out of order, I swapped them around. To make things easy for me, I put BASE at $500, which just happens to be the third line on the video screen. That way I could watch everything happen without struggling to code I/O routines.

I wrote a program which would initialize a 5-byte string to all $01 (no program, really just a data definition at line 1670); another which copies the string to BASE (LOAD, lines 1590-1650); another which counts up from 0101010101 to 0505050505, so that all possible combinations would be run through (NEXT, lines 1770-1870); and another to do all these in connection with SORT, which performed a list of SWAP calls. The result was a method for visualizing and checking various groups of SWAPs to see if they could sort any initial permutation into ascending order. Assemble, and type MGO NEXT to see it all work.

Here is the code, with two possible SWAP orders which work, of nine steps each.

I also got interested in permutation generation, and came up with the following macros and code to generate all 120 permutations of five items, without any extra steps, each step being the simple interchange of two items. Assemble, and type MGO PERMUTE to see it generate 120 strings of the letters ABCDE in different arrangements.

```
==============================================================================
DOCUMENT :AAL-8407:DOS3.3:Faster.ShiftRt1.txt
==============================================================================

 1970  SHIFT.MAC.RIGHT.ONE
 1980         LDX #0        FOR X=0 TO 19
 1990         TXA           NEW 1ST NYBBLE = 0
 2000  .1     STA TEMP      SAVE FOR HI NYBBLE
 2010         LDA MAC,X     MOVE LOW NYBBLE TO HI SIDE
 2020         ASL
 2030         ASL
 2040         ASL
 2050         ASL
 2060         PHA           SAVE ON STACK
 2070         LDA MAC,X     MOVE HI NYBBLE TO LOW
 2080         LSR
 2090         LSR
 2100         LSR
 2110         LSR
 2120         ORA TEMP      MERGE WITH NEW HI NYBBLE
 2130         STA MAC,X
 2140         PLA           GET HI NYBBLE OF NEXT BYTE
 2150         INX           NEXT X
 2160         CPX #20
 2170         BCC .1
 2180         RTS
```

==============================================================================
DOCUMENT :AAL-8407:DOS3.3:LIST.PRIMES.txt
==============================================================================

£24576dÅA–24576¡32767Rn≠,(A)–0ƒ∫È("       "»‰((A…24576) 2»1),7);:N –
N»1bx≠N–10ƒ∫:N–0hÇÇ

==============================================================================
DOCUMENT :AAL-8407:DOS3.3:S.DP18.DIVIDE.txt
==============================================================================

```
 1000  *SAVE S.DP18 DIVIDE
 4220  *--------------------------------
 4230  *  DAC = ARG / DAC
 4240  *--------------------------------
 4250  DDIV   JSR SWAP.ARG.DAC  ...CHANGE TO DAC = DAC/ARG
 4260  DDIVR  LDA ARG.EXPONENT  CHECK FOR ZERO DENOMINATOR
 4270         BEQ .2            ...X/0 IS ILLEGAL
 4280  *---FORM SIGN OF QUOTIENT--------
 4290         LDA DAC.SIGN
 4300         EOR ARG.SIGN
 4310         STA DAC.SIGN
 4320  *---COMPUTE EXPONENT OF QUOTIENT-
 4330         SEC
 4340         LDA DAC.EXPONENT
 4350         BEQ .0       ...0/X=0
 4360         SBC ARG.EXPONENT
 4370         CLC
 4380         ADC #$40     ADJUST OFFSET
 4390         STA DAC.EXPONENT
 4400  *---CHECK OVER/UNDERFLOW---------
 4410         BPL .3       ...NEITHER
 4420         ASL          SEE WHICH...
 4430         BPL .1       ...OVERFLOW
 4440  .0     LDA #0       ...UNDERFLOW, SET RESULT = 0
 4450         STA DAC.SIGN
 4460         STA DAC.EXPONENT
 4470         RTS
 4480  .1     JMP AS.OVRFLW
 4490  .2     JMP AS.ZRODIV     DIVISION BY ZERO ERROR
 4500  *---SET UP QUOTIENT LOOP---------
 4510  .3     SED          DECIMAL MODE
 4520         LDA #0
 4530         STA MAC+9    CLEAR FIRST QUOTIENT DIGIT
 4540         LDX #20      DO 20 DIGITS
 4550         BNE .5       ...ALWAYS
 4560  *---CONTINUE QUOTIENT LOOP-------
 4570  .4     LDA DAC.HI
 4580         PHP          SAVE ZERO STATUS
 4590         LSR
 4600         LSR
 4610         LSR
 4620         LSR
 4630         PHA          DAC LEFT EXTENSION
 4640         JSR SHIFT.DAC.LEFT.ONE
 4650         JSR SHIFT.MAC.LEFT.ONE
 4660         PLA          DAC LEFT EXTENSION
 4670         PLP          SEE IF FIRST TWO DIGITS = 0
 4680         BEQ .9       ...YES, SO QUOTIENT IS ALSO ZERO
 4690  *---SUBTRACT WHILE POSSIBLE------
```

```
4700  .5     INC MAC+9    COUNT 1 SUBTRACTION
4710         PHA          DAC LEFT EXTENSION
4720         SEC          DO A TRIAL SUB
4730         LDY #9
4740  .7     LDA DAC.HI,Y
4750         SBC ARG.HI,Y
4760         STA DAC.HI,Y
4770         DEY
4780         BPL .7
4790         PLA          DAC LEFT EXTENSION
4800         SBC #0
4810         BCS .5       NO BORROW
4820  *---OVERSHOT, SO RESTORE---------
4830         LDY #9       BORROW,SO ADD IT BACK IN
4840         CLC
4850  .8     LDA DAC.HI,Y
4860         ADC ARG.HI,Y
4870         STA DAC.HI,Y
4880         DEY
4890         BPL .8
4900         DEC MAC+9    BACK OFF QUOTIENT DIGIT, TOO
4910  *---NEXT DIGIT-------------------
4920  .9     DEX          ALL DIGITS?
4930         BNE .4       ...NOT YET, KEEP GOING
4940  *---ADJUST EXP, CHECK OVERFLOW---
4950         CLD          BINARY MODE
4960         INC DAC.EXPONENT  ADJUST FOR OFFSET
4970         BPL .10      ...NO OVERFLOW PROBLEM
4980         LDA MAC      COULD BE OVERFLOW
4990         AND #$F0
5000         BNE .1       ...OVERFLOW
5010  *---COPY QUOTIENT TO DAC---------
5020  .10    LDY #9
5030  .11    LDA MAC,Y
5040         STA DAC.HI,Y
5050         DEY
5060         BPL .11
5070         JMP NORMALIZE.DAC
5080  *--------------------------------
5090  *   SHIFT 20 DIGITS IN MAC RIGHT ONE PLACE
5100  *--------------------------------
5110  SHIFT.MAC.LEFT.ONE
5120         LDY #4
5130  .1     ASL MAC+9
5140         ROL MAC+8
5150         ROL MAC+7
5160         ROL MAC+6
5170         ROL MAC+5
5180         ROL MAC+4
5190         ROL MAC+3
5200         ROL MAC+2
5210         ROL MAC+1
5220         ROL MAC
5230         DEY
```

```
5240        BNE .1
5250        RTS
5260 *--------------------------------
```

```
=======================================================================
DOCUMENT :AAL-8407:DOS3.3:S.DP18.FIN.txt
=======================================================================

 1000  *SAVE S.DP18 FIN
 1010  *--------------------------------
 1020  *   DP18 INPUT CONVERSION
 1030  *--------------------------------
 1040  FIN    LDA #0       CLEAR WORK AREA
 1050         LDX #WRKSZ-1      (DAC, SGNEXP, EXP,
 1060  .1     STA WORK,X          DGTCNT, & DECFLG)
 1070         DEX
 1080         BPL .1       LEAVE X=$FF WHEN FINISHED
 1090         LDA #$40
 1100         STA DAC.EXPONENT
 1110  *---HANDLE LEADING SIGN----------
 1120         JSR AS.CHRGOT
 1130         BCC .2       IF DIGIT 0-9
 1140         JSR FIN.SIGN ...SEE IF + OR - SIGN
 1150         BNE .4       ...NEITHER + NOR -
 1160         BCC .3       ...+
 1170         STX DAC.SIGN ...-, SET TO $FF
 1180         BCS .3       ...ALWAYS
 1190  *---GET DIGITS TILL NON-DIGIT----
 1200  .2     JSR ACCUMULATE.DIGIT
 1210  .3     JSR AS.CHRGET   GET NEXT CHARACTER
 1220         BCC .2       ...DIGIT
 1230  *---".", "E", OR END-------------
 1240  .4     CMP #'.      DECIMAL POINT?
 1250         BEQ .9       YES
 1260         CMP #'E      LETTER E
 1270         BNE .10      END OF NUMBER
 1280  *---HANDLE EXPONENT FIELD--------
 1290         JSR AS.CHRGET
 1300         BCC .6       ...DIGIT, ASSUME POSITIVE
 1310         JSR FIN.SIGN ...SEE IF + OR - SIGN
 1320         BNE .8       ...NEITHER + NOR -
 1330         BCC .5       ...+
 1340         ROR SGNEXP   ...-, SO SET SGNEXP NEGATIVE
 1350  .5     JSR AS.CHRGET   GET FIRST DIGIT OF EXP
 1360         BCS .8          ...NO DIGITS!
 1370  .6     AND #$0F        ...ISOLATE EXP 1ST DIGIT
 1380         STA EXP
 1390         JSR AS.CHRGET   GET 2ND DIGIT OF EXP, IF ANY
 1400         BCS .8          ...NO MORE DIGITS
 1410         AND #$0F        ISOLATE 2ND DIGIT
 1420         PHA             SAVE ON STACK
 1430         LDA EXP         MULTIPLY 1ST DIGIT BY 10
 1440         ASL
 1450         ASL         (CLEARS CARRY TOO)
 1460         ADC EXP      *5
 1470         ASL          *10  (CARRY STILL CLEAR)
 1480         STA EXP      ADD 2ND DIGIT
```

```
1490        PLA
1500        ADC EXP
1510        STA EXP      2 DIGIT EXP
1520        CMP #64+18   ALLOW .00000000000000001E+82
1530        BCS .7       OR 99999999999999999E-82
1540        JSR AS.CHRGET   GET NEXT CHAR
1550        BCS .8       NO MORE DIGITS
1560 .7     JMP AS.OVRFLW   OVERFLOW ERROR
1570 .8     ASL SGNEXP   CHECK SIGN OF EXP
1580        BCC .10      ...POSITIVE
1590        LDA #0       ...NEGATIVE, SO COMPLEMENT EXP
1600        SBC EXP
1610        JMP .11      ...ALWAYS
1620 *---FOUND DECIMAL POINT----------
1630 .9     ROR DECFLG   SET DECIMAL POINT FLAG
1640        BIT DECFLG   CHECK FOR TWO DECIMAL POINTS
1650        BVC .3       NO
1660 *---COMPUTE FINAL EXPONENT-------
1670 .10    LDA EXP      GET EXPLICIT EXPONENT
1680 .11    CLC
1690        ADC DAC.EXPONENT
1700        LDX DGTCNT   SEE IF ANY SIGNIFICANT DIGITS
1710        BNE .12      ...YES
1720        TXA          ...NO, MAKE EXPONENT ZERO
1730 .12    STA DAC.EXPONENT
1740        TAX          TEST RANGE OF EXPONENT
1750        BMI .13      ...NOT IN RANGE 0...7F
1760        RTS
1770 *---EITHER UNDER- OR OVER-FLOW---
1780 .13    ASL          UNDER, OR OVER?
1790        BCC .7       ...OVERFLOW
1800        LDA #0
1810        STA DAC.SIGN
1820        BEQ .12      ...ALWAYS
1830 *--------------------------------
1840 ACCUMULATE.DIGIT
1850        AND #$0F     ISOLATE DIGIT
1860        BEQ .4       ZERO DIGIT
1870        TAX          SAVE DIGIT IN X-REG
1880        LDA DGTCNT   NO MORE THAN 20 SIGNIFICANT DIGITS
1890        CMP #20
1900        BCS .2       DISCARD EXTRA DIGITS
1910 *---STORE THE DIGIT IN DAC-------
1920        LSR          ODD/EVEN TO CARRY
1930        TAY          INDEX TO Y-REG
1940        TXA          GET DIGIT FROM X-REG
1950        BCS .1       ODD DIGIT ON RIGHT SIDE
1960        ASL          EVEN DIGIT MUST BE SHIFTED
1970        ASL
1980        ASL
1990        ASL
2000 .1     ORA DAC.HI,Y MERGE
2010        STA DAC.HI,Y
2020 *---COUNT THE DIGIT--------------
```

```
2030  .2     INC DGTCNT   COUNT SIGNIFICANT DIGIT
2040         LDA DECFLG   SEE IF IN FRACTION
2050         BMI .3       YES
2060         INC DAC.EXPONENT  NO
2070  .3     RTS
2080  *---DIGIT = 0-------------------
2090  .4     LDA DGTCNT   SEE IF LEADING ZERO
2100         BNE .2       NO
2110         LDA DECFLG   SEE IF PART OF FRACTION
2120         BPL .5       NO, COMPLETELY IGNORE IT
2130         DEC DAC.EXPONENT
2140  .5     RTS
2150  *-------------------------------
2160  *   SCAN + OR - SIGN
2170  *   -------------------
2180  *      +     .EQ., .CC.
2190  *      -     .EQ., .CS.
2200  *    OTHER   .NE.
2210  *-------------------------------
2220  FIN.SIGN
2230         CMP #'-'
2240         BEQ .2
2250         CMP #TKN.MINUS
2260         BEQ .2
2270         CMP #'+'
2280         BEQ .1
2290         CMP #TKN.PLUS
2300  .1     CLC
2310  .2     RTS
```

```
==============================================================
DOCUMENT :AAL-8407:DOS3.3:S.DP18.FstrMult.txt
==============================================================

 1000  *SAVE S.DP18 FASTER MULTIPLY
 1010  *--------------------------------
 1020  * DAC = ARG * DAC
 1030  *--------------------------------
 1040  DMULT  LDA DAC.EXPONENT  IF DAC=0, EXIT
 1050         BEQ .3
 1060         LDA ARG.EXPONENT  IF ARG=0, SET DAC=0 AND EXIT
 1070         BEQ .4
 1080  *---CLEAR RESULT REGISTER--------
 1090         LDA #0
 1100         LDY #19
 1110  .1     STA MAC,Y
 1120         DEY
 1130         BPL .1
 1140  *---FORM PRODUCT OF FRACTIONS----
 1150         JSR MULTIPLY.BY.LOW.DIGITS
 1160         JSR SHIFT.MAC.RIGHT.ONE
 1170         JSR SHIFT.DAC.RIGHT.ONE
 1180         JSR MULTIPLY.BY.LOW.DIGITS
 1190  *---ADD THE EXPONENTS------------
 1200         LDA DAC.EXPONENT
 1210         CLC
 1220         ADC ARG.EXPONENT
 1230         CMP #$C0     CHECK FOR OVERFLOW
 1240         BCS .5       ...OVERFLOW
 1250         SBC #$3F     ADJUST OFFSET
 1260         BMI .4       ...UNDERFLOW
 1270         STA DAC.EXPONENT
 1280  *---FORM SIGN OF PRODUCT---------
 1290         LDA DAC.SIGN
 1300         EOR ARG.SIGN
 1310         STA DAC.SIGN
 1320  *---MOVE MAC TO DAC--------------
 1330         LDY #9
 1340  .2     LDA MAC,Y
 1350         STA DAC.HI,Y
 1360         DEY
 1370         BPL .2
 1380  *---NORMALIZE DAC----------------
 1390         JSR NORMALIZE.DAC
 1400         LDA MAC      IF LEADING DIGIT=0,
 1410         AND #$F0     THEN GET ANOTHER DIGIT
 1420         BNE .3
 1430         LDA MAC+10
 1440         LSR
 1450         LSR
 1460         LSR
 1470         LSR
 1480         ORA DAC.HI+9
```

```
1490         STA DAC.HI+9
1500  .3     RTS
1510  .4     LDA #0
1520         STA DAC.SIGN
1530         STA DAC.EXPONENT
1540         RTS
1550  .5     JMP AS.OVRFLW
1560  *---------------------------------
1570  *      MULTIPLY BY EVERY OTHER DIGIT
1580  *---------------------------------
1590  MULTIPLY.BY.LOW.DIGITS
1600         SED          DECIMAL MODE
1610         LDX #9
1620         LDY #19
1630  .1     LDA DAC.HI,X
1640         AND #$0F     ISOLATE NYBBLE
1650         BEQ .2       0, SO NEXT DIGIT
1660         JSR MULTIPLY.ARG.BY.N
1670  .2     DEY          NEXT MAC POSITION
1680         DEX          NEXT DAC DIGIT
1690         BPL .1       DO NEXT DIGIT
1700         CLD          BINARY MODE
1710         RTS          DONE
1720  *---------------------------------
1730  MULTIPLY.ARG.BY.N
1740         STA DIGIT    N = 1...9
1750         STY TEMP     SAVE Y
1760         STX TEMP+1   SAVE X
1770  .1     LDX #9       INDEX INTO ARG
1780         CLC
1790  .2     LDA ARG.HI,X
1800         ADC MAC,Y    ADD IT
1810         STA MAC,Y
1820         DEY          NEXT MAC
1830         DEX          NEXT ARG
1840         BPL .2       NEXT DIGIT
1850         BCC .4       NO CARRY
1860  .3     LDA #0       PROPAGATE CARRY
1870         ADC MAC,Y
1880         STA MAC,Y
1890         DEY
1900         BCS .3       MORE CARRY
1910  .4     LDY TEMP     GET POSITION IN MAC
1920  .5     DEC DIGIT    NEXT DIGIT
1930         BNE .1
1940         LDX TEMP+1
1950         RTS          DONE
1960  *---------------------------------
1970  SHIFT.MAC.RIGHT.ONE
1980         LDY #4       4 BITS RIGHT
1990  .0     LDX #1       20 BYTES
2000         LSR MAC
2010  .1     ROR MAC,X
2020         INX          NEXT BYTE
```

```
2030        PHP
2040        CPX #20
2050        BCS .2       NO MORE BYTES
2060        PLP
2070        JMP .1
2080 .2     PLP
2090        DEY          NEXT BIT
2100        BNE .0
2110        RTS
2120 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8407:DOS3.3:S.SFPrimesImp.txt
==============================================================================

 1000        .LI MOFF
 1010  *SAVE S.SUPER-FAST PRIMES IMPROVED
 1020        .OR $8000    SAFELY OUT OF WAY
 1030  *--------------------------------
 1040  BASE   .EQ $6000    BASE OF PRIME ARRAY
 1050  BEEP   .EQ $FF3A    BEEP THE SPEAKER
 1060  SQZZZZ .EQ 0,1
 1070  START  .EQ 2
 1080  COUNT  .EQ 4,5
 1090  *--------------------------------
 1100        .MA ZERO
 1110        STA ]1+$000,X
 1120        STA ]1+$100,X
 1130        STA ]1+$200,X
 1140        STA ]1+$300,X
 1150        STA ]1+$400,X
 1160        STA ]1+$500,X
 1170        STA ]1+$600,X
 1180        STA ]1+$700,X
 1190        .EM
 1200  *--------------------------------
 1210  *      MAIN CALLING ROUTINE
 1220  *
 1230  MAIN   LDA #-100   DO 1000 TIMES SO WE CAN MEASURE
 1240        STA COUNT    THE TIME IT TAKES
 1250        LDA /-100
 1260        STA COUNT+1
 1270        JSR BEEP     ANNOUNCE START
 1280  .1     JSR PRIME
 1290        INC COUNT
 1300        BNE .1
 1310        INC COUNT+1
 1320        BNE .1
 1330        JMP BEEP     SAY WE'RE DONE
 1340  *--------------------------------
 1350  *      PRIME ROUTINE
 1360  *      SETS ARRAY STARTING AT BASE
 1370  *      TO $FF IF NUMBER IS NOT PRIME
 1380  *      CHECKS ONLY ODD NUMBERS > 3
 1390  *      INC = INCREMENT OF KNOCKOUT
 1400  *      N = KNOCKOUT VARIABLE
 1410  *--------------------------------
 1420  PRIME
 1430        LDX #0
 1440        TXA          CLEAR WORKING ARRAY
 1450  .1     >ZERO BASE
 1460        >ZERO BASE+$0800
 1470        >ZERO BASE+$1000
 1480        >ZERO BASE+$1800
```

```
1490        INX
1500        BNE .1       NOT FINISHED CLEARING
1530 *--------------------------------
1540        LDA /BASE+4  POINT AT FIRST PRIME-SQUARED
1550        STA SQZZZZ+1      (WHICH IS 3*3=9)
1560        LDA #BASE+4
1570        STA SQZZZZ
1580        LDA #1       POINT AT FIRST PRIME (3)
1590        BNE .4       ...ALWAYS
1600 *--------------------------------
1610 .2     TXA
1620        ASL
1630        ASL
1640        ADC SQZZZZ
1650        STA SQZZZZ
1660        BCC .3
1670        INC SQZZZZ+1
1680 .3     LDA BASE,X   GET A POSSIBLE PRIME
1690        BNE .8       THIS ONE HAS BEEN KNOCKED OUT
1700        TXA
1710 *--------------------------------
1720 .4     STA START
1730        ASL          INC = START*2 + 1
1740        ADC #1
1750        STA .7+1
1760        LDA SQZZZZ+1 MOVE MULT TO N
1770        STA .6+2
1780        LDA SQZZZZ
1790 .5     TAX
1800        BEQ .9       ...SPECIAL CASE FOR X=0
1810 *---STRIKE OUT MULTIPLES---------
1820 .6     STA $FF00,X  REMEMBER THAT N IS REALLY AT .6+2
1830 .7     ADC #*-*     N = N + INC
1840        TAX
1850        BCC .6       DONT'T BOTHER TO ADD, NO CARRY
1860        CLC
1870        INC .6+2      INC HIGH ORDER
1880        BPL .5       ...NOT FINISHED
1890 *--------------------------------
1900        LDX START    GET OUR NEXT KNOCKOUT
1910 .8     INX          POINT AT NEXT ODD NUMBER
1920        CPX #64      UP TO 127
1930        BCC .2       WE'RE DONE IF X>127
1940        RTS
1950 *--------------------------------
1960 .9     LDA .6+2
1970        STA .10+2
1980 .10    STA $FF00
1990        TXA
2000        BEQ .7       ...ALWAYS
2010 *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8407:DOS3.3:S.SWAP.AND.SORT.txt
==============================================================================

```
 1000  *SAVE S.SWAP AND SORT
 1010         .LIST MOFF,CON
 1020  *---------------------------------
 1030         .MA S
 1040         LDX #]1
 1050         LDY #]2
 1060         JSR SWAP
 1070         .EM
 1080  *---------------------------------
 1090         .MA INC
 1100         INC PERM+]1
 1110         LDA PERM+]1
 1120         CMP #6
 1130         BCC :1
 1140         LDA #1
 1150         STA PERM+]1
 1160  :1
 1170         .EM
 1180  *---------------------------------
 1190  *      SWAP (X,Y)
 1200  *---------------------------------
 1210  SWAP   LDA BASE,X
 1220         CMP BASE,Y
 1230         BCC .1
 1240         PHA
 1250         LDA BASE,Y
 1260         STA BASE,X
 1270         PLA
 1280         STA BASE,Y
 1290  .1     RTS
 1300  *---------------------------------
 1310  *      SORT BY SWAPS
 1320  *---------------------------------
 1330  SORT
 1340         .DO 0    CHANGE TO 1 TO SELECT MCDONALD'S LIST
 1350         >S 4,5       MCDONALD'S ORDER
 1360         >S 3,5
 1370         >S 3,4
 1380         >S 1,2
 1390         >S 1,4
 1400         >S 1,3
 1410         >S 2,5
 1420         >S 2,4
 1430         >S 2,3
 1440         .ELSE
 1450         >S 1,4       MY ORDER
 1460         >S 2,5
 1470         >S 1,3
 1480         >S 3,5
```

```
        >S 2,4
        >S 1,2
        >S 2,3
        >S 3,4
        >S 4,5
        .FIN
        RTS
*--------------------------------
BASE    .EQ $500
*--------------------------------
LOAD    LDX #5        COPY PERM LIST TO BASE ON SCREEN
.1      LDA PERM,X
        STA BASE,X
        STA BASE+128,X
        DEX
        BNE .1
        RTS
*--------------------------------
PERM    .HS 000101010101
*--------------------------------
CHECK   LDX #4        CHECK IF LIST IS SORTED
.1      LDA BASE+1,X
        CMP BASE,X
        BCC .2
        DEX
        BNE .1
.2      RTS
*--------------------------------
NEXT    >INC 5        INCREMENT PERM LIST
        BCC .1        EACH BYTE RANGES FROM
        >INC 4        01 TO 05
        BCC .1
        >INC 3
        BCC .1
        >INC 2
        BCC .1
        >INC 1
        BCC .1
        RTS           FINISHED
.1      JSR LOAD      COPY PERMLIST TO SCREEN
        JSR SORT      SORT IT ON THE SCREEN
        JSR CHECK     CHECK IF SORTED
        BCS NEXT      ...SORTED, TRY NEXT SEQUENCE
        RTS           ...NOT SORTED
*--------------------------------
        .MA SS
        LDX #]1
        LDY #]2
        JSR EXCHANGE
        .EM
*--------------------------------
EXCHANGE
        LDA PERM,X
        PHA
```

```
2030        LDA PERM,Y
2040        STA PERM,X
2050        PLA
2060        STA PERM,Y
2070        LDX #1
2080 .1     LDA PERM,X
2090        ORA #$C0
2100        JSR $FDED
2110        INX
2120        CPX #6
2130        BCC .1
2140        LDA #$A0
2150        JSR $FDED
2160        RTS
2170 *---------------------------------
2180        .MA S3
2190   >SS 1,2
2200   >SS 1,3
2210   >SS 1,2
2220   >SS 1,3
2230   >SS 1,2
2240        .EM
2250 *---------------------------------
2260        .MA S4
2270   >S3
2280        JSR $FD8E
2290   >SS 1,4
2300   >S3
2310        JSR $FD8E
2320   >SS 2,4
2330   >S3
2340        JSR $FD8E
2350   >SS 3,4
2360   >S3
2370        JSR $FD8E
2380        .EM
2390 *---------------------------------
2400 PERMUTE
2410        LDX #5
2420 .1     TXA
2430        STA PERM,X
2440        DEX
2450        BNE .1
2460 *---------------------------------
2470   >SS 1,1
2480   >S4
2490   >SS 1,5
2500   >S4
2510   >SS 1,5
2520   >S4
2530   >SS 1,5
2540   >S4
2550   >SS 1,5
2560   >S4
```

```
2570  *--------------------------------
2580         RTS
```

```
==============================================================================
DOCUMENT :AAL-8407:DOS3.3:Sieve.Eratos.1.txt
==============================================================================

 1000  *SAVE SIEVE OF ERATOSTHENES.1
 1010  *--------------------------------
 1020  *   CODED BY PETER J. MCINERNEY, NEW ZEALAND
 1030  *--------------------------------
 1040         .OR $3800
 1050  ARRAY  .EQ $4000
 1060  *--------------------------------
 1070  SIEVE  MOVE    #999,D6   DO 1000 TIMES
 1080  *---CLEAR WORKING ARRAY----------
 1090  .1     MOVE    #ARRAY,A0 CLEAR ARRAY FROM
 1100         MOVE    #$FFF,D0  $4000 TO $7FFF
 1110  .2     CLR.L   (A0)+
 1120         DBF     D0,.2
 1130  *---INIT VARIABLES---------------
 1140         MOVEQ   #3,D0     START AT 3
 1150         MOVEQ   #1,D1     SUM OF ODD NUMBERS
 1160         MOVEQ   #1,D2     COUNT OF ODD NUMBERS
 1170         MOVEQ   #1,D3     USED FOR STRIKING NON-PRIMES
 1180         MOVE    #ARRAY,A0 START OF ARRAY
 1190         BRA.S   .4        JUMP INTO LOOP
 1200  *---START SIFTING----------------
 1210  .3     ADDQ    #1,D2     COUNT ODD NUMBERS
 1220         ADD     D2,D1     GET SUM OF ODDS
 1230  .4     CMPI.B  #0,0(A0,D0) IS THIS A PRIME?
 1240         BNE.S   .6          NO
 1250  *---STRIKE OUT MULTIPLES---------
 1260         MOVE    D1,D4     GET 8*S+1 = N*N
 1270         ASL     #3,D4
 1280         ADDQ    #1,D4
 1290         MOVE    D0,D5     ONLY STRIKE ODD MULTIPLES
 1300         ASL     #1,D5
 1310  .5     MOVE.B  D3,0(A0,D4)  STRIKE ONE
 1320         ADD     D5,D4        NEXT STRIKE
 1330         CMPI    #$4000,D4    ...FINISHED?
 1340         BLS     .5           ...NO
 1350  *---GET NEXT SIEVE SIZE----------
 1360  .6     ADDQ    #2,D0     NEXT ODD NUMBER
 1370         CMPI    #127,D0   UNTIL SQRT $4000-1
 1380         BLS     .3
 1390  *---DO IT ALL 1000 TIMES---------
 1400         DBF     D6,.1   NEXT TIME
 1410         RTS
```

```
===============================================================
DOCUMENT :AAL-8407:DOS3.3:Sieve.Eratos.2.txt
===============================================================

 1000  *SAVE SIEVE OF ERATOSTHENES.2
 1010  *--------------------------------
 1020  *   CODED BY PETER J. MCINERNEY, NEW ZEALAND
 1030  *--------------------------------
 1040         .OR $3800
 1050  ARRAY  .EQ $4000
 1060  *--------------------------------
 1070  SIEVE  MOVE    #999,D6   DO 1000 TIMES
 1080  *---CLEAR WORKING ARRAY----------
 1090  .1     MOVE    #ARRAY,A0 CLEAR ARRAY FROM
 1100         MOVE    #$FF,D0   $4000 TO $7FFF
 1110  .2     CLR.L   (A0)+
 1120         CLR.L   (A0)+
 1130         CLR.L   (A0)+
 1140         CLR.L   (A0)+
 1150         CLR.L   (A0)+
 1160         CLR.L   (A0)+
 1170         CLR.L   (A0)+
 1180         CLR.L   (A0)+
 1190         DBF     D0,.2
 1200  *---INIT VARIABLES---------------
 1210         MOVEQ   #3,D0     START AT 3
 1220         MOVEQ   #4,D4     CORRESPONDS TO 9
 1230         MOVEQ   #4,D2     DELTA
 1240         MOVEQ   #1,D3     USED FOR STRIKING NON-PRIMES
 1250         MOVE    #ARRAY+1,A0 POSITION OF 3
 1260         MOVE    #ARRAY,A1 START OF ARRAY
 1270         BRA.S   .4        JUMP INTO LOOP
 1280  *---START SIFTING----------------
 1290  .3     ADDQ    #4,D2     UPDATE DIFFERENCE
 1300         ADD     D2,D4     UPDATE SQUARE POINTER
 1310  .4     CMPI.B  #0,(A0)+  IS THIS A PRIME?
 1320         BNE.S   .6          NO
 1330  *---STRIKE OUT MULTIPLES---------
 1340         MOVE    D4,D5     GET LATEST SQUARE
 1350  .5     MOVE.B  D3,0(A1,D5)  STRIKE ONE
 1360         ADD     D0,D5        NEXT STRIKE
 1370         CMPI    #$2000,D5    ...FINISHED?
 1380         BLS     .5           ...NO
 1390  *---GET NEXT SIEVE SIZE----------
 1400  .6     ADDQ    #2,D0     NEXT ODD NUMBER
 1410         CMPI    #127,D0   UNTIL SQRT $4000-1
 1420         BLS     .3
 1430  *---DO IT ALL 1000 TIMES---------
 1440         DBF     D6,.1   NEXT TIME
 1450         RTS
```

==============================================================================
DOCUMENT :AAL-8408:Articles:Big.BSAVEs.txt
==============================================================================

Modify DOS 3.3 for Big BSAVEs..............Bob Sander-Cederlof

Jim Sather (author of "Understanding the Apple II" and designer of the QuikLoader card) called today, and one topic of discussion was DOS 3.3's limit of 32767 for the maximum size of a binary file. Jim has been blowing 27256 EPROMs, which are 32768 bytes long. To write a whole EPROMs worth of code on disk it takes two files, because the EPROM holds one more byte than the maximum size file.

The limit doesn't apply if you write the file with the .TF directive in the S-C Macro Assembler, but it is checked when you type in a BSAVE command. The "L" parameter must be less than 32768.

I remembered that somewhere very recently I had read of a quick patch to DOS to remove the restriction. Where? Hardcore Computing? Call APPLE? Washington Apple Pi?

The answer was "yes" to both Call APPLE and W.A.P., because Bruce Field's excellent Apple Doctor column is printed in both magazines. The July 1984 Call APPLE, on page 55, has the answer:

"Sure, change memory location $A964 in DOS from $7F to $FF. From Applesoft this can be done with POKE 43364,255. This changes the range attribute table in DOS to allow binary files as large as 65535 bytes."

By the way, please do not try to BSAVE 65535 bytes on one file. You will not succeed, because doing so will involve saving bytes from the $C000-C0FF range. This is where all the I/O soft switches are, any you will drive your Apple and peripherals wild. And you will not be able to BLOAD it, because it will load on top of the DOS buffers. In general, do not BSAVE any area of RAM which includes $C000-C0FF. Do not BLOAD into the DOS buffers or DOS variables.

If you want to test Bruce's patch, make the patch and then BSAVE filename,A$800,L$8E00. This will save from $800 through $95FF.

```
=============================================================================
DOCUMENT :AAL-8408:Articles:DP18.FOUT.txt
=============================================================================
```

18-Digit Arithmetic, Part 4................Bob Sander-Cederlof

This month we will look at two output conversion routines. The first one always prints in exponential form, while the second one allows setting a field width and number of fractional digits. The routines are written so that the output string may either be printed or fed to an Applesoft string variable.

Let's assume that the value to be printed has already been loaded into the DP18 accumulator, DAC. Lines 1230-1270 describe DAC as a 12-byte variable. The exponent is in the first byte, DAC.EXPONENT. It has a value from $00 to $7F:

```
       $00 means the whole number is zero
       $01 means the power of ten exponent is -63
       $3F means 10^-1
       $40 means 10^0
       $41 means 10^1
       $7F means 10^63
```

The 18 digits of the number, plus two extension digits, are in the next ten bytes in decimal format (each digit takes four bits). The extension is zeroed when you load a fresh value into DAC, but after some computations it holds two more digits to guard against roundoff and truncation problems.

The sign of the number is stored in DAC.SIGN: if the value in DAC.SIGN is from $00 to $7F, the number is positive; if from $80 to $FF, the number is negative.

If you have been following the DP18 series from the beginning, and typing in all the code (or getting it from the Quarterly Disks), then you will realize that to integrate each installment takes some work. In order to print the sections separately, and have them separately readable, I must repeat some variable declarations. The listing this month refers to two previously printed subroutines, DADD and MOVE.YA.ARG. These are simply equated to $FFFF in lines 1030 and 1040, so that the code will assemble. If you really want it to work, you have to remove those two lines and include the code for the subroutines. The fact that three installments have already been printed also somewhat restricts me, because even if I see possible improvements I must be careful not to contradict the code you already have.

Quick Standard Format Conversion

The subroutine QUICK.FOUT which begins on line 1600 converts the contents of DAC to a string in FOUT.BUF in the format

```
sd.dddddddddddddddddEsxx
```

The first s is the sign, which is included only if negative. The d's are a series of up to 18 significant digits (trailing zeroes will not be included). The letter E is always included, to signify the power-of-ten exponent field. The letter s after the E is the sign of the exponent: it is always included, and will be either + or -. The xx is a two-digit exponent, and both digits will always be included. The decimal point will be included only if there are non-zero digits after it. If the number is exactly zero, the string in FOUT.BUF will be simply "0". Here are some more examples:

```
value           string
1000            "1E+03"
.001            "1E-03"
-262564478.5    "-2.625644785E+08"
```

Two processes are involved in converting from DAC to FOUT.BUF. One is the analysis of the DAC contents; the other is the process of storing sequential characters into FOUT.BUF. The latter process is handled in most cases by the subroutine at lines 3720-3820. Entry at STORE.CHAR stores the contents of the A-register in the next position in FOUT.BUF, and increments the position pointer (INDEX). Entry at STORE.DIGIT first converts the value in the A-register to an ASCII digit by setting the high nybble to "3". (The digits 0-9 are $30-$39 in ASCII.)

QUICK.FOUT begins by setting INDEX, the FOUT.BUF position pointer, to 0. At lines 1630-1700 the special case of the value in DAC being exactly zero is tested and handled. If the value in DAC is zero, then DAC.EXPONENT will be zero. (This is a convention throughout DP18, to simplify making values of zero and testing for them.) If the value is zero, ASCII zero is stored in FOUT.BUF, followed by a terminating $00 byte.

If the value is not zero, the next job is to check the sign of the value. Lines 1710-1740 insert a minus sign in FOUT.BUF if the value is negative.

Lines 1760-1910 pull out the 18 digits of the mantissa from DAC.HI through DAC.HI+8. The extension digits are ignored. The code here looks an awfully lot like a routine to convert from hex to ASCII, ignoring the possible hex digits A-F. That is because the digits are four bits each, and ARE like hex digits. Lines 1830-1860 insert the decimal point after the first digit.

Lines 1930-2020 look at the formatted number in FOUT.BUF and trim off the trailing zeroes. If all digits after the decimal point are zero, the decimal point is trimmed off too. If you would rather that QUICK.FOUT always printed exactly 18 digits, trailing zeroes and all, you could cut out these lines.

Lines 2040-2290 format the exponent field. First the letter E is installed in FOUT.BUF. Then lines 2060-2120 install the exponent sign. There is a little adjustment here due to the fact that the value in DAC is in the form ".DDDD" times a power of ten, and we are converting to "D.DDD" times a power. That means the exponent in DAC.EXPONENT is one larger than we will print. The DEY at line 2080 adjusts for this offset.

Lines 2130-2180 get the absolute value of the exponent by removing the $40 bias and taking the 2's complement if the result is negative. Lines 2190-2290 convert the binary value of the exponent to two decimal digits, and insert them into FOUT.BUF. Lines 2300-2310 terminate the FOUT.BUF string with $00.

Once the value has been converted to a string in FOUT.BUF, we can either print it or put it into an Applesoft string variable. The subroutine QUICK.PRINT which begins at line 1370 calls QUICK.FOUT and then prints the characters from FOUT.BUF.

Fancier Formatted Conversion

The second conversion routine, which begins at line 2350, allows you to specify the number of digits to display after the decimal point, and the number of characters in the output field. The value will be formatted into the field against the right end, with leading blanks as necessary to fill the field. The value will be rounded to the number of digits that will be converted. If you are familiar with the FORTRAN language, you will recognize this as the "Fw.d" format. W is the width of the field, and D is the number of fractional digits. Here are some examples:

```
     W  D   value   string
     12 5   1234.56 "  1234.56000"
     12 1   1234.56 "      1234.6"
```

As before, the output string will be stored in FOUT.BUF in ASCII code, terminated by a $00 byte. If the value will not fit into the W.D field you specify, the entire field will be filled with "*" characters.

As listed here, I have set FOUT.BUF as a 41-byte variable. This means the maximum W is 40, leaving room for the terminating $00 byte. If you want longer conversions, simply change line 1060.

FOUT expects the W and D parameters to be in the A- and Y-registers, respectively. Lines 2380-2460 check W and D for legality. If W is larger than FOUT.BUF.SIZE-1, then it is set to that value. We don't want to store converted characters beyond the end of FOUT.BUF! Then if D is larger than W-1, it is pruned back.

Lines 2480-2540 initialize various variables used during the following conversion. Once again, INDEX will point to the position in FOUT.BUF. I could probably have economized some in the use of variables by re-using the same variables for different purposes, but I wanted to keep them separate to make it easier to code and debug.

Line 2560 calls ROUND.DAC.D to round the value in DAC to D digits after the decimal point. This boils down to adding .5 times 10 to the D power to the value in DAC. ROUND.DAC.D, at lines 3860-4000, does just that. First the rounding number is built in ARG, then DADD adds ARG to FAC.

Lines 2570-2610 store a minus sign into SIGN.CHAR if the value in DAC is negative. SIGN.CHAR was initialized to $00 above. If the sign is negative, line 2590 will increment SIGN.SIZE. SIGN.SIZE will either be 0 or 1, and will be used later in determining how many leading blanks are needed. We cannot store the sign character into FOUT.BUF until the leading blanks have been stored.

Lines 2630 to 2710 compute how many digits will be printed before the decimal point (NO.LEADING.DIGITS), and how many zeroes before the first significant digit after the decimal point (NO.LEADING.ZEROES). If the power-of-ten exponent was negative, there will be no leading digits and some leading zeroes; if positive, there will be some leading digits and no leading zeroes. For example,

```
    .2345E-5     .000002345    5 leading zeroes
    .2345E+3     234.5         3 leading digits
```

What if the exponent is more than 18? This would mean more digits might be extracted from DAC than exist, so lines 2730-2790 limit NO.LEADING.DIGITS to 18. NO.INTEGRAL.ZEROES takes up the slack, to print any necessary zeroes between the last significant digit before the decimal point, and the decimal point. For example, if W=25 and D=2, and the value is .1234E+20, we will get NO.LEADING.DIGITS=18 and NO.INTEGRAL.ZEROES=2:

```
       "  12340000000000000000.00"
```

Lines 2810-2870 now calculate the total number of non-blank characters which will be required: one for sign if the sign is negative, all the leading digits and integral zeroes before the decimal point, one for the decimal point itself, and D fractional digits. (Just now I noticed that I could have saved two bytes and two cycles by changing line 2810 from CLC to SEC, and eliminating the ADC #1 at line 2860.)

Lines 2890-2920 compute how many significant digits of fraction will be needed. You specified D digits of fraction, but only DD of them will come from the value in DAC. This will be less than D if there are any leading zeroes.

Lines 2940-2970 check whether the converted number can fit in a W-wide field. If not, Lines 3370-3400 fill the field with stars and exit.

Lines 2980-3030 compute how many leading blanks will be needed to right justify the number in the W-field. There is some hopscotch here because we are going to put "0." in front of numbers that have no integral digits.

At long last, we are ready to begin string characters in FOUT.BUF. Lines 3050-3070 store the leading blanks. A subroutine STORE.N.CHARS does the dirty work. STORE.N.CHARS (lines 3670-3710) expects the character to be stored in the A-register, and the count in the Y-register. It also expects that the Z-status is set according to the count in Y. Thus, if the count is zero, the subroutines returns immediately without storing any characters.

STORE.N.DIGITS, at lines 3440-3620, is quite similar to STORE.N.CHARS. Once again, the count must be in the Y-register and the Z-status should reflect the value in Y. Digits are picked out of the value in DAC using an index DIGIT.PICKER, and stored into FOUT.BUF using STORE.DIGIT.

Lines 3090-3110 store the sign if it is negative. Lines 3120-3210 print whatever digits are needed before the decimal point. This will include leading digits (if any) and integral zeroes (if any), or simply one zero (if neither of the other).

Lines 3230-3320 store the fractional part. This includes the decimal point, leading fractional zeroes (if any), and fractional digits (if any).

Finally, lines 3340-3350 store a terminating $00 at the end of the string in FOUT.BUF.

A subroutine called FORMAT.PRINT at line 1450 calls FOUT and then prints the contents of FOUT.BUF. You could now write a higher level routine, if you wish, which would examine the exponent to determine whether the number would fit in a 20-character field. If not, you could use QUICK.PRINT. If so, use FOUT with W=40 and D=18, and then truncate leading spaces and trailing zeroes. This would give you a complete print routine for any numbers, printing them in simple form when they fit and exponential form when they don't. Indeed, just such a routine already exists in DP18, but will have to wait for a future installment. FOUT can also be used as the base for a complete PRINT USING facility, and that is also already in DP18 waiting for future installments.

Meanwhile, enjoy these two conversions, and experiment with your own.

==============================================================================
DOCUMENT :AAL-8408:Articles:Enbl.Dsbl.IRQ.txt
==============================================================================

Enable/Disable IRQ from Applesoft..........Bob Sander-Cederlof

If you have applied the patches to DOS 3.3 that we published in the January 1984 issue (pages 10,11), and if you now are using interrupts from such sources as the Timemaster II or a handy pushbutton, you have probably run into the need to enable and disable IRQ from within an Applesoft program. (That sentence is the kind you have to read without interruption, so I really should have begun the paragraph with SEI and ended it with CLI.)

What is need is four bytes of assembly language, at a location that you can CALL. For example:

```
       300- 58    CLI
       301- 60    RTS
       302- 78    SEI
       303- 60    RTS
```

If those four bytes are in memory as shown, you can CALL 768 to enable IRQ interrupts, and CALL 771 to disable them. You can install the four bytes like this:

```
       100 POKE 768,88: POKE 769,96
       110 POKE 770,120:POKE 771,96
```

Now there are often times when poking into page 3 is not possible. Are there other tricky ways to get those bytes installed, without using page 3?

I found a half dozen or so. First, realize that the four bytes only need to be there when you call them. The rest of the time the same locations could be used for other purposes. For example, we could poke them into the input buffer at $200, as long as we do it every time we CALL it:

```
       100 POKE 512,88:POKE 513,96:CALL 512
             to enable interrupts, or

       500 POKE 512,120:POKE 513,96:CALL 512
             to disable them.
```

The result of a multiplication or division is left, sometimes normalized and sometimes not, in $62...$66. If we find two numbers whose product leaves the bytes $58 and $60 at $62 and $63, we could CALL 98:

```
       100 X = 1*707 : CALL 98 : REM ENABLE IRQ
       200 X = 1*963 : CALL 98 : REM DISABLE IRQ
```

On the next page is a table showing the various methods I found:

```
Enable (CLI..RTS)      Disable (SEI..RTS)
-------------------------------------------------------
100 POKE 38,88            100 POKE 38,120
110 POKE 39,96            110 POKE 39,96
120 CALL 38               120 CALL 38
-------------------------------------------------------
100 CALL 8411232-8411065  100 CALL 8419424-8419257
-------------------------------------------------------
100 GOSUB 24664           100 GOSUB 24696
...                       ...
24664 CALL 117:RETURN     24696 CALL 117:RETURN
-------------------------------------------------------
100 X = 1*707 : CALL 98   100 X = 1*963 : CALL 98
-------------------------------------------------------
100 X = RND(-8411323.5)   100 X = RND(-8419424.5)
110 CALL 203              110 CALL 203
-------------------------------------------------------
100 HOME:FLASH:PRINT"X "  100 HOME:FLASH:PRINT"8 "
110 NORMAL:CALL1024       110 NORMAL:CALL1024
-------------------------------------------------------
```

Can you figure out how all these work? They are pretty tricky! Can you think of some more?

```
==================================================================
DOCUMENT :AAL-8408:Articles:Front.Page.txt
==================================================================
```

$1.80

Volume 4 -- Issue 11                    August, 1984

In This Issue...



New Cross Assemblers Available

We have recently added three new cross assemblers to the S-C Macro family.

```
     Intel Z-8..................$32.50
     General Instruments 1650....$50.00
     General Instruments 1670....$50.00
```

Unlike previous cross assemblers, which were based on Version 1.0 of the S-C Macro Assembler, these are based on Version 1.1.  This means 80-column support for the Videx, STB-80, and Apple //e-//c 80-column, as well as standard 40-column.  It also adds certain directives and fixes some problems which were in version 1.0.

We have also been hard at work generating Version 2.0 of the S-C Macro Assembler.  It will be ready soon, complete with a brand new manual. It will support all the new opcodes and address modes of the 65C02, 65802, and 65816 processors.  Owners of older versions of the S-C Assemblers will be able to upgrade for a very reasonable fee.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
==============================================================================
DOCUMENT :AAL-8408:Articles:LCR.Diagram.txt
==============================================================================

One of
64 hash                  Calling Line Lists
pointers

  ----     ---------     ----
 |    |--->|    |    |--->|    |--->0
  ----     ---------     ----

           ---------     ----      ----      ----
          |    |    |--->|    |--->|    |--->|    |--->0
           ---------     ----      ----      ----
Called
Line
Chain      ---------
          |    |    |--->0
           ---------

           ---------     ----
          |    |    |--->|    |--->0
           ---------     ----

Found a Call

    * Use high 6 bits of called line number to index Hash Table
    * Get pointer from Hash Table to find start of chain
         * If no pointer in Hash Table, make new entry
    * Search chain for same line number
         * If not found, make new link in chain
         * If found, search calling line list
    * Enter new calling line in list
```

=============================================================================
DOCUMENT :AAL-8408:Articles:LCR.txt
=============================================================================

Line Number Cross Reference.........................Bill Morgan

Have you ever had to modify a BASIC program written by someone who didn't seem to know what he was doing? Deciphering several hundred undocumented lines of split FOR/NEXTs and tangled GOTOs can lead to a severe headache. We recently had a consulting job that involved just such a project: one program to be altered was about a hundred sectors of spaghetti-plate Applesoft. A couple of the biggest problems were figuring out which lines used a particular variable, and what lines called others, or were called from where.

Back in November of 1980, AAL published a Variable Cross Reference program which neatly took care of the first problem by producing a listing in alphabetical order of all the variables used and all the lines using them. At the end of that article, Bob S-C pointed out that the program could, with some effort, be modified into just the sort of Line Number Cross Reference we now needed. Well, I drew the job of making that modification, and here's what I came up with.

The Basis

These Cross Reference programs use a hash-chain data structure to store the called and calling line numbers. Each called line has its own list of lines which refer to it. We locate these lists by using the upper six bits of the line number for an index into a table located at $280. This table contains the address of the beginning of each of the 64 possible chains. Each chain is made up of the data for a range of 1024 possible called line numbers. The first one has called lines 0-1023, the second has 1024-2047, and so on.

The entry for each called line is made up of a pointer to the next called line in that chain, this called line number, a pointer to the next calling line, and the number of this calling line. Each subsequent calling line entry has only the last four bytes. A pointer with a value of zero marks the end of each chain and each list.

VCR used three characters for each variable: the first two letters of the variable name and a type designator of "$", "%" or " ". The first character was the hash index and the last two characters were stored at the beginning of each variable's chain. LCR uses the high-order 6 bits of the called line number for the hash index and stores both bytes of the number in the chain. This is slightly redundant, so if you want to store more information about the called line, you can use the upper six bits of the chain entry.

VCR stored the calling line numbers with the high byte first, backwards from usual 6502 practice. This was done so the same search-compare code could handle both variable names and line numbers. To simplify the conversion I kept the same structure, even though it's no longer strictly necessary.

The Program

LCR, the overall control level, is identical to VCR and just calls the other routines.

INITIALIZATION prepares a couple of pointers and zeroes the hash table.  The only difference here is the size of the hash table.

PROCESS.LINE is also the same as in VCR.  This routine steps through the lines of the Applesoft program, moving the calling line number into our data area and JSRing to SCAN.FOR.CALLS to work on each line.

SCAN.FOR.CALLS is the first really new section of code.  We start by setting a flag used to mark ON ... GO statements.  Then we step through the bytes of the line, looking for tokens that call another line.  GOTO and GOSUB are processed immediately.  For a THEN token we check to see if the next character is a number.  If it is, we deal with it; if not, we go on.  If we find an ON token, we set the flag and keep looking.  After a GOTO or GOSUB we check ONFLAG.  If there was an ON, we look for a comma to mark another called line number.

PROCESS.CALL first converts the ASCII line number of the called line into a two-byte binary number and then searches the data structure for that line.  If it is there, we simply add this calling line to the list.  If we don't find the called line we create a new entry for it.

CONVERT.LINE.NUMBER is lifted straight from Applesoft's LINGET, at $DA0C.

NEXT.CHAR is a utility routine to get the next byte from the program and advance the pointer.

SEARCH.CALL.TABLE starts the search pointer on the appropriate chain.

CHAIN.SEARCH uses the pointer in an entry to step to the next entry. If the pointer is zero, then there is no next entry and the search has failed.  We then compare the line number in the entry to the one we're looking for.  If the entry is less than the search key, we go on.  If it is equal, we update the pointer and report success.  If we hit an entry greater than the key, the search fails and we return.

SEARCH.LINE.CHAIN is called after SEARCH.CALL.TABLE has found a match. Here we move the pointer to the calling line field of the matching entry and use the current calling line for a search key.

ADD.NEW.ENTRY first updates the pointers in the previous entry and this new entry, and the end-of-table pointer.  We then make sure there is room for the new entry and move the data up into the new space.

Now we are done with the routines devoted to building the Cross Reference tables.  Interestingly, SEARCH.CALL.TABLE, CHAIN.SEARCH,

SEARCH.LINE.CHAIN, and ADD.NEW.ENTRY are the real heart of this program, and the only change I had to make in these routines from VCR to LCR was to alter the method of figuring the hash index in SEARCH.CALL.TABLE. Next we come to getting the data back out of the tables and onto a display.

PRINT.REPORT first sets a pointer we'll be using later on and then steps through the hash table, calling PRINT.CHAIN for each entry found.

PRINT.CHAIN starts out by checking for a pause or abort signal from the keyboard. It then moves the current called line number into LINNUM, checks to see if it really exists, and prints it, followed by an asterisk if it is undefined. Now we move a pointer up to the start of the calling line list and call PRINT.LINNUM.CHAIN to display all the entries. The last step is to move the pointer up to the next called line in this chain, if any, and go back to do that one.

CHECK.DEFINITION keeps its own pointer into the program and steps along checking each called line to see if it actually exists. It provides a space or an asterisk to be printed after the line number.

PRINT.LINNUM.CHAIN displays the calling lines stored for each called line. We first tab to the next column (or line if necessary), then get the line number out of the list and print it. Lastly, we move the pointer up to the next entry, if any, and loop back.

TAB.NEXT.COLUMN prints enough blanks to move over to the next output position. If a new line is necessary, it checks the line number to see if the new line should go to the screen only, or also to a printer. This is Louis Pitz's addition, designed to automatically handle either 40- or 80-column output.

PRINT.LINE.NUMBER and CHECK.FOR.PAUSE are pretty standard routines to convert a two-byte binary number into five decimal characters, and to provide for pause/abort during display.

Well, now we have a Line Number Cross Reference to go along with the Variable Cross Reference. Now all that remains is to integrate the two programs into one master Applesoft Cross Reference Utility. Maybe you could call it with "&V" for VCR, or "&L" for LCR, and simply "&" to get both listings. Any takers out there?

PS: Bob suggested that I add a diagram of the hash chain structure, and a summary of the search process. OK, here they are...

```
==============================================================================
DOCUMENT :AAL-8408:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler Version 1.0.......................................$80
S-C Macro Assembler Version 1.1....................................$92.50
Version 1.1 Update.................................................$12.50
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code)........................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks...............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $75)  $65
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $35)  $30
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36
Aztec C Compiler System (Manx Software)..................(reg. $199) $180

Blank Diskettes (Verbatim)............2.50 each, or package of 20 for $45
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                          or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  6 cents each
ZIF Game Socket Extender (Ohm Electronics) ...........................$20
QuikLoader EPROM System (SCRG)...............................($179) $170

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler.................($22.95)  $21
    "Understanding the Apple II", Sather...................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster...........($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner...................($19.95)  $18
```

```
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
    "Microcomputer Design & Troubleshooting", Zumchak.......($17.95)  $17
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

==============================================================================
DOCUMENT :AAL-8408:Articles:Slow.Chips.txt
==============================================================================

Speaking of Slow Chips.......................Robert H. Bernard

William O'Ryan's article (AAL June 1984) about making the 65C02 work in II+s reminds me of some other slow chip problems I have had in the past with Apples.

Years ago, I had a problem with an SSM AIO card in an Apple that traced to a slow 74LS138 at position H2. The symptom was that every few hours the program would fly off into the weeds. I traced it to the device select for the slot, which caused the data on the bus to be late for ROM program fetches from the card. I was able to fix the problem in that case by swapping H2 with another '138 from a different (less critical) position.

Some time later I was able to fix a problem in another machine by swapping the ROM SELECT chip at position F12 (another 74LS138) with another '138. There are apparently many marginal timing situations in II+s, and they are not necessarily in the oldest ones.

All this slow circuit stuff has some interesting side effects. I personally had a number of conversations with SSM about this problem before I found the real cause, and all they could suggest was a capacitor on the clock line. Even after I found the problem, the SSM people I talked to seemed uninterested in the fix, perhaps because they couldn't apply it directly to their product.

The unfortunate end result was that a number of organizations that previously sold or recommended AIO cards stopped doing so. A domino effect was that our local retailer stopped pushing Anadex printers (which required the DTR signal, at that time only available on the AIO) rather than find another serial card to replace the AIO. I always wondered if the Anadex people noticed the effect on their sales....

```
==========================================================================
DOCUMENT :AAL-8408:DOS3.3:S.DP18.FOUT.txt
==========================================================================

 1000  *SAVE S.DP18 FOUT
 1010  *--------------------------------
 1020  AS.COUT       .EQ $DB5C
 1030  DADD          .EQ $FFFF
 1040  MOVE.YA.ARG   .EQ $FFFF
 1050  *--------------------------------
 1060  FOUT.BUF      .BS 41
 1070  FOUT.BUF.SIZE .EQ *-FOUT.BUF
 1080  *--------------------------------
 1090  W          .BS 1
 1100  D          .BS 1
 1110  INDEX      .BS 1
 1120  SIGN.SIZE  .BS 1
 1130  SIGN.CHAR  .BS 1
 1140  ZERO.CHAR  .BS 1
 1150  WW         .BS 1
 1160  DD         .BS 1
 1170  DIGIT.PICKER        .BS 1
 1180  NO.LEADING.ZEROES   .BS 1
 1190  NO.LEADING.DIGITS   .BS 1
 1200  NO.INTEGRAL.ZEROES  .BS 1
 1210  NO.LEADING.BLANKS   .BS 1
 1220  *--------------------------------
 1230  DAC           .BS 12
 1240  DAC.EXPONENT  .EQ DAC
 1250  DAC.HI        .EQ DAC+1
 1260  DAC.EXTENSION .EQ DAC+10
 1270  DAC.SIGN      .EQ DAC+11
 1280  *--------------------------------
 1290  ARG           .BS 12
 1300  ARG.EXPONENT  .EQ ARG
 1310  ARG.HI        .EQ ARG+1
 1320  ARG.EXTENSION .EQ ARG+10
 1330  ARG.SIGN      .EQ ARG+11
 1340  *--------------------------------
 1350  *   QUICK PRINT
 1360  *--------------------------------
 1370  QUICK.PRINT
 1380         JSR QUICK.FOUT
 1390         JMP FOR.PRINT.1
 1400  *--------------------------------
 1410  *   FORMATTED PRINT
 1420  *      (A)=WIDTH OF FIELD
 1430  *      (Y)=# OF FRACTIONAL DIGITS
 1440  *--------------------------------
 1450  FORMAT.PRINT
 1460         LDX #'0      USE ZEROES BEFORE FRACTION
 1470         STX ZERO.CHAR
 1480         JSR FOUT
```

```
*--------------------------------
FOR.PRINT.1
       LDY #0
.1     LDA FOUT.BUF,Y
       BEQ .2
       JSR AS.COUT
       INY
       BNE .1       ...ALWAYS
.2     RTS
*--------------------------------
*      QUICK CONVERSION
*--------------------------------
QUICK.FOUT
       LDY #0
       STY INDEX
       LDA DAC.EXPONENT
       BNE .0       NUMBER IS NOT ZERO
       INC INDEX
       STY FOUT.BUF+1
       LDA #'0
       STA FOUT.BUF  MAKE IT '0'
       RTS
.0     LDA DAC.SIGN
       BPL .1
       LDA #'-      NEGATIVE
       JSR STORE.CHAR
*--------------------------------
.1     LDA DAC.HI,Y NEXT BYTE OF #
       PHA
       LSR
       LSR
       LSR
       LSR
       JSR STORE.DIGIT
       CPY #0
       BNE .2
       LDA #'.      PUT DECIMAL POINT
       JSR STORE.CHAR
.2     PLA          DO 2ND DIGIT
       JSR STORE.DIGIT
       INY
       CPY #9       8 MORE BYTES
       BCC .1
*--------------------------------
       LDY INDEX    TRUNCATE TRAILING ZEROS
.3     DEY
       LDA FOUT.BUF,Y
       CMP #'0
       BEQ .3       DONE
       CMP #'.      TRAILING DECIMAL PT?
       BNE .4       NO
       DEY          YES, DELETE IT
.4     INY
       STY INDEX    SAVE NEW END OF NUMBER
```

```
2030  *--------------------------------
2040         LDA #'E
2050         JSR STORE.CHAR E FOR EXPONENT
2060         LDA #'+
2070         LDY DAC.EXPONENT
2080         DEY
2090         CPY #$40
2100         BCS .5
2110         LDA #'-
2120  .5     JSR STORE.CHAR
2130         TYA          EXPONENT
2140         SEC
2150         SBC #$40     REMOVE OFFSET
2160         BPL .6
2170         EOR #$FF
2180         ADC #1
2190  .6     LDX #'0-1
2200         SEC
2210  .8     INX
2220         SBC #10
2230         BCS .8
2240         ADC #'9+1
2250         PHA
2260         TXA
2270         JSR STORE.CHAR
2280         PLA
2290         JSR STORE.CHAR
2300         LDA #0
2310         JMP STORE.CHAR
2320  *--------------------------------
2330  *   FORMATTED CONVERSION
2340  *      (A)=WIDTH OF FIELD
2350  *      (Y)=# OF FRACTIONAL DIGITS
2360  *--------------------------------
2370  FOUT
2380         CMP #FOUT.BUF.SIZE    LIMIT WIDTH
2390         BCC .1
2400         LDA #FOUT.BUF.SIZE-1
2410  .1     STA W
2420         CPY W        FORCE D<W
2430         BCC .2
2440         TAY
2450         DEY
2460  .2     STY D
2470  *--------------------------------
2480         LDA #0
2490         STA INDEX
2500         STA SIGN.SIZE
2510         STA SIGN.CHAR
2520         STA NO.INTEGRAL.ZEROES
2530         STA NO.LEADING.ZEROES
2540         STA DIGIT.PICKER
2550  *--------------------------------
2560         JSR ROUND.DAC.D   ROUND TO D DIGITS
```

```
2570        LDA DAC.SIGN
2580        BPL .3
2590        INC SIGN.SIZE
2600        LDA #'-      MINUS SIGN
2610        STA SIGN.CHAR
2620 *--------------------------------
2630 .3     SEC
2640        LDA DAC.EXPONENT
2650        SBC #$40     REMOVE OFFSET
2660        BPL .4
2670        EOR #$FF
2680        STA NO.LEADING.ZEROES
2690        INC NO.LEADING.ZEROES
2700        LDA #0
2710 .4     STA NO.LEADING.DIGITS
2720 *--------------------------------
2730        SEC
2740        LDA NO.LEADING.DIGITS
2750        SBC #18
2760        BMI .5
2770        STA NO.INTEGRAL.ZEROES
2780        LDA #18      18 SIGNIF DIGITS MAX
2790        STA NO.LEADING.DIGITS
2800 *--------------------------------
2810 .5     CLC          CALCULATE TOTAL # OF DIGITS
2820        LDA SIGN.SIZE
2830        ADC NO.LEADING.DIGITS
2840        ADC NO.INTEGRAL.ZEROES
2850        ADC D
2860        ADC #1
2870        STA WW
2880 *--------------------------------
2890        SEC
2900        LDA D
2910        SBC NO.LEADING.ZEROES
2920        STA DD
2930 *--------------------------------
2940        SEC
2950        LDA W
2960        SBC WW
2970        BMI .14      ...OVERFLOW
2980        STA NO.LEADING.BLANKS
2990        LDA NO.LEADING.DIGITS
3000        BNE .6
3010        DEC NO.LEADING.BLANKS
3020        BPL .6
3030        INC NO.LEADING.BLANKS IT WENT -, MAKE 0
3040 *---STORE LEADING BLANKS---------
3050 .6     LDA #' '     BLANK
3060        LDY NO.LEADING.BLANKS
3070        JSR STORE.N.CHARS
3080 *---STORE SIGN-------------------
3090        LDA SIGN.CHAR
3100        BEQ .8
```

```
3110         JSR STORE.CHAR
3120  *---STORE INTEGRAL DIGITS--------
3130  .8     LDY NO.LEADING.DIGITS
3140         BEQ .10
3150         JSR STORE.N.DIGITS
3160         BEQ .11      ...ALWAYS
3170  .10    LDA ZERO.CHAR  NO INTEGER PART,SO PRINT 0
3180         JSR STORE.CHAR
3190  .11    LDA #'0
3200         LDY NO.INTEGRAL.ZEROES
3210         JSR STORE.N.CHARS
3220  *---STORE FRACTION---------------
3230         LDA #'.
3240         JSR STORE.CHAR
3250         LDA DD
3260         ORA NO.LEADING.ZEROES
3270         BEQ .13
3280         LDA ZERO.CHAR
3290         LDY NO.LEADING.ZEROES
3300         JSR STORE.N.CHARS
3310         LDY DD
3320         JSR STORE.N.DIGITS
3330  *---TERMINATE STRING-------------
3340  .13    LDA #0
3350         JMP STORE.CHAR
3360  *--------------------------------
3370  .14    LDA #'*'      FILL FIELD WITH STARS
3380         LDY W
3390         JSR STORE.N.CHARS
3400         JMP .13
3410  *--------------------------------
3420  *   STORE NEXT (Y) DIGITS
3430  *--------------------------------
3440  SND..1 LDA DIGIT.PICKER
3450         CMP #20
3460         BCC .1
3470         LDA #0
3480         BEQ .2       ...ALWAYS
3490  .1     LSR          LEFT/RIGHT --> C
3500         TAX          INDEX --> X
3510         INC DIGIT.PICKER
3520         LDA DAC.HI,X
3530         BCS .2
3540         LSR
3550         LSR
3560         LSR
3570         LSR
3580  .2     JSR STORE.DIGIT
3590         DEY
3600  STORE.N.DIGITS
3610         BNE SND..1
3620         RTS
3630  *--------------------------------
3640  *   STORE (Y) OF THE CHARACTER IN (A)
```

```
3650  *      (Z-STATUS IF COUNT IS 0)
3660  *-------------------------------
3670  SNC..1 JSR STORE.CHAR
3680         DEY
3690  STORE.N.CHARS
3700         BNE SNC..1
3710         RTS
3720  *-------------------------------
3730  *      STORE A CHAR IN THE BUFFER
3740  *-------------------------------
3750  STORE.DIGIT
3760         AND #$0F
3770         ORA #'0'
3780  STORE.CHAR
3790         LDX INDEX
3800         STA FOUT.BUF,X
3810         INC INDEX
3820         RTS
3830  *-------------------------------
3840  *      ROUND DAC TO (D) DECIMAL PLACES
3850  *-------------------------------
3860  ROUND.DAC.D
3870         LDA DAC.SIGN  GET THE SIGN
3880         PHA          SAVE IT
3890         LDA #CON.1HALF
3900         LDY /CON.1HALF
3910         JSR MOVE.YA.ARG   MOVE .5*10^-D INTO ARG
3920         PLA          GET SIGN
3930         STA ARG.SIGN
3940         LDA D        GET # OF PLACES
3950         EOR #$FF     MAKE IT NEGATIVE BY 2S COMPLEMENT
3960         SEC          ADD 1 DURING NEXT ADD
3970         ADC #$40     ADD OFFSET
3980         STA ARG.EXPONENT
3990         JMP DADD     ADD .5*10^-D;FOUT WILL TRUNCATE IT
4000  *-------------------------------
4010  CON.1HALF .HS 40500000000000000000
```

==============================================================================
DOCUMENT :AAL-8408:DOS3.3:S.DP18.PackUn.txt
==============================================================================

```
 1000  *SAVE S.DP18 PACK & UNPACK
 1010  *------------------------------
 1020  *     ADDRESSES INSIDE APPLESOFT
 1030  *------------------------------
 1040  AS.OVRFLW .EQ $E8D5    OVERFLOW ERROR
 1050  *--------------------------------
 1060  *      PAGE ZERO USAGE
 1070  *------------------------------
 1080  PNTR   .EQ $5E,5F
 1090  *------------------------------
 1100  *      MOVE (Y,A) INTO DAC AND UNPACK
 1110  *--------------------------------
 1120  MOVE.YA.DAC
 1130         STA PNTR
 1140         STY PNTR+1
 1150         LDY #9       MOVE 10 BYTES
 1160  .1     LDA (PNTR),Y
 1170         STA DAC,Y
 1180         DEY
 1190         BPL .1
 1200         INY          Y=0
 1210         STY DAC.EXTENSION
 1220         LDA DAC.EXPONENT
 1230         STA DAC.SIGN
 1240         AND #$7F
 1250         STA DAC.EXPONENT
 1260         RTS
 1270  *--------------------------------
 1280  *      MOVE (Y,A) INTO ARG AND UNPACK
 1290  *--------------------------------
 1300  MOVE.YA.ARG
 1310         STA PNTR
 1320         STY PNTR+1
 1330         LDY #9       MOVE 10 BYTES
 1340  .1     LDA (PNTR),Y
 1350         STA ARG,Y
 1360         DEY
 1370         BPL .1
 1380         INY          Y=0
 1390         STY ARG.EXTENSION
 1400         LDA ARG.EXPONENT
 1410         STA ARG.SIGN
 1420         AND #$7F
 1430         STA ARG.EXPONENT
 1440         RTS
 1450  *--------------------------------
 1460  *      PACK AND MOVE DAC TO (Y,A)
 1470  *--------------------------------
 1480  MOVE.DAC.YA
```

```
1490        STA PNTR
1500        STY PNTR+1
1510        JSR ROUND.DAC
1520        LDA DAC.EXPONENT
1530        BIT DAC.SIGN
1540        BPL .1        POSITIVE
1550        ORA #$80      NEGATIVE
1560 .1     LDY #0
1570 .2     STA (PNTR),Y
1580        INY
1590        LDA DAC,Y
1600        CPY #10
1610        BCC .2
1620        RTS
1630 *--------------------------------
1640 *      ROUND DAC BY EXTENSION
1650 *--------------------------------
1660 ROUND.DAC
1670        LDA DAC.EXTENSION
1680        CMP #$50      COMPARE TO .5
1690        BCC .3        NO NEED TO ROUND
1700        LDY #8
1710        SED           DECIMAL MODE
1720 .1     LDA #0
1730        ADC DAC.HI,Y
1740        STA DAC.HI,Y
1750        BCC .2        NO NEED FOR FURTHER PROPAGATION
1760        DEY
1770        BPL .1        ...MORE BYTES
1780        INC DAC.EXPONENT
1790        BMI .4        ...OVERFLOW
1800        LDA #$10      .999...9 ROUNDED TO 1.000...0
1810        STA DAC.HI
1820 .2     CLD
1830 .3     LDA #0
1840        STA DAC.EXTENSION
1850        RTS
1860 .4     CLD
1870        JMP AS.OVRFLW
1880 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8408:DOS3.3:S.LCR.txt
==============================================================================

 1000  *SAVE S.LCR
 1010  *---------------------------------
 1020  *    LINE NUMBER CROSS REFERENCE
 1030  *       FOR APPLESOFT PROGRAMS
 1040  *
 1050  *       Based on Variable Cross Reference
 1060  *       Original by Bob S-C 11/80
 1070  *       Modified by Louis Pitz 8/83
 1080  *       Adapted by Bill Morgan 8/84
 1090  *---------------------------------
 1100         .OR $6000
 1110  *      .TF B.LCR
 1120  *---------------------------------
 1130         LDA #$4C     set & vector
 1140         STA $3F5
 1150         LDA #LCR
 1160         STA $3F6
 1170         LDA /LCR
 1180         STA $3F7
 1190         RTS
 1200  *---------------------------------
 1210  TEMP    .EQ $15
 1220  COUNTER .EQ $16
 1230  ONFLAG  .EQ $17          ON ... GO flag
 1240  DEFFLAG .EQ $17
 1250  PNTR    .EQ $18,19       pointer into program
 1260  LZFLAG  .EQ $1A          leading zero flag
 1270  DATA    .EQ $1A thru $1D
 1280  NEXTLN  .EQ $1A,1B       address of next line
 1290  LINNUM  .EQ $1C,1D       current line number
 1300  STPNTR  .EQ $1E,1F       pointer into call table
 1310  TPTR    .EQ $9B,9C       temp pointer
 1320  ENTRY   .EQ $9D thru $A4  8 bytes
 1330  CALL    .EQ ENTRY+2
 1340  SIZE    .EQ $A5,A6
 1350  HSHTBL  .EQ $280
 1360  *---------------------------------
 1370  PRGBOT  .EQ $67,68       beginning of program
 1380  LOMEM   .EQ $69,6A       beginning of variable space
 1390  EOT     .EQ $6B,6C       end of variable table
 1400  *---------------------------------
 1410  COMMA      .EQ ',
 1420  CR         .EQ $8D
 1430  TKN.GOTO   .EQ $AB
 1440  TKN.GOSUB  .EQ $B0
 1450  TKN.ON     .EQ $B4
 1460  TKN.THEN   .EQ $C4
 1470  *---------------------------------
 1480  MON.CH     .EQ $24
```

```
KEYBOARD   .EQ $C000
STROBE     .EQ $C010
AS.MEMFULL .EQ $D410
MON.PRBL2  .EQ $F94A
MON.CROUT  .EQ $FD8E
MON.COUT   .EQ $FDED
MON.COUT1  .EQ $FDF0
*--------------------------------
LCR    JSR INITIALIZATION
.1     JSR PROCESS.LINE
       BNE .1            until end of program
       JSR PRINT.REPORT
       JSR INITIALIZATION  erase call table
       LDA #0            clear $A4 so Applesoft
       STA $A4           will work correctly
       RTS
*--------------------------------
INITIALIZATION
       LDA LOMEM         start call table
       STA EOT           after program
       LDA LOMEM+1
       STA EOT+1
       LDX #$80          # of bytes for hash pointers
       LDA #0
.1     STA HSHTBL-1,X
       DEX
       BNE .1
       LDA PRGBOT        start pointer at
       STA PNTR          beginning of program
       LDA PRGBOT+1
       STA PNTR+1
       RTS
*--------------------------------
PROCESS.LINE
       LDY #3            capture pointer and line #
.1     LDA (PNTR),Y
       STA DATA,Y
       DEY
       BPL .1
       LDA DATA+1        check if end
       BEQ .3            yes, return .EQ.
       CLC
       LDA PNTR          adjust pointer to
       ADC #4            skip over data
       STA PNTR
       BCC .2
       INC PNTR+1
.2     JSR SCAN.FOR.CALLS
       LDA DATA          point to next line
       STA PNTR
       LDA DATA+1        and return .NE.
       STA PNTR+1
.3     RTS
*--------------------------------
```

```
SCAN.FOR.CALLS
       LDA #$FF
       STA ONFLAG
.1     JSR NEXT.CHAR
       BEQ .4
       CMP #TKN.THEN     scan for call token
       BEQ .2
       CMP #TKN.GOTO
       BEQ .3
       CMP #TKN.GOSUB
       BEQ .3
       CMP #TKN.ON
       BNE .1            no match, keep going
       LSR ONFLAG        set flag for ON token
       BPL .1         ...always

.2     LDY #0            after THEN, check
       LDA (PNTR),Y      for line number
       CMP #'0
       BCC .1            <0 isn't
       CMP #'9+1
       BCS .1            neither is >9

.3     JSR PROCESS.CALL  handle this call
       LDA ONFLAG        are we in ON?
       BMI SCAN.FOR.CALLS no, go on
       JSR NEXT.CHAR     yes, look for comma
       BEQ .4            EOL
       CMP #COMMA
       BEQ .3            comma, get another call
       BNE SCAN.FOR.CALLS ...always

.4     RTS
*---------------------------------
PROCESS.CALL
       JSR CONVERT.LINE.NUMBER
       JSR SEARCH.CALL.TABLE
       BCC .2            found same call
       LDA #0
       STA ENTRY+4       start of line number chain
       STA ENTRY+5
       LDA LINNUM+1      MSB first
       STA ENTRY+6
       LDA LINNUM
       STA ENTRY+7
       LDA #8            add 8 byte entry
.1     JMP ADD.NEW.ENTRY

.2     JSR SEARCH.LINE.CHAIN
       BCC .3            found same line number
       LDA #4            add 4 byte entry
       BNE .1         ...always

.3     RTS
```

```
2570  *--------------------------------
2580  CONVERT.LINE.NUMBER
2590         LDA #0
2600         STA CALL+1
2610         STA CALL
2620  .1     JSR NEXT.CHAR
2630         BEQ .2           EOL
2640         SEC
2650         SBC #'0          make value
2660         BCC .2           <0 isn't number
2670         CMP #9+1
2680         BCS .2           >9 isn't number
2690         PHA              save value
2700         LDA CALL
2710         STA TEMP
2720         LDA CALL+1       multiply CALL * 10
2730         ASL
2740         ROL TEMP
2750         ASL
2760         ROL TEMP
2770         ADC CALL+1
2780         STA CALL+1
2790         LDA TEMP
2800         ADC CALL
2810         STA CALL
2820         ASL CALL+1
2830         ROL CALL
2840         PLA              get value this digit
2850         ADC CALL+1       and add it in
2860         STA CALL+1
2870         BCC .1
2880         INC CALL
2890         BCS .1        ...always
2900
2910  .2     LDA PNTR         back up PNTR
2920         BNE .3
2930         DEC PNTR+1
2940  .3     DEC PNTR
2950         RTS
2960  *--------------------------------
2970  NEXT.CHAR
2980         LDY #0
2990         LDA (PNTR),Y
3000         BEQ .1           EOL
3010         INC PNTR         bump pointer
3020         BNE .1
3030         INC PNTR+1
3040  .1     RTS
3050  *--------------------------------
3060  SEARCH.CALL.TABLE
3070         LDA CALL         hi-byte of called line
3080         AND #$FC         hi 6 bits
3090         LSR              make 0-126
3100         ADC #HSHTBL      carry is clear
```

```
        STA STPNTR
        LDA /HSHTBL
        ADC #0
        STA STPNTR+1
*--- fall into CHAIN.SEARCH routine
*--------------------------------
CHAIN.SEARCH
.1      LDY #0             point at pointer in entry
        LDA (STPNTR),Y
        STA TPTR
        INY
        LDA (STPNTR),Y
        BEQ .4             end of chain, not in table
        STA TPTR+1
        LDX #2             2 bytes in number
        LDY #2             point at line number in entry
.2      LDA (TPTR),Y       compare numbers
        CMP ENTRY,Y
        BCC .3             not this one, but keep looking
        BNE .4             not in this chain
        DEX
        BEQ .5             same number
        INY                next byte pair
        BNE .2          ...always

.3      JSR .5             update pointer, clear carry
        BCC .1          ...always

.4      SEC                did not find
        RTS

.5      LDA TPTR           point to matching entry
        STA STPNTR
        LDA TPTR+1
        STA STPNTR+1
        CLC
        RTS
*--------------------------------
SEARCH.LINE.CHAIN
        CLC                adjust pointer to start
        LDA STPNTR         of line # chain
        ADC #4
        STA ENTRY
        LDA STPNTR+1
        ADC #0
        STA ENTRY+1
        LDA #ENTRY
        STA STPNTR
        LDA /ENTRY
        STA STPNTR+1
        LDA LINNUM         put line number into symbol
        STA ENTRY+3
        LDA LINNUM+1
        STA ENTRY+2
```

```
3650        JMP CHAIN.SEARCH
3660 *--------------------------------
3670 ADD.NEW.ENTRY
3680        STA SIZE
3690        CLC                 see if room
3700        LDX #1
3710        LDY #0
3720        STY SIZE+1
3730 .1     LDA (STPNTR),Y    get current pointer
3740        STA ENTRY,Y       into new entry
3750        LDA EOT,Y         point old entry
3760        STA (STPNTR),Y    to this one
3770        STA TPTR,Y
3780        ADC SIZE,Y        and adjust end-of-table
3790        STA EOT,Y
3800        INY
3810        DEX
3820        BPL .1            now do low-bytes
3830 *--- see if there's going to be enough room
3840        LDA EOT
3850        CMP #LCR
3860        LDA EOT+1
3870        SBC /LCR
3880        BCS .3            MEM FULL error
3890 *--- move entry into call table
3900        LDY SIZE
3910        DEY
3920 .2     LDA ENTRY,Y
3930        STA (TPTR),Y
3940        DEY
3950        BPL .2
3960        LDA TPTR
3970        STA STPNTR
3980        LDA TPTR+1
3990        STA STPNTR+1
4000        RTS
4010
4020 .3     JMP AS.MEMFULL    abort with error message
4030 *--------------------------------
4040 PRINT.REPORT
4050        LDA PRGBOT
4060        STA PNTR          start defined line search
4070        LDA PRGBOT+1      at beginning of program
4080        STA PNTR+1
4090        LDA #0            start at chain 0
4100 .1     STA TEMP
4110        ASL
4120        TAY
4130        LDA HSHTBL+1,Y
4140        BEQ .2            no entries for this chain
4150        STA STPNTR+1
4160        LDA HSHTBL,Y
4170        STA STPNTR
4180        JSR PRINT.CHAIN
```

```
.2     INC TEMP
       LDA TEMP
       CMP #$40
       BCC .1             still more chains
       RTS                finished
*--------------------------------
PRINT.CHAIN
       JSR CHECK.FOR.PAUSE
       BEQ .1             <CR> abort
       LDY #2
       LDA (STPNTR),Y
       STA LINNUM+1
       INY
       LDA (STPNTR),Y
       STA LINNUM
       JSR CHECK.DEFINITION
       JSR PRINT.LINE.NUMBER
       LDA DEFFLAG        "*" or " "
       JSR MON.COUT
       CLC
       LDA STPNTR
       ADC #4             point at line # chain
       STA TPTR
       LDA STPNTR+1
       ADC #0
       STA TPTR+1
       JSR PRINT.LINNUM.CHAIN
       JSR MON.CROUT
       LDY #1
       LDA (STPNTR),Y     pointer to next call
       BEQ .2             no more
       PHA
       DEY
       LDA (STPNTR),Y
       STA STPNTR
       PLA
       STA STPNTR+1
       BNE PRINT.CHAIN   ...always

.1     PLA                return to top level
       PLA                if <CR> abort
.2     RTS
*--------------------------------
CHECK.DEFINITION
       LDY #3
       LDX #1
.1     LDA (PNTR),Y       look at next line in program
       CMP LINNUM,X
       BCC .4             < our number, get new line
       BNE .2             >  "    "   , not defined
       DEY                =  "    "   , go on
       DEX                now do low order bytes
       BPL .1
       LDA #" "           found it!
```

```
4730        BNE .3          ...always
4740
4750 .2     LDA #"*"           flag undefined line
4760 .3     STA DEFFLAG
4770        RTS
4780
4790 .4     LDY #1
4800        LDA (PNTR),Y       hi-byte of next line address
4805        BEQ .2
4810        PHA
4820        DEY
4830        LDA (PNTR),Y       and lo-byte
4840        STA PNTR
4850        PLA
4860        STA PNTR+1
4870        JMP CHECK.DEFINITION
4880 *---------------------------------
4890 PRINT.LINNUM.CHAIN
4900        LDA #0             reset counter to 0
4910        STA COUNTER        for each call
4920 .1     JSR TAB.NEXT.COLUMN
4930        LDY #2             point at line #
4940        LDA (TPTR),Y
4950        STA LINNUM+1
4960        INY
4970        LDA (TPTR),Y
4980        STA LINNUM
4990        JSR PRINT.LINE.NUMBER
5000        LDY #1             set up next pointer
5010        LDA (TPTR),Y
5020        BEQ .2             end of chain
5030        PHA
5040        DEY
5050        LDA (TPTR),Y
5060        STA TPTR
5070        PLA
5080        STA TPTR+1
5090        BNE .1          ...always
5100
5110 .2     RTS
5120 *---------------------------------
5130 TAB.NEW.LINE
5140        JSR MON.CROUT
5150
5160 TAB.NEXT.COLUMN
5170 .1     LDA #7             first tab stop
5180 .2     CMP MON.CH         cursor position
5190        BCS .3             perform tab
5200        ADC #6             next tab stop
5210        CMP #33            end of line?
5220        BCC .2
5230        INC COUNTER        count the screen line
5240        LDA COUNTER
5250        AND #1             look at odd-even bit
```

```
       BEQ TAB.NEW.LINE  both scrn and printer
       LDA #CR
       JSR MON.COUT1     <CR> to screen only
       JMP .1         ...always

.3     BEQ .4            already there
       SBC MON.CH        calculate # of blanks
       TAX
       JSR MON.PRBL2
.4     RTS
*--------------------------------
PRINT.LINE.NUMBER
       LDX #4            print 5 digits
       STX LZFLAG        turn on leading zero flag
.1     LDA #'0           digit=0
.2     PHA
       SEC
       LDA LINNUM
       SBC PLNTBL,X
       PHA
       LDA LINNUM+1
       SBC PLNTBH,X
       BCC .3            less than divisor
       STA LINNUM+1
       PLA
       STA LINNUM
       PLA
       ADC #0            increment digit
       BNE .2         ...always

.3     PLA
       PLA
       CMP #'0
       BEQ .5            zero, might be leading
       SEC               turn off LZFLAG
       ROR LZFLAG
.4     ORA #$80
       JSR MON.COUT
       DEX
       BPL .1
       RTS
.5     BIT LZFLAG        leading zero flag
       BMI .4            no
       CPX #0            if all zeroes, print one
       BEQ .4
       LDA #'            blank
       BNE .4         ...always

PLNTBL .DA #1
       .DA #10
       .DA #100
       .DA #1000
       .DA #10000
PLNTBH .DA /1
```

```
5800        .DA /10
5810        .DA /100
5820        .DA /1000
5830        .DA /10000
5840 *--------------------------------
5850 CHECK.FOR.PAUSE
5860        LDA KEYBOARD      keypress?
5870        BPL .2            no, go on
5880        STA STROBE
5890        CMP #CR           RETURN?
5900        BEQ .2            yes
5910 .1     LDA KEYBOARD      no, wait for
5920        BPL .1            another stroke
5930        STA STROBE
5940        CMP #CR           return .EQ. if RETURN
5950 .2     RTS
5960 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8409:Articles:Clear.Arrays.txt
==============================================================================
```

Faster Amper-routine to Zero Arrays.........Johan Zwiekhorst
Maasmechelen, Belgium

Although I have never subscribed to Apple Assembly Line, a friend of mine (who lives in nearby Heerlen, the Netherlands) does, and I always read his copies.

A few days ago I needed a routine to clear to zero all the elements in a number of Applesoft arrays, so I started looking in my friend's collection of AAL for such a program. I found the article entitled "Save Garbage by Emptying Arrays" in the December 1982 issue, pages 22-25.

That routine, however, only cleared string arrays. Bob designed it to set all strings in an array to null strings, so that garbage collection would be faster. But I needed a fast way to clear integer and real arrays as well. Bob's routine was also limited to clearing one array per call.

My routine clears any type of arrays, and can accept a list of array names separated by commas. It uses the ampersand hook, like this:

```
       & CLEAR array1,array2,array3,...
```

You can load the routine in any available memory, anywhere you have a spare 79 bytes. The listing shows it assembled into the ever-popular $300 space, but there are no internal addresses which require it to be there. Just be sure you hook the ampersand to the program, wherever you put it. If it is at $300, hook it like this:

```
       POKE 1013,76 : POKE 1014,0 : POKE 1015,3
```

The program is very similar to Bob's 1982 version: I eliminated the check he made for string arrays, added ampersand control, and checked for a comma to allow a list of array names rather than just one.

Lines 1250-1260 check that the byte following the ampersand is the CLEAR token. If not, a SYNTAX ERROR will result. If it is CLEAR, all is well.

Lines 1280-1290 check for a comma, and are not used until we have finished clearing an array. At the end, lines 1690-1710, you find my test after clearing an array. If the next byte of program is not a colon or end of line, it will branch back to the comma-test.

The code in between zeroes all the data bytes in an array. I could have done it the same way Bob did, but I did change a few things. Compare mine with his and you will learn two ways to control a clearing loop.

**How about a complete example of using &CLEAR? Lets make three arrays, with a mixture of types and dimensions. Of course, when the DIM statement works it initially zeroes the arrays, but I needed them cleared again later on.**

```
       100 DIM A(10,20), B%(200,4,4), C%(20)
       110 PRINT CHR$(4)"BLOAD B.CLEAR ARRAYS,A$300"
       120 POKE 1013,76:POKE1014,0:POKE1015,3
       ...
       500 &CLEAR A,B%,C$
```

==============================================================================
DOCUMENT :AAL-8409:Articles:Dan.Pote.Ad.txt
==============================================================================

---

Help Wanted

Electronic Engineer

Applied Engineering, manufacturer of Apple peripherals, needs a digital design engineer with Apple experience.

(214) 492-2027

---

```
==============================================================================
DOCUMENT :AAL-8409:Articles:DP18.Link.txt
==============================================================================
```

18-Digit Arithmetic, Part 5................Bob Sander-Cederlof

There is a lot of ground to cover in this installment, so I have been forced to use smaller type to squeeze it all in. I want to describe and list the code for the linkage to Applesoft, and for handling arithmetic expressions.

Loading and Linking to Applesoft

The ampersand (&) statement, according to the Applesoft Reference Manual (page 123, top of page) is

"intended for the computer's internal use only; it is not a proper Applesoft command. This symbol, when executed as an instruction, causes an unconditional jump to location $3F5. Use reset ctrl-C return to recover."

Not so! The &-statement is intended for adding extensions to the Applesoft language! It does cause a jump by the Applesoft interpreter to $3F5. If you have not set up any extensions you will get a syntax error when you use "&". But if you have extensions installed, you can work all manner of miracles. DP18 is one such miraculous extension. There are many more around, both in the public domain and in the form of commercial products.

This of course leads to a problem. What if you want to use two or more such extensions? I have written DP18 so that you can chain together one or more additional extension packages as you see fit.

It is very important to decide where the DP18 package will reside in memory. I spent weeks tossing around various options, back when I was designing the DPFP 21-digit package. Of course, at that time, Apples came equipped with anywhere from 16K to 64K RAM; now you can depend on almost all Apples having at least 48K RAM. I still favor the decision I made four years ago, to load the double precision code at $803, after shifting the Applesoft program far enough up in RAM to leave room.

I have a program I call ML LOADER, which is included on the S-C Macro Assembler disk as a sample program. It performs the function of moving an already- executing Applesoft program up higher in RAM. By including the following line at the beginning of my Applesoft program, I can load DP18 and link it to the & hook at $3F5:

```
10 IF PEEK(104)=8 THEN PRINT CHR$(4)"BLOADB.ML LOADER"
  :POKE 768,0 : POKE769,30 : CALL770
  :PRINT CHR$(4)"BLOAD DP18"
  :POKE 1014,PEEK(2051) : POKE 1015,PEEK(2052)
```

PEEK(104) looks at the high byte of the starting address of the Applesoft program. Normally Applesoft programs begin at $801, so PEEK(104)=8. If DP18 has not yet been loaded, then PEEK(104) will still be equal to 8. If it has already been loaded, then the rest of line 10 is skipped.

B.ML LOADER loads at $300. Its function is to shove the Applesoft program higher in RAM. You POKE the distance to shove into 768 (low byte) and 769 (high byte), than CALL 770. When you wake up an instant later, you have been relocated. The Applesoft program keeps on executing as though nothing happened. Only now there is a gaping hole between $800 and whatever.

DP18 loads at $803 and extends well into page $25. I grabbed 30 pages, moving the Applesoft program to $2601.It thus clobbers hires screen 1 memory. If you want to use hires screen 2 and the program is too large to fit under it, use POKE 769,88 instead of POKE 769,30 in line 10. This makes the program start at $6001, and leaves $2600-3FFF totally unused.

If you want to use other ampersand routines, POKE the link address at locations 2053 and 2054 ($805 and $806). If DP18 finds an ampersand command not starting with "DP", it jumps indirectly through this vector. The vector initially contains the address of Applesoft's SYNTAX ERROR routine, but it can be changed to allow using more than one set of &-routines.

Calling DP18

Whenever you want to execute a DP18 feature, you use the "&DP" statement. If DP18 has been properly connect to the & hook at $3F5, then the & will send the computer to DP18 (at line 2430 in the listing which follows). At this point DP18 begins to analyze and execute the characters that follow the ampersand.

If the first two characters after the ampersand are not "DP", the program will jump to a vector at $805 & $806. This normally points to Applesoft's SYNTAX ERROR routine. However, this location can easily be patched to point to your own ampersand routine.

If the first two characters are correct, DP18 will analyze succeeding statements separated by colons on the same line. There must be a colon immediately after the "&DP" statement. All of the rest of the statements on the line will be executed by DP18, rather than by the normal Applesoft interpreter. If you want to shut off DP18 before the end of the line, two colons in a row with nothing between will do so.

```
     150  & DP: INPUT X(0)
     160  & DP:Y(0) = X(0) * X(0) * PI: PRINT Y(0) :: GOTO 150
```

It is not necessary that the "&DP:" be the first statement in a line. For example, the following statement will take the square root of a number if the two strings are equal. It uses an Applesoft string comparison, and a double precision square root.

```
170  IF A$ = "SQR" THEN  & DP:Y(0) =  SQR (X(0))
```

You can also type double precision statements as direct commands in Applesoft once DP18 has been loaded.

```
]&DP:PRINT X(0): PRINT X(0) ^ 2
```

Four types of statements can be executed by the DP18 package: assignment, INPUT, PRINT, and IF statements. INPUT and PRINT statements will be covered in a later installment.

The DP18 IF statement evaluates a logical expression in 18-digit precision, and then reverts to normal Applesoft processing:

```
180 &DP : IF A(0) < 1.52345678976543 THEN X = 3
```

The DP18 assignment statement takes two forms: real assignment, and string assignment. String assignment is used to convert DP18 values to strings, so that they can be used by normal Applesoft:

```
190 &DP : A$ = STR$ (X(0))
```

Real assignments are the normal computational statements, like:

```
200 &DP : A(0) = (4*PI*R(0)^3)/3
```

DP18 Variables

All variables referenced by DP18 must consist of two adjacent array elements. The array must be a REAL array, that is, it must not be INTEGER or STRING.

Remember that Applesoft array subscripts begin with 0 and go up to the limit defined in the DIM statement. An array dimensioned "3,11,11" has three dimensions. The first runs from 0 to 3; the second from 0 to 11; and the third also from 0 to 11. It could contain 4*12*12=576 real elements, or 2*12*12=288 double precision elements.

Applesoft arrays are stored in memory with the leftmost subscript varying the fastest. For example, in the array XY(3,10,10), element XY(0,j,k) comes immediately before element XY(1,j,k). Therefore you may, in effect, create an array of double precision values by merely prefixing an extra dimension to the dimension list.

If you wish to set up separate variables, you may do so by dimensioning them to have two real elements each. For example, the statement

```
10  DIM A(1),B(1),C(1),X(1)
```

will set up four separate variables for use with DP18. You reference the variables within double precision statements with the subscript 0. For example:

```
20  & DP:X(0) = (A(0) + B(0)) * C(0)
```

Note that you don't have to dimension these variables, since Applesoft will default to a dimension of 10. However, it is a good idea to dimension all double precision variables because it saves memory (only 2 real elements are allocated instead of 11) and it makes it easier for someone else to follow your program.

If you wish to create an array of double precision values, you do so by dimensioning the array with one extra dimension. The extra dimension comes first and should be "1"; this dimension generates two real items, or one double precision item. For example,

```
10  DIM A(1,12),B(1,5,6)
```

creates two arrays that can be used for double precision values. The array A can be thought of as an array of 13 double precision values from A(0,0) to A(0,12). The array B could store 42 double precision values from B(0,0,0) to B(0,5,6). If you always remember to use one extra dimension, to put that extra dimension first, to set that dimension to "1", and to refer to items with the first subscript = 0, then you will succeed in using DP18.

DP18 Constants

Double precision constants are entered in the same way as single precision constants. The differences between standard Applesoft and the DP18 constants are that DP18 converts and stores 18 significant digits rather than 9, and that exponents may be in the range of +/- 63 rather than +/- 38.

Conversion of constants is very fast in DP18. DP18 will convert constants over 4 times faster than normal Applesoft, even using more digits! It is quicker to convert a constant than it is to find and use a DP18 variable, especially multi-dimensioned variables. This is completely opposite from normal Applesoft, where variables are quicker than constants.

Conversion Between Single and Double Precision

You will often need to convert a single precision value into a double precision one for purposes of computation. This is easily done by first converting it to a string and then using DP18's VAL function as shown here.

```
100  REM CONVERT X TO DOUBLE PRECISION VALUE
110  DIM DP(1)
120  INPUT "VALUE TO BE CONVERTED? ";X
130  &DP:DP(0) =  VAL ( STR$ (X))
140  &DP: PRINT DP(0)
150  GOTO 120
```

You will also want to convert from double precision back to single precision. This also involves converting to a string, but takes more than one statement.

```
     100  REM CONVERT DP(0) TO SINGLE PRECISION VALUE
     110  DIM DP(1)
     120  &DP:INPUT "VALUE TO BE CONVERTED? ";DP(0)
     130  &DP:A$ =  STR$ (DP(0))
     140  X =  VAL (A$) : PRINT X
     150  GOTO 120
```

Note that lines 130 and 140 could be combined onto one line if there were two colons separating the statements. See the section on functions for more information about the STR$ and VAL functions.

DP18 Arithmetic Expressions

Expressions in DP18 are very much like expressions in Applesoft. Except for AND and OR, they are evaluated using the standard rules of precedence as found on page 36 of the Applesoft manual. AND and OR have the same precedence in DP18 and are executed left to right. The order of precedence is listed below. Operations on a higher line are executed before operations on a lower line. Operators on the same line are executed left to right.

```
( ) function calls
+ - NOT        unary operators
^
* /
+ -
<  >  =  <=  >=  =>  =<  <>  ><
AND OR
```

These all work the same as they do in Applesoft, except that they operate on double precision numbers.

DP18 supports many of the numerical functions that Applesoft does: SIN, COS, TAN, LOG, EXP, SGN, ABS, INT, SQR, ATN, VAL, and the string function STR$. There is also a special function, PI, which has no arguments. You don't even write parentheses after it. You just use it like it was a constant. Wherever you use it, you get the value pi accurate to 20 digits.

Explanation of the Code

As in previous installments of this series on DP18, I cannot show everything at once. A whole series of subroutines which have either already been printed or will be printed in future installments are represented in this listing by ".EQ $FFFF" in lines 1330-1550. All the data areas actually used in the code listed this month are included, so that you can see what the code is working with and on.

As mentioned above, the "&" statement sends Applesoft to line 2430. Lines 2430-2500 check for "DP" following the ampersand. If not "DP", then lines 2370-2390 branch to the next ampersand interpreter in your chain. If you have not set up another &-interpreter, then the SYNTAX ERROR message will pop out.

DP.NEXT.CMD (lines 2520-2800) begins by looking for a colon or end-of-line. End of line means you are through with DP18, so an RTS carries you back to the Applesoft interpreter. A colon means you are ready with a DP18 statement. If the next character is also a colon, however, you are sent back to Applesoft (lines 2570-2580). Next I check for the three legal tokens (IF, INPUT, and PRINT) and branch accordingly.

Since IF is simple and IF is included in this listing, let's look at IF now. Lines 3130-3280 handle the IF statement. First I evaluate the expression, which is considered to be a logical expression with a true-or-false value. Zero means false, non-zero means true. Following the expression I must find either a THEN or GOTO token. The truth value is found in DAC.EXPONENT, because a $00 exponent means a zero value. AS.IF.JUMP in the Applesoft ROMs can handle the rest, because the THEN or GOTO pops us out of DP18 back to normal Applesoft. Neat!

Meanwhile, back in DP.NEXT.CMD, if the statement is not IF-INPUT-PRINT it must be an assignment statement. If I am successful at getting a variable name next, it may be either a DP18 variable or a string variable. If AS.VALTYP is negative, it is a string variable and DP.STR takes over. If not, CHECK.DP.VAR will verify that it is a real array variable. The address is saved at RESULT, the DP18 expression evaluated, and then the answer saved at RESULT. And back to the top of DP.NEXT.CMD.

DP.STR handles statements like A$=STR$(xxx) where xxx is a DP18 expression. You can probably follow the comments in this section.

GET.A.VAR checks to see that the current character from your program is a letter, because all variables must start with a letter. If so, AS.PTRGET will search the variable tables and return with an address in the Y- and A-registers. CHECK.DP.VAR compares this address with the beginning of the array variable table. If it is inside the array table, and if the variable is real (not string or integer), it is a valid DP18 variable.

DP.EVALUATE cracks and calculates a DP18 expression. A special stack is used for temporary values, and it is deep enough to hold 10 of them. If your expression is so complicated that more than 10 temporary values need to be stacked (very unlikely), then the FORMULA TOO COMPLEX message will scream. Applesoft uses the hardware stack in page 1 for the same purpose, but it only has to stack 5-byte values; DP18 stacks 12 bytes for each value. EVALUATE starts by emptying the stack, zeroing a parenthesis level count, and clearing the accumulator (DAC). After DP.EXP finishes all the dirty work, The stack must be empty and the parenthesis level zero or there was a SYNTAX ERROR.

Actually parsing and computing an expression can be done in many ways. I chose a recursive approach that breaks the job up into little independent pieces small enough to understand. First, let's allow all expressions to be a series of relational expressions connected with ANDs and ORs. The simplest case of this is merely a relational expression alone. And the simplest relational expression is an expression all by itself with no relations. If the expression does have relational operators or ANDs or ORs, the result will be a true or false value. If not, it will have a numerical value.

Comment blocks atop DP.EXP, DP.RELAT, DP.SUM, etc. show the continued breakdown of parts of an expression. DP.RELAT connects one or more sums with relational operators. DP.SUM connects one or more terms with "+" and "-" operators. DP.TERM connects one or more factors with "*" and "/" operators. DP.FACTOR connects one or more elements with the exponentiation operator (^). DP.ELEMENT cracks a constant, searches for a variable's value, calls a function, or calls on DP.EXP recursively to handle an expression in parentheses. DP.ELEMENT also handles the unary operators "+", "-", and "NOT".

If DP.ELEMENT determines that the element is a function call, there are several types. The VAL function is supervised by lines 5800-5830. Since the argument of the VAL function is a string expression, it is significantly different from the other functions. The ATN function is also given special treatment, because DP18 allows the ATN function to be called with one or two arguments. All the rest of the functions have one DP18 expression for an argument, so they are handled as a group. A table of addresses at lines 2160-2310 directs us to the appropriate processor. The code for all these functions will be revealed in future installments.

DP.VARNUM is called upon to handle variables and numbers. First lines 6130- 6280 check for and handle the special DP18 constant "PI". Lines 6300-6350 handle DP18 variables, and lines 6370-6470 handle numbers.

PUSH.DAC.STACK pushes the 12-byte value in DAC on the special expression stack, unless there is not enough room. POP.STACK.ARG pulls a 12-byte value off the stack and plops it into ARG.

And Next Month...

There are three major areas left for future installments: INPUT, PRINT, and the math functions. Some of you have been diligently studying and entering each installment as we go, and are gradually obtaining a powerful package. Others are waiting for the Quarterly Disks, to conserve their fingertips. Remember, all the source code each three months is available on disk for only $15.

==============================================================================
DOCUMENT :AAL-8409:Articles:Fast.Scrn.Msgs.txt
==============================================================================

Put Your Messages on the Screen..............William M. Reed

COUT is slow.  COUT with DOS looking on is even slower.  And I suppose with ProDOS, more so.  If you want to get a short message on the screen in a hurry, you can bypass COUT and put it there directly.

In all of the following examples I am going to assume that the message is stored in RAM exactly as it should be on screen, and that after the last character is a byte with $00 in it.  I also assume that you are only writing one line, so that the message will not spill over to another line.

Here is a loop that writes a message on the bottom line of the screen:

```
MESSAGE
       LDY #0
.1     LDA MESSAGE,Y
       BEQ .2          ...END OF MESSAGE
       STA $7D0,Y
       INY
       BNE .1          ...ALWAYS
.2     RTS
```

If you want to write on the current line, whose base address is kept by the monitor in BASL and BASH ($28 and $29), just change the STA $7D0,Y line to STA (BASL),Y.

All well and good for 40 columns, but what about the 80-column //e and //c screens?  Well, you can still do it, like this:

```
MESSAGE.80
       LDX #0       MESSAGE INDEX
.1     TXA
       LSR          COLUMN/2, ODD/EVEN TO CARRY
       TAY          INDEX INTO SCREEN MEMORY
       LDA MESSAGE,X
       BEQ .3       ...END OF MESSAGE
       STA PAGE1
       BCS .2       ...ODD, PAGE 1
       STA PAGE2    ...EVEN, PAGE 2
.2     STA (BASL),Y
       INX
       BNE .1       ...ALWAYS
.3     RTS
PAGE1  .EQ $C054
PAGE2  .EQ $C055
```

Of course, these routines put the messages on the screen only.  But that may be just what you want, to put messages on the screen without affecting the report going out to file or printer.  Also, these routines do not handle CR, end of line, scroll, etc; but they sure get to the screen in a hurry!

==============================================================================
DOCUMENT :AAL-8409:Articles:Front.Page.txt
==============================================================================

$1.80

Volume 4 -- Issue 12                September, 1984

In This Issue...



News about Micromation.

Jack Lewis's company, which among other things makes a line of Apple-related products to support the Heathkit Hero robot, has changed its name to Arctec Systems, Inc.

Jack also has a stand alone voice recognition system with an RS-232C interface which may be of interest to some of you.  It contains a 65C02 processor, 4K ROM, and 16K of battery-backed-up RAM.  Speaker-dependent recognition of up to 256 words or short phrases is possible, with 95-98% accuracy claimed.  Arctec's number is (301) 730-1237, in Columbia, Maryland.

And Some Bad Tidings

The saddest news I have heard lately is of the demise (bankruptcy) of Softalk Publishing.  Softalk has been my favorite of all the magazines devoted to the Apple.  At this point I do not know how to obtain copies of any of their back issues, or of the books they have published.  I assume, and hope, they will be available again soon. With the passing of so many companies, via Chapter 11, many magazines are having great difficulty this year.  Unpaid advertising bills then cause a domino effect....

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8409:Articles:Graph.Biblio.txt
==============================================================================

Bibliography on Apple Hi-Res Graphics......Bob Sander-Cederlof

There has been a lot of material published in the last seven years about Apple's hi-res graphics. The problem is finding it! Most of the neat programs and explanations have not yet made it from the pages of various magazines into full size books. I recently decided to make a list, so that I don't have to keep leafing through mile-high stacks of magazines. Since I have never been a devotee of Pascal, I purposely omitted most articles relating to graphics in that language. I also omitted reviews and announcements of commercial hi-res products.

I looked through my book shelves and noted all books I could find there. I also went through all my back issues of Byte, Micro, Call APPLE, and Apple Orchard. Still to go are Nibble, Kilobaud, Softalk, and Creative Computing.

Books
Apple Graphics & Arcade Game Design, Jeffrey Stanton. The Book Co., 1982, 288 pages, $19.95. By the time you work through this one, you have a functioning hi-res arcade game!

Apple II Graphics, Ken Williams. Prentice Hall, 1983, 150 pages, $19.95. (Originally a series of articles in Softline Magazine, Sep 81 through Jan 83.)

Applied Apple Graphics, Pip Forer. Prentice-Hall, 1984, about 400 pages plus diskette, price unknown. Lo-res, Hi-res, 3-D, etc., with over 50 program in BASIC on disk.

Graphically Speaking, Mark Pelczarski. Softalk Books, 1984, 170 pages, $19.95. Originally a series of articles which ran from May 1982 through September 1983 in Softalk Magazine. Includes many programs in Applesoft and assembly language. Covers drawing, animation, filling, packing/unpacking, and 3-D. Disk available.

Microcomputer Graphics, Roy E. Myers. Addison-Wesley, 1982, 282 pages, $12.95. More than 80 Applesoft programs. 2-D and 3-D graphics, windowing, transformations, hidden lines, and much more. Disk available.

Books with some material on Apple Graphics
Animation, Games, and Sound for the Apple II/IIe, Tony Fabbri. Prentice-Hall, 144 pages.

Enhancing Your Apple II, Volume I, Don Lancaster. Howard Sams & Co., 1984, 268 pages, $15.95. Hardware and software tricks for switching between modes and screens dynamically, programs for hundreds of hi-res

colors and patterns, fast screen fill. Good explanations of the way things work.

What's Where in the Apple, William F. Luebbert. Micro Ink, 1982, about 300 pages. First half of book is text describing Apple; chapter 14 covers lo-res graphics, and chapter 16 covers hi-res graphics. Includes details about hardware switches, memory mapping, and firmware.

Magazine columns
Assembly Lines, Roger Wagner, Softalk Magazine. From March 82 to June 83 this column covered various topics in Apple hi-res graphics. It should be made into a book, but has not yet been.

The Graphics Page, Bill Budge, Softalk Magazine. Oct 83 through Jun 84. Deep material, by the author of Pinball Construction Set. Further installments were promised, but not yet seen.

Apple II Graphics, Ken Williams, Softline Magazine. Sep 81 through Jan 83. Now available in book form (see above).

Graphically Speaking, Mark Pelczarski, Softalk Magazine. May 82 through Sep 83. Now available in book form.

Magazine Articles
Byte

Apple FAX: Weather Maps on a Video Screen, Keith H. Sueker. Jun 84, 146-151.

CHEDIT: a Graphics-Character Editor, Jerry Sweet. May 82, p426-444.

Double the Apple II's Color Choices, Robert H. Sturges. Nov 83, p449-463.

Double-Width Silentype Graphics for Apple, Charles Putney. Feb 82, p413-423.

GRPRINT: an Apple Utility Program, Douglas Arnott. Dec 82, 398-403.

Interactive 3-D Graphics for Apple II, Andrew Pickholtz. Nov 82, 474-505.

Kinetic String Art for the Apple, Louis Cesa. Nov 80, p62-63.

More Colors for your Apple, Allen Watson, Steve Wozniak. Jun 79, p60-68.

New Shape Subroutine for the Apple, Richard T. Simoni. Aug 83, p292-309.

Picture Perfect Apple, Phil Roybal. Jan 81, p226-235.

Shape Table Conversion for the Apple II, Dave Partyka. Nov 79, p63.

Simplified Theory of Video Graphics, Allen Watson. Part 1, Nov 80, p180-189. Part 2, Dec 80, p142-156.

Three-dimensional Graphics for the Apple II, Dan Sokol, Nov 80, p148-154.

Micro

Apple Bits, Richard C. Vile Jr. Part I, Sep 81, p75-77. Part 2, Oct 81, p94-96. Part 3, Nov 81, p105-108. (Lo-Res)

Apple Color Filter, Stephen R. Berggren. Jun 81, p53-54.

A Hi-Res Graph Plotting Subroutine in Integer BASIC for the Apple II, Richard Fam. Feb 80, p9-10.

Apple Graphics, staff. Sep 81, p49. Intro to several other articles.

Apple Graphics for Okidata Microline 80, Gary Little. May 83, p80-86.

Apple Hi-Res Graphics and Memory Use, Dan Weston. Nov 82, p79-81.

Apple II High Resolution Graphics Memory Organization, Andrew H. Eliason. Oct-Nov 1978, p43-44.

Apple II Hi-Res Picture Compression, Bob Bishop. Nov 79, p17-24.

Apple Pascal Hi-Res Screen Dump, Robert D. Walker. Feb 83, p54-55.

Apple Shootdown, a lo-res graphics game, Eric Grammer. Nov 82, p72-73.

A Versatile Hi-Res Graphics Routine for the Apple, Adam P. King. Mar 83, p77-81.

Constructing Truly 3-D Mazes, Dr. Alan Stankiewicz. Aug 84, p19-21.

Creating Shape Tables, Improved!, Peter A. Cook. Sep 80, p7-12.

Define Hi-Res Characters for the Apple II, Robert F. Zant. Aug 79, p44-45.

Getting Around the Apple Hi-Res Graphics Page, Eagle Berns. Nov 82, p93-95.

Graphing Rational Functions, Ron Carlson. Dec 80, 7-9.

Hi-Res Characters for Logo, Dan Weston. Sep 83, p50-53.

Hi-Res Screen Dump for Epson MX-80, Robert D. Walker. Apr/May 84, p55-61.

How to Do a Shape Table Easily and Correctly, John Figueras. Dec 79, p11-22.

Introduction to 3-D Rotation on the Apple, Chris Williams. Nov 82, p99-101.

Paddle Hi-Res Graphics, Kim G. Woodward. Sep 81, p68-69.

Random Number Generator in Machine Language for the Apple, Arthur Matheny. Includes a graphics simulation of a globular cluster. Aug 82, p57-60.

SHAPER: A Utility Program for Managing Shape Tables, Clement D. Osborne. Sep 81, p50-56.

Sun and Moon on the Apple, Svend Ostrup. Jan 83, p35-37. Hi-res simulation of orbits and phases.

True 3-D Images on Apple II, Art Radcliffe. Sep 81, p71-73.

Call-A.P.P.L.E.

80-column //e Lo-Res Graphics, Rob Moore. Jul 83, p9-13.

Adding XPLOT to Applesoft, Mark Harris. Apr 84, p17-18,24.

A Higher Text Apple-cation, Donal Buchanan. Nov 82, p47-50. Using Higher Text for ancient alphabets.

Animation with Data Arrays, Pat Connelly. Nov-Dec 80, p11-17.

Apple Gaming: Playing Card Generation, Jim Hilger. Nov-Dec 79, p39-45; Jan 80, p39. Hi-res playing card pictures from Integer BASIC.

Applesoft Firmware Card Hi-Res Routines, Steve Alex. Oct 79, p33.

Applesoft Graphics Mover, Homer O. Porter. Sep 83, p29-31.

Arcade Graphics Techniques, Chris Jochumson. Apr 83, p9-14.

Character Generator ROM, Ian M. Jackson. Nov 82, p21-29. Programs for moving a Higher Text font into ROM.

Color 21, Darrell Aldrich. Jul-Aug 79, p21.

Color Me Apple, M. A. Iannce. Nov 82, p9-18. In-depth explanation of hi-res color with demo program.

Doing the Splits, Roy Myers. Aug 82, p61-65. Making room for hi-res pictures by moving your program.

Graphic Garbage Collection, Richard Cornelius & Melvin Zandler. Nov 82, p53-55. Lets you watch garbage collection activity on the hi-res screen.

Higher Text in Action, Steve Brugger. Jan 84, p30-31.

Higher Text on the Loose, Val J. Golding. Jun 81, p47-49. Explanation of the background functions of Higher Text.

Hi-Res Dump program modification, Tom Lewellen. Jul-Aug 79, p36.

Hi-Res Full Scroll, Edward C. So. Feb 82, p23-34. Scroll up, down, left, or right by one pixel position at a time.

Hi-Res Hi-Jinx, Edward C. So. Apr 82, p59. POKEing and PEEKing dots.

Hi-Res Screen Switch (program), Wes Huntress. Jul-Aug 80, p48-49.

Hi-Res Slide Show, Stowe Keller. Dec 83, p49-54.

Magic Square Dance, J. Taylor. Sep 83, p51-52.

MX-100 Hi-Res Dump, Bruce C. Detterich. Mar 82, p41-49.

Painting the RAMcard, Donald W. Miller Jr. Apr 83, p51-54.

Picture Compression, Edward C. So. May 82, p21-35. Very complete, builds on Bob Bishop's attempts.

Playing Card Generator, Vincent Aderente. Nov 82, p31-35. Applesoft versions of Jim Hilger's stuff.

Scrunch, Darrell Aldrich. Jun 79, p21-23. Squeeze four pictures into one screen.

Shape Display Utility, Major Peter M. Beck. Mar-Apr 80, p39.

Shape Table Splicer, Cyrus W. Roton. Sep 83, p33-35.

Slow Plot, Jim Morriset. Nov 82, p63-64. Speed control for hi-res drawing.

Smooth Animation, Jonathan Kandell. Feb 83, p61-62.

The Graphics Toolkit, Randi J. Rost. Part 1: Apr 84, p10-15 (screen mapping). Part 2: May 84, p23-26. Part 3: Aug 84, p43-48. Line drawing algorithms, disassembly of Applesoft HPLOT.

Three Dee Demos, David Sun. Jan 83, p49-51.

Understanding Hi-Res Graphics, Loy Spurlock. Jan 80, p6-15.

Using the Splitter, Norman L. Kushnick. Jan 83, p53-55. More help in making room for pictures.

Why Don't You Watch Where You're Going?, Kenneth Manly. Oct 80, p25-28. A hi-res SCRN function.

**Zoom, Neil Konzen. Jan 80, p28-32. Expand a 1/9th screen to full size.**

**Apple Orchard**

**Double-Size Graphics for the Silentype, Bruce F. Field. Spring 81, p30-34.**

**Hi-Res Dump Routine for Integral Data Printer (IDS-225), Darrell & Ron Aldrich. Mar-Apr 80, p54-55. (Originally published in Call APPLE).**

**Hi-Res Graphics: Resolving the Resolution Myth, Bob Bishop. Fall 80, p7-10.**

**How the Dot Patterns Produce Colors, Allen Watson III. Jan 84, p44-46.**

**How the Double Hi-Res Hardware Came to Be, Allen Watson III. Jan 84, p42,43.**

**Notes on Hi-Res Graphics Routines in Applesoft, C. K. Mesztenyi. Spring 81, p17-19.**

**Practical Super Hi-Res Graphics, R. H. Good. Spring 81, p20.**

**Secrets of Professional Graphics, William Harvey. Part I, Behind the Scenes, Sep-Oct 82, p64-72. Part II, The Real Challenge: Putting It All Together, Nov-Dec 82, p36-53. Part III, Techniques of Animation, Mar 83, p62-69.**

**Shape Definition Conversion Table, David G. Huffman. Fall 81, p78-79.**

**Shaping Up with the Apple II, Mark L. Crosby. Mar-Apr 80, p37-45.**

**The Mysterious Orange Vertical Line, Pete Rowe. Fall 80, p11.**

**True 16-color Hi-Res, Allen Watson III. Jan 84, p26-41.**

**Understanding Hi-Res Graphics, and how to include text in your Hi-Res Graphics Programs, Loy Spurlock. Fall 80, p12-21.**

```
==============================================================================
DOCUMENT :AAL-8409:Articles:Index.2.Mask.txt
==============================================================================
```

Turn an Index into a Mask..................Bob Sander-Cederlof

How do you write a program that will turn a number from 0 to 7 into a bit mask $01, $02, ...$40, $80? I want an index of 0 to return $01, 1 to return $02, 2 to return $04, and so on up to 7 returning $80.

The simplest, shortest, and speediest is to use a direct table look-up. Assuming the byte with the index value is in the A-register, the code would look like this:

```
       AND #7          isolate index bits
       TAX             index to X-register
       LDA TABLE,X     get mask from table
```

and the table would look like this:

```
       TABLE .HS 01020408
             .HS 10204080
```

This technique has the wonderful advantage that if you need a different translation, you can simply use a different table. For example, if you want the reverse pattern, with 0 returning $80 and 7 returning $01, simply change the table to:

```
       TABLE .HS 80402010
             .HS 08040201
```

The table lookup method has the shortest code, but counting the table does take 14 bytes. If you don't worry so much about speed and flexibility, you can write a little loop that will create the mask value like this:

```
       MAKE.MASK.2
               AND #7     isolate index bits
               TAX        index into X-register
               LDA #$01   initial mask value
       .1      ASL        shift loop to position
               DEX        to Xth bit
               BPL .1     shifts once to many
               ROR        restore after extra shift
               RTS
```

I put an RTS at the end because this piece of code makes a nice size subroutine. Nevertheless, for comparison to the table lookup code above, let's count neither the JSR to call it nor the RTS at the end. The shift-loop method takes only 10 bytes, four less than the table lookup. But it is slower, taking 14 cycles if the index is 0, 21 if 1, up to 63 for an index of 7. Sometimes saving four bytes is more important that speed, and sometimes speed is more important.

To generate the reverse sequence with the shift loop method, make three simple changes to MAKE.MASK.2: the initial mask value from $01 to $80; the ASL to LSR; and the ROR to ROL.

Note that both techniques shown above use the X-register. If the X-register is busy, you could use the Y-register instead. Just for the challenge, I wanted to see if I could write a reasonably efficient index-to-mask routine that did not use the X- or Y- registers at all.

The first method that came to mind was fast enough, but took too much space and did not seem creative. It involved a series of CMP and BEQ instructions to branch to 8 different LDA's:

```
       SILLY.WAY
               AND #7  isolate index
               BEQ .0  index=0
               CMP #1
               BEQ .1  index=1
               ...
               CMP #6
               BEQ .6    index=6
               LDA #$80  index=7
               RTS
       .0      LDA #$01
               RTS
       .1      LDA #$02
               RTS
               ...
       .6      LDA #$40  index=6
               RTS
```

If I had written every line above, you would see that it takes 52 bytes.

Next I though of a more efficient way to do the CMP's so that not so many were needed.

```
       NOT.SO.SILLY.WAY
               AND #7  isolate mask
               BEQ .0  index=0
               CMP #4
               BEQ .4  index=4
               BCS .60 index=5, 6, or 7
               CMP #2  index=1, 2, or 3
               BEQ .2  index=2
               BCS .3  index=3
               LDA #$02  index=1
               RTS
       .60     CMP #6  index=5, 6, 0r 7
               BEQ .6  index=6
               BCS .7  index=7
               LDA #$20  index=5
               RTS
```

```
        .0      LDA #$01  index=0
                RTS
        .2      LDA #$04  index=2
                RTS
            and so on.
```

This method takes a total of 46 bytes.

Here is one which is even shorter, which uses "tricky" arithmetic.

```
        TRICKY.WAY
                AND #7
                CMP #2
                BCC .5  (0 or 1) plus 1
                BEQ .5  (2) plus CARRY plus 1 --> 4
                CMP #4
                BCC .4  (3) plus 4+1 --> 8
                BEQ .3  (4) plus 6+4+1+C --> $10
                CMP #6
                BCC .2  (5) plus $10+6+4+1
                BEQ .1  (6) plus $1E+$10+6+4+1+C
                ADC #$3F  (7) plus $3F+$1E+$10+6+4+1+C
        .1      ADC #$1E
        .2      ADC #$10
        .3      ADC #6
        .4      ADC #4
        .5      ADC #1
                RTS
```

Not counting the RTS, that is 31 bytes. Cases 0 and 1 take only 9 cycles. The longest one, when the index is 7, takes 32 cycles.

All of these longer methods can be made to generate the reverse sequence by simply inverting the index before beginning the tests. Use "EOR #7" before the "AND #7".

I came up with an even trickier version, which shaved another byte or two off TRICKY.WAY. Believe it or not, it really works:

```
        TRICKIER.WAY.REVERSE
                EOR #7
        TRICKIER.WAY
                AND #7     isolate index
                SEC        00-01-02-03-04-05-06-07
                ROL        01-03-05-07-09-0B-0D-0F
                CMP #3
                BCC .0     turn 0 into $01
                CMP #7
                BCC .12    03-->02, 05-->04
                ADC #6     ..-..-..-0E-10-12-14-16
                CMP #$12
                BCC .34    0E-->08, 10-->10
                ADC #$2B   ..-..-..-..-..-3E-40-42
                CMP #$42
```

```
             BCC .56    3E-->20, 40-->40
             ASL        42-->84-->80
       .56   AND #$E0
       .34   AND #$F8
       .12   AND #$FE
       .0    RTS
```

If the index is 0, this one takes 11 cycles.  Worst case is for index 7, at 34 cycles.

A source file on the quarterly disk will include all of the above examples, plus a driving program that runs through all 8 cases and displays the results for each and every method.

In real life, I would probably use the shift-loop or the table look up.  Most likely the table lookup, because it is the easiest to understand and modify, and by far the shortest in time.  Nevertheless, it is very useful to experiment with other techniques.  You learn a lot from the experience, and it is fun!

```
==================================================================
DOCUMENT :AAL-8409:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1..................................$92.50
Version 1.1 Update...............................................$12.50
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM...................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $50)  $45
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $30)  $25
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36
Aztec C Compiler System (Manx Software)..................(reg. $199) $180

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                         or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
QuikLoader EPROM System (SCRG)..............................($179) $170

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner...................($19.95)  $18
```

```
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
    "Microcomputer Design & Troubleshooting", Zumchak.......($17.95)  $17
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

==============================================================================
DOCUMENT :AAL-8409:Articles:Reviews..txt
==============================================================================

Some Great New Books......................Bob Sander-Cederlof

"Beneath Apple ProDOS", by Don Worth and Pieter Lechner. Quality Software, 1984, 276 pages plus 10 page reference card, $19.95. (By it from us for $18 plus shipping.)

We have been waiting a long time for this one, by the authors of "Beneath Apple DOS". If you read that one, you'll want this one too. And if you use or plan to use ProDOS, you will almost REQUIRE this new book.

Apple has documented ProDOS pretty thoroughly, but just TRY to get a copy of their books. Hardly any Apple dealers stock the reference manuals now. Apple requires a minimum order to buy the manuals, and they are a relatively slow moving item. Hence, dealers don't order them. Some we have talked with lately refused to admit they knew of the existence of even the Apple //e Reference Manual (over 18 months old now)! And Apple so far will not sell the books to anyone who is not an authorized Apple dealer. Catch-22, right?

But even if you have Apple's ProDOS reference manuals, as I do, you still need "Beneath Apple ProDOS". Look at the table of contents, and see if you can resist.

The most heavily thumbed pages in my copy of "Beneath Apple DOS" are the ones which give detailed comments on the entire DOS assembly language image. Unfortunately, the equivalent section does not come bound in to "Beneath Apple ProDOS". Since Apple has decided to freeze DOS, a published commentary is possible. But ProDOS is deliberately kept warm and fluid. So far there are at least four versions around; all have the same characteristics and machine language interface, but subroutines have been shuffled and rewritten. A line-by-line commentary could become obsolete every six months.

A special coupon is bound into the book at the place where you would expect the commentary. If you want the commentary, you remove the coupon page, fill in your name, address, and ProDOS version number, and send it with $12.50 to Quality Software. With the commentary you will receive a new coupon so your can order a subsequent supplement when ProDOS changes versions.

"Assembly Cookbook for the Apple II/IIe", Don Lancaster. Howard Sams & Co., 1984, 408 pages, $21.95. (Buy it from us for $20 plus shipping.)

Don is sold on the synergistic combination of a full-screen 80-column word processor for handling source code with an assembler. His favorite pairing is Applewriter //e with EDASM (from DOS ToolKit). Consequently a large section of the book is devoted to how the marriage is performed, what the advan- tages are, and how to work around or ignore the disadvantages. Don knows Applewriter inside out, and uses it for all his word processing as well as for programming. There are some distinct advantages to using the same editor for both: writing books about assembly language programming is easier; only one set of commands, tricks, and quirks need be learned. Applewriter //e's WPL language helps overcome the disadvantages of using a screen-oriented processor on line-oriented information.

The second half of the book contains sample assembly language programs, explained in detail. These are not your run-of-the- mill examples, but great subroutines and programs you can actually use, as well as learn from.

"Microcomputer Design and Troubleshooting", by Eugene M. Zumchak. Howard Sams & Co., 1982, 350 pages, $17.95. (Buy it from us for $17 plus shipping.)

From time to time I am called upon to understand and work with electronics. My degree is in Electronic Engineering, but I got it in the vacuum tube era (over 20 years ago). What now fits on one chip used to fill a whole ship.... Anyway, I struggle through. But I have found a book recently that has really helped: it is not really a new book, but is new to me.

Gene Zumchak has a unique approach, which is PRACTICAL. He believes in designs which are easy to troubleshoot. He tells how adding a few low cost components here and there will avoid the expense of a logic analyzer and three weeks of debugging time. For example, using an EPROM emulator and a few LED's in critical places in a microprocessor design could save endless hours of burning and erasing EPROMs, attaching logic analyzer leads and watching oscilloscope traces, and pulling all your hair out. Although every chapter has helpful ideas in the areas of trouble prevention and diagnosis, chapter 6 is devoted entirely to the subject. Another feature Gene promotes is low power consumption.

Jack Lewis is president of Micromation, a company which makes hardware for use with the Hero-1 Robot. They have designed interfaces between Apple and Hero, speech input processors, and much more. When Jack began, he contracted with Gene Zumchak to teach his people the techniques which are now in this book. Jack is the one who recommended the book to me.

And now I recommend the book to you, if you like to dabble in hardware design. Even practicing designers will find the ideas well worth the price of reading the book.

I also recommend "The Computer Journal", a monthly newsletter/ magazine published by Art Carlson. $24/year (U.S.) gets you regular articles such as "Build a 68008 CPU Board for the S-100 Bus", "Electronic Dial Indicator", "Writing Your Own Threaded Language", and "Interfacing Tips and Troubles". Write to Art at P. O. Box 1697, Kalispell, MT 59903.

```
==========================================================================
DOCUMENT :AAL-8409:DOS3.3:S.CLEAR.ARRAYS.txt
==========================================================================

 1000  *SAVE S.CLEAR ARRAYS
 1010  *--------------------------------
 1020  *   &CLEAR <ARRAY LIST>
 1030  *      SETS ALL VALUES IN REAL ARRAYS TO 0
 1040  *                     INTEGER ARRAYS TO 0
 1050  *                      STRING ARRAYS TO ""
 1060  *--------------------------------
 1070  *   WRITTEN BY JOHAN ZWIEKHORST, BASED ON
 1080  *   "CLEAR STRING ARRAY" BY BOB SANDER-CEDERLOF
 1090  *   IN DECEMBER, 1982 APPLE ASSEMBLY LINE
 1100  *--------------------------------
 1110  T.CLEAR    .EQ $BD  "CLEAR" TOKEN
 1120  *--------------------------------
 1130  ARYPT      .EQ $94
 1140  LOWTR      .EQ $9B
 1150  ARYEND     .EQ $9D  (= FAC)
 1160  *--------------------------------
 1170  CHRGET     .EQ $B1
 1180  CHRGOT     .EQ $B7
 1190  SYNERR     .EQ $DEC9
 1200  GETARYPT   .EQ $F7D9
 1210  *--------------------------------
 1220         .OR $300     (COULD BE ANYWHERE YOU LIKE)
 1230  *--------------------------------
 1240  CLEAR.ARRAYS
 1250         CMP #T.CLEAR      &CLEAR?
 1260         BEQ .3            ...YES
 1270  .1     JMP SYNERR
 1280  .2     CMP #$2C          COMMA?
 1290         BNE .1
 1300  *---GET STARTING ADDRESS---------
 1310  .3     JSR CHRGET   GET NEXT CHAR (SHOULD BE LETTER)
 1320         JSR GETARYPT FIND NAME/ADDRESS OF ARRAY
 1330         LDY #4       COMPUTE SIZE OF PREAMBLE
 1340         LDA (LOWTR),Y     # DIMENSIONS
 1350         ASL               *2, AND CLEAR CARRY
 1360         ADC #5            +5 (2 FOR NAME)
 1370         ADC LOWTR            (2 FOR OFFSET)
 1380         PHA                  (1 FOR # DIMS)
 1390         LDA LOWTR+1
 1400         ADC #0            ADD CARRY
 1410         STA ARYPT+1
 1420  *---GET ENDING ADDRESS-----------
 1430         CLC          ADD OFFSET TO GET ADDRESS OF END
 1440         LDY #2
 1450         LDA (LOWTR),Y
 1460         ADC LOWTR
 1470         STA ARYEND
 1480         INY
```

```
1490        LDA (LOWTR),Y
1500        ADC LOWTR+1
1510        STA ARYEND+1
1520 *---SET UP POINTER TO START------
1530        PLA
1540        TAY
1550        LDA #0
1560        STA ARYPT
1570        LDX ARYPT+1
1580 *---LOOP TO SET ELEMENTS ZERO----
1590 .4     STA (ARYPT),Y
1600        INY
1610        BNE .5       ...USUALLY
1620        INX          ...NEXT PAGE
1630        STX ARYPT+1
1640 .5     CPY ARYEND   AT END YET?
1650        BNE .4       ...NO
1660        CPX ARYEND+1
1670        BNE .4       ...NO
1680 *---CHECK IF ANOTHER ARRAY-------
1690        JSR CHRGOT
1700        BNE .2       ...YES, UNLESS SYNTAX ERROR
1710        RTS
```

```
=========================================================================
DOCUMENT :AAL-8409:DOS3.3:S.DP18AmperLink.txt
=========================================================================

 1000  *SAVE S.DP18 AMPER-LINK
 1010  *---------------------------------
 1020         .OR $803
 1030  *--------------------------------
 1040  *    APPLESOFT SUBROUTINES
 1050  *--------------------------------
 1060  AS.ADDON     .EQ $D998    ADD (Y) TO TXTPTR
 1070  AS.IF.JUMP   .EQ $D9DA    HANDLE T/F FOR IF
 1080  AS.FRMNUM    .EQ $DD67    EVAL FP FORMULA
 1090  AS.CHKCLS    .EQ $DEB8    CHECK FOR )
 1100  AS.CHKOPN    .EQ $DEBB    CHECK FOR (
 1110  AS.CHKCOM    .EQ $DEBE    CHECK FOR COMMA
 1120  AS.SYNCHR    .EQ $DEC0    CHARACTER SCAN OR FAIL
 1130  AS.SYNERR    .EQ $DEC9    SYNTAX ERROR
 1140  AS.PTRGET    .EQ $DFE3    FIND VARIABLE
 1150  AS.ISLETC    .EQ $E07D    LETTER CHECK
 1160  AS.FRMCPX    .EQ $E430    "FORMULA TOO COMPLEX" ERROR
 1170  AS.GETSPA    .EQ $E452    GET SPACE FOR STRING
 1180  AS.MOVSTR    .EQ $E5E2    MOVE STRING
 1190  *---------------------------------
 1200  *      PAGE ZERO USAGE
 1210  *--------------------------------
 1220  AS.VALTYP    .EQ $11    LAST FAC OP 0=NUM,FF=STRING
 1230  ARYTAB       .EQ $6B,6C
 1240  AS.FRESPA    .EQ $71,72
 1250  VARNAM       .EQ $81,82
 1260  AS.CHRGET    .EQ $B1
 1270  AS.CHRGOT    .EQ $B7
 1280  TXTPTR       .EQ $B8,B9
 1290  P2           .EQ $F9
 1300  *---------------------------------
 1310  *   DP18 SUBROUTINES ASSEMBLED ELSEWHERE
 1320  *---------------------------------
 1330  DP.PRINT            .EQ $FFFF
 1340  DP.INPUT            .EQ $FFFF
 1350  FIN                 .EQ $FFFF
 1360  DP.SGN              .EQ $FFFF
 1370  DP.INT              .EQ $FFFF
 1380  DP.ABS              .EQ $FFFF
 1390  DP.SQR              .EQ $FFFF
 1400  DP.LOGE             .EQ $FFFF
 1410  DP.EXPE             .EQ $FFFF
 1420  DP.COS              .EQ $FFFF
 1430  DP.SIN              .EQ $FFFF
 1440  DP.TAN              .EQ $FFFF
 1450  MOVE.DAC.YA         .EQ $FFFF
 1460  QUICK.FOUT          .EQ $FFFF
 1470  DP.POWER            .EQ $FFFF
 1480  DSUB                .EQ $FFFF
```

```
DADD                .EQ $FFFF
DMULT               .EQ $FFFF
DDIV                .EQ $FFFF
DP.ATN              .EQ $FFFF
DP.VAL              .EQ $FFFF
MOVE.YA.DAC         .EQ $FFFF
MOVE.YA.DAC.1       .EQ $FFFF
*--------------------------------
*   AMPERSAND VECTORS
*--------------------------------
         .DA DP18     STARTING ADDRESS FOR &-INTERPRETER
AMP.LINK .DA AS.SYNERR   LINK TO NEXT &-INTERPRETER
*-------------------------------
*      WORK AREAS FOR DPFP
*-------------------------------
WORK          .EQ *
SGNEXP        .BS 1
EXP           .BS 1
DGTCNT        .BS 1
DECFLG        .BS 1
*-------------------------------
DAC           .BS 12
DAC.EXPONENT  .EQ DAC
DAC.HI        .EQ DAC+1
DAC.EXTENSION .EQ DAC+10
DAC.SIGN      .EQ DAC+11
*-------------------------------
WRKSZ         .EQ *-WORK
*-------------------------------
ARG           .BS 12
*--------------------------------
*BUFFER FOR 'FOUT' AND
*LARGE ACC FOR MULTIPLICATION
*--------------------------------
FOUT.BUF      .BS 41
FOUT.BUF.SIZE .EQ *-FOUT.BUF
MAC           .EQ FOUT.BUF
*-------------------------------
STACK.SIZE    .EQ 12*10 10 ENTRIES BEFORE OVERFLOW
STACK.PNTR    .BS 1
STACK         .BS STACK.SIZE
RPAREN.CNT    .BS 1
*-------------------------------
REL.OPS       .BS 1
RESULT        .BS 2
INDEX         .BS 1
*-------------------------------
*      TOKEN ASSIGNMENTS
*-------------------------------
TKN.PLUS   .EQ 200  +
TKN.MINUS  .EQ 201  -
TKN.STAR   .EQ 202  *
TKN.SLASH  .EQ 203  /
TKN.POWER  .EQ 204  ^
```

```
2030  TKN.EQUAL  .EQ 208  =
2040  TKN.PRINT  .EQ 186  PRINT
2050  TKN.INPUT  .EQ 132  INPUT
2060  TKN.STR    .EQ 228  STR$
2070  TKN.IF     .EQ 173  IF
2080  TKN.THEN   .EQ 196  THEN
2090  TKN.GOTO   .EQ 171  GOTO
2100  TKN.NOT    .EQ 198  NOT
2110  TKN.AND    .EQ 205  AND
2120  TKN.OR     .EQ 206  OR
2130  *-------------------------------
2140  *      JMP TABLE FOR FUNCTIONS
2150  *--------------------------------
2160  DP.FUNC
2170         .DA DP.SGN-1        SGN (TKN 210)
2180         .DA DP.INT-1        INT
2190         .DA DP.ABS-1        ABS
2200         .DA AS.SYNERR-1     USR
2210         .DA AS.SYNERR-1     FRE
2220         .DA AS.SYNERR-1     SCRN(
2230         .DA AS.SYNERR-1     PDL
2240         .DA AS.SYNERR-1     POS
2250         .DA DP.SQR-1        SQR
2260         .DA AS.SYNERR-1     RND
2270         .DA DP.LOGE-1       LOG  #220
2280         .DA DP.EXPE-1       EXP
2290         .DA DP.COS-1        COS
2300         .DA DP.SIN-1        SIN
2310         .DA DP.TAN-1        TAN
2320  *      ATN HANDLED SPECIALLY
2330  *--------------------------------
2340  *--------------------------------
2350  *      &-INTERPRETER FOR DP18
2360  *--------------------------------
2370  NOT.DP18.CALL
2380         JSR AS.CHRGOT
2390         JMP (AMP.LINK)  SYNTAX ERROR OR NEXT CHAINED &-ROUTINE
2400  *--------------------------------
2410  *   & ENTRY POINT
2420  *--------------------------------
2430  DP18   CMP #'D'     CHECK FOR "DP:" AFTER "&"
2440         BNE NOT.DP18.CALL
2450         LDY #1
2460         LDA (TXTPTR),Y
2470         CMP #'P'
2480         BNE NOT.DP18.CALL
2490         INY          ADD 2 TO TXTPTR, TO POINT
2500         JSR AS.ADDON      AT NEXT CHAR AFTER "&DP"
2510  *--------------------------------
2520  DP.NEXT.CMD
2530         JSR AS.CHRGOT  SEE IF EOL
2540         BNE DP.SYNERR.1  ...NEITHER COLON NOR EOL
2550         TAY          CHECK FOR EOL
2560         BEQ .3       ...EOL, SO RETURN
```

```
2570        JSR AS.CHRGET     CHARACTER AFTER COLON
2580        BEQ .3       ...COLON OR EOL
2590        CMP #TKN.PRINT
2600        BEQ .1
2610        CMP #TKN.INPUT
2620        BEQ .2
2630        CMP #TKN.IF
2640        BEQ DP.IF
2650        JSR GET.A.VAR GET ADDRESS OF VAR
2660        LDX AS.VALTYP
2670        BMI DP.STR  STRING VAR
2680        JSR CHECK.DP.VAR
2690        STY RESULT+1   SAVE ADRS OF VARIABLE
2700        STA RESULT
2710        LDA #TKN.EQUAL   NEXT CHAR MUST BE "="
2720        JSR AS.SYNCHR  OR ELSE SYNTAX ERROR
2730        JSR DP.EVALUATE
2740        LDA RESULT
2750        LDY RESULT+1
2760        JSR MOVE.DAC.YA
2770        JMP DP.NEXT.CMD
2780 .1     JMP DP.PRINT
2790 .2     JMP DP.INPUT
2800 .3     RTS
2810 *---------------------------------
2820 DP.SYNERR.1
2830        JMP AS.SYNERR
2840 *---------------------------------
2850 *   <STRING> = STR$(<DPEXP>)
2860 *---------------------------------
2870 DP.STR STA P2            SAVE ADDR OF STRING VARIABLE
2880        STY P2+1
2890        LDA #TKN.EQUAL    MUST HAVE "="
2900        JSR AS.SYNCHR
2910        LDA #TKN.STR      MUST HAVE "STR$"
2920        JSR AS.SYNCHR
2930        JSR AS.CHKOPN     MUST HAVE "("
2940        JSR DP.EVALUATE   GET EXPRESSION
2950        JSR AS.CHKCLS     MUST HAVE ")"
2960        JSR QUICK.FOUT    CONVERT TO SIMPLE STR$ FORMAT
2970        DEC INDEX         DON'T COUNT TRAILING $00 BYTE
2980        LDA INDEX         GET LENGTH
2990        JSR AS.GETSPA     GET SPACE IN STRING AREA
3000        LDY #0            MOVE DATA INTO VARIABLE
3010        STA (P2),Y        LENGTH
3020        LDA AS.FRESPA
3030        INY
3040        STA (P2),Y        LO ADDRESS
3050        LDA AS.FRESPA+1
3060        INY
3070        STA (P2),Y        HI ADDRESS
3080        LDX #FOUT.BUF     COPY STRING DATA INTO PLACE
3090        LDY /FOUT.BUF
3100        LDA INDEX
```

```
        JSR AS.MOVSTR
        JMP DP.NEXT.CMD
*--------------------------------
*   IF <DPEXP> THEN <NORMAL STATEMENTS>
*   IF <DPEXP> THEN <LINE #>
*   IF <DPEXP> GOTO <LINE #>
*--------------------------------
DP.IF   JSR AS.CHRGET     GOBBLE THE IF
        JSR DP.EVALUATE   GET THE EXPRESSION
        JSR AS.CHRGOT     GET NEXT CHAR
        CMP #TKN.GOTO     GOTO?
        BEQ .1            ...YES
        LDA #TKN.THEN     ...NO, TRY "THEN"
        JSR AS.SYNCHR
.1      LDA DAC.EXPONENT  GET RESULT OF EXPRESSION
        JMP AS.IF.JUMP    LET APPLESFOT FIRMWARE DO IT
* AS.IF.JUMP COMPARES ACC TO 0. IF 0, IT SKIPS
* TO NEXT PROG. LINE. IF # 0, IT EXECUTES NEXT STATEMENT
*--------------------------------
*   GET VARIABLE NAME AND ADDRESS
*--------------------------------
GET.A.VAR
        JSR AS.ISLETC     1ST CHAR MUST BE LETTER
        BCC DP.SYNERR.1       NO, SYN ERR
        JMP AS.PTRGET GET ADRS OF VAR
*--------------------------------
*   CHECK IF VALID DP18 VARIABLE
*      ASSUME THIS ROUTINE CALLED AFTER "GET.A.VAR"
*      A DP18 VARIABLE MUST BE A REAL ARRAY
*--------------------------------
CHECK.DP.VAR
        CPY ARYTAB+1      BE SURE IT IS AN ARRAY
        BCC DP.SYNERR.1       NO, SYNTAX ERROR
        BNE .1       YES, AN ARRAY
        CMP ARYTAB
        BCC DP.SYNERR.1       NOT AN ARRAY, SYNTAX ERROR
.1      BIT VARNAM+1 BE SURE FLOATING POINT
        BMI DP.SYNERR.1       NO, SYNTAX ERROR
        RTS
*--------------------------------
*   EVALUATE DP18 EXPRESSION
*--------------------------------
DP.EVALUATE
        LDA #0            START WITH EMPTY STACK
        STA RPAREN.CNT    ...AND NO PARENTHESES
        STA STACK.PNTR
        JSR DP.ZERO       ZERO TO DAC
        JSR DP.EXP        EVALUATE AN EXPRESSION
        LDA STACK.PNTR    SHOULD BE BACK TO EMPTY STACK
        ORA RPAREN.CNT    AND NO PARENTHESES
        BNE DP.SYNERR.2   ...SYNTAX ERROR
        RTS               ...ALL OKAY!
*--------------------------------
*   GENERAL EXPRESSION
```

```
3650  *      EXP = RELAT
3660  *      EXP = EXP LOGOP RELAT
3670  *    LOGOP = "AND" OR "OR"
3680  *--------------------------------
3690  DP.EXP JSR DP.RELAT
3700  .1     JSR AS.CHRGOT
3710         CMP #TKN.AND
3720         BEQ .3
3730         CMP #TKN.OR
3740         BNE .6       ...FINISHED
3750  *---<EXP> OR <EXP>--------------
3760         JSR .5       GET NEXT RELAT
3770         ORA DAC.EXPONENT
3780         BNE .4       ...TRUE
3790  .2     JSR DP.FALSE ...FALSE
3800         JMP .1
3810  *---<EXP> AND <EXP>-------------
3820  .3     JSR .5       GET NEXT RELAT
3830         AND DAC.EXPONENT
3840         BEQ .2       ...FALSE
3850  .4     JSR DP.TRUE  ...TRUE
3860         JMP .1
3870  *---GET <EXP> AFTER RELOP--------
3880  .5     LDA DAC.EXPONENT
3890         PHA
3900         JSR AS.CHRGET
3910         JSR DP.RELAT
3920         PLA
3930  .6     RTS
3940  *--------------------------------
3950  DP.SYNERR.2
3960         JMP AS.SYNERR
3970  *--------------------------------
3980  *   RELATIONAL EXPRESSION
3990  *      RELAT = SUM
4000  *      RELAT = RELAT RELOP SUM
4010  *      RELOP = "<", "=", ">", "<=", "=<", ">=",
4020  *              "=>", "<>", OR "><"
4030  *--------------------------------
4040  DP.RELAT
4050         JSR DP.SUM   GET <EXP>
4060  .1     LDA #0
4070         STA REL.OPS
4080         JSR AS.CHRGOT
4090  .2     SEC          > IS $CF, = IS $D0, < IS $D1
4100         SBC #$CF     > IS 0, = IS 1, < IS 2
4110         BCC .4       ...NOT RELOP
4120         CMP #$03
4130         BCS .4       ...NOT RELOP
4140         ROL          > IS 0, = IS 2, < IS 4
4150         BNE .3       4 OR 2
4160         LDA #1       > IS 1
4170  .3     EOR REL.OPS  SET BITS IN REL.OPS: 00000<=>
4180         CMP REL.OPS  CHECK FOR REPEATED OPS
```

```
4190        BCC DP.SYNERR.2   ...YES, SYNTAX ERROR
4200        STA REL.OPS
4210        JSR AS.CHRGET     GET NEXT CHAR
4220        JMP .2       CHECK FOR <=> AGAIN
4230  *---PERFORM RELOP----------------
4240  .4     LDA REL.OPS  WERE THERE ANY?
4250        BEQ .8       NO, RETURN
4260        CMP #7       ALL THREE OPS?
4270        BEQ DP.SYNERR.2   ...YES, SYNTAX ERROR
4280        JSR PUSH.DAC.STACK  SAVE EXP1
4290        JSR DP.SUM          GET NEXT EXP2
4300        JSR POP.STACK.ARG   GET EXP1 IN ARG
4310        JSR DSUB            FORM EXP1 - EXP2
4320        LDA DAC.EXPONENT
4330        BEQ .45             EXP1 = EXP2
4340        LDA DAC.SIGN
4350        BMI .6              EXP1 < EXP2
4360        LDA REL.OPS         EXP1 > EXP2
4370        AND #$01            ">" OPERATOR?
4380        BEQ .7              ...NO, FALSE
4390        BNE .5              ...YES, TRUE
4400  .45    LDA REL.OPS  EXP1 = EXP2
4410        AND #$02     "=" OPERATOR?
4420        BEQ .7       ...NO, FALSE
4430  .5     JSR DP.TRUE  ...YES, TRUE
4440        JMP .1
4450  .6     LDA REL.OPS  EXP1 < EXP2
4460        AND #$04     "<" OPERATOR?
4470        BNE .5       ...YES, TRUE
4480  .7     JSR DP.FALSE ...NO, FALSE
4490        JMP .1
4500  .8     RTS
4510  *--------------------------------
4520  *   SUMMATION
4530  *      SUM = TERM
4540  *      SUM = SUM ADDOP TERM
4550  *    ADDOP = "+" OR "-"
4560  *--------------------------------
4570  DP.SUM JSR DP.TERM
4580  .1     JSR AS.CHRGOT
4590        CMP #TKN.PLUS
4600        BEQ .3       +
4610        CMP #'+
4620        BEQ .3       +
4630        CMP #TKN.MINUS
4640        BEQ .4       -
4650        CMP #'-
4660        BEQ .4       -
4670        RTS          END OF EXP
4680  .3     CLC          .CC. FOR +, .CS. FOR -
4690  .4     PHP          SAVE WHETHER + OR -
4700        JSR PUSH.DAC.STACK
4710        JSR AS.CHRGET
4720        JSR DP.TERM
```

```
4730        JSR POP.STACK.ARG
4740        PLP          .CC. FOR +, .CS. FOR -
4750        BCC .5
4760        LDA DAC.SIGN
4770        EOR #$FF
4780        STA DAC.SIGN
4790 .5     JSR DADD
4800        JMP .1
4810 *--------------------------------
4820 *   TERMS OF A SUMMATION
4830 *      TERM = FACTOR
4840 *      TERM = TERM MULOP FACTOR
4850 *     MULOP = "*" OR "/"
4860 *--------------------------------
4870 DP.TERM
4880        JSR DP.FACTOR
4890 .1     JSR AS.CHRGOT
4900        CMP #TKN.STAR     *?
4910        BEQ .2
4920        CMP #TKN.SLASH    / ?
4930        BEQ .3
4940        RTS
4950 .2     CLC          .CC. FOR *, .CS. FOR /
4960 .3     PHP          SAVE * OR / FLAG
4970        JSR PUSH.DAC.STACK
4980        JSR AS.CHRGET
4990        JSR DP.FACTOR
5000        JSR POP.STACK.ARG
5010        PLP          GET * OR / FLAG
5020        BCS .4       .../
5030        JSR DMULT    ...*
5040        JMP .1
5050 .4     JSR DDIV
5060        JMP .1
5070 *--------------------------------
5080 *   FACTORS OF A TERM
5090 *      FACTOR = ELEMENT
5100 *      FACTOR = FACTOR ^ ELEMENT
5110 *--------------------------------
5120 DP.FACTOR
5130        JSR AS.CHRGOT
5140        JSR DP.ELEMENT.1
5150 .1     JSR AS.CHRGOT
5160        CMP #TKN.POWER  ^?
5170        BEQ .2
5180        RTS          NO
5190 .2     JSR PUSH.DAC.STACK
5200        JSR DP.ELEMENT
5210        JSR POP.STACK.ARG
5220        JSR DP.POWER
5230        JMP .1
5240 *--------------------------------
5250 *   ELEMENTS OF A FACTOR
5260 *      ELEMENT = NUMBER, VARIABLE, OR FUNCTION()
```

```
*       ELEMENT = (EXP)
*       ELEMENT = UNARY ELEMENT
*         UNARY = "+" OR "-" OR "NOT"
*---------------------------------
DP.ELEMENT
       JSR AS.CHRGET
DP.ELEMENT.1
       CMP #TKN.PLUS     CHECK FOR UNARY +
       BEQ DP.ELEMENT    ...YES, JUST IGNORE IT
       CMP #TKN.MINUS    CHECK FOR UNARY -
       BNE .1            ...NO
       JSR DP.ELEMENT    GET THE EXP VALUE (RECURSIVE CALL)
       LDA DAC.SIGN   AND NEGATE IT
       EOR #$FF
       STA DAC.SIGN
       RTS
*---CHECK FOR (EXP)--------------
.1     CMP #'(
       BNE .2       ...NO
       INC RPAREN.CNT
       JSR AS.CHRGET  GET 1ST CHAR OF EXP
       JSR DP.EXP   (EXP)
       JSR AS.CHKCLS
       DEC RPAREN.CNT
       RTS
*---TRY VARIOUS FUNCTIONS--------
.2     TAY               SEE IF FUNCTION
       BPL DP.VARNUM     ...NO, TRY NUMBER OR VARIABLE
       CMP #TKN.NOT      "NOT"?
       BEQ .5            ...YES
       CMP #210          CHECK RANGE
       BCC DP.SYNERR.3   ...NOT VALID DP FUNCTION
       CMP #229          MAY BE "VAL"
       BEQ .4            ...VAL(STRING)
       CMP #225          ATN?
       BCC .3            ...NO, BUT IN RANGE FOR OTHERS
       BNE DP.SYNERR.3   ...NOT VALID DP18 FUNCTION
       JMP DP.ATN
.3     SBC #209          CARRY CLEAR SUBS 1 MORE
       ASL               MULT BY 2
       TAY               INDEX INTO TABLE
       LDA DP.FUNC+1,Y   GET HI ADR
       PHA
       LDA DP.FUNC,Y     GET LO ADR
       PHA
       JSR AS.CHRGET
       JSR AS.CHKOPN MUST HAVE (
       INC RPAREN.CNT
       JSR DP.EXP   EVALUATE ARG
       JSR AS.CHKCLS
       DEC RPAREN.CNT
       RTS          EVALUATES FUNCTION
*---"VAL" FUNCTION--------------
.4     JSR AS.CHRGET
```

```
5810        JSR AS.CHKOPN
5820        JSR DP.VAL
5830        JMP AS.CHKCLS
5840 *---"NOT" ELEMENT----------------
5850 .5     JSR DP.ELEMENT    GET ARGUMENT (RECURSIVE CALL)
5860        LDA DAC.EXPONENT
5870        BEQ DP.TRUE
5880 *      FALL INTO DP.FALSE
5890 *--------------------------------
5900 DP.ZERO
5910 DP.FALSE
5920        LDA #0       FALSE, PUT 0 IN DAC
5930        LDY #11
5940 .1     STA DAC,Y
5950        DEY
5960        BPL .1
5970        RTS
5980 *--------------------------------
5990 DP.TRUE
6000        LDA #CON.ONE      TRUE, PUT 1 IN DAC
6010        LDY /CON.ONE
6020        JMP MOVE.YA.DAC
6030 *--------------------------------
6040 DP.SYNERR.3 JMP AS.SYNERR
6050 *--------------------------------
6060 *   VARIABLE OR NUMBER
6070 *      VARNUM = DP18 VARIABLE
6080 *      VARNUM = NUMBER
6090 *      VARNUM = NEGOP NUMBER
6100 *      VARNUM = "PI"
6110 *--------------------------------
6120 DP.VARNUM
6130        LDY #0
6140        LDA (TXTPTR),Y
6150        CMP #'P      CHECK FOR PI
6160        BNE .1
6170        INY          Y=1
6180        LDA (TXTPTR),Y
6190        CMP #'I
6200        BNE .1
6210        INY          Y=2
6220        LDA (TXTPTR),Y
6230        CMP #'(      MUST NOT BE ARRAY
6240        BEQ .1
6250        JSR AS.ADDON      ADVANCE TXTPTR PAST "PI"
6260        LDA #CON.PI
6270        LDY /CON.PI
6280        JMP MOVE.YA.DAC.1 GET PI INTO DAC W/GUARD DIGITS
6290 *---CHECK FOR VARIABLE-----------
6300 .1     JSR AS.CHRGOT
6310        JSR AS.ISLETC
6320        BCC .2            ...NOT LETTER, TRY NUMBER
6330        JSR AS.PTRGET     ...LETTER, GET VARIABLE ADDR
6340        JSR CHECK.DP.VAR  BE SURE IT IS REAL ARRAY
```

```
6350        JMP MOVE.YA.DAC   GET VALUE INTO DAC
6360 *---CHECK FOR NUMBER-------------
6370 .2     CMP #'.      DECIMAL POINT?
6380        BEQ .3       YES
6390        CMP #TKN.PLUS PLUS?
6400        BEQ .3       YES
6410        CMP #TKN.MINUS MINUS?
6420        BEQ .3       YES
6430        CMP #'0
6440        BCC DP.SYNERR.3       NOT A DIGIT
6450        CMP #'9+1
6460        BCS DP.SYNERR.3       NOT A DIGIT
6470 .3     JMP FIN      CONVERT NUMBER
6480 *--------------------------------
6490 *      PUSH (DAC) ONTO EXPRESSION STACK
6500 *--------------------------------
6510 PUSH.DAC.STACK
6520        LDY STACK.PNTR
6530        CPY #STACK.SIZE-12
6540        BCS .2       STACK ALREADY FULL
6550        LDX #0
6560 .1     LDA DAC,X
6570        STA STACK,Y
6580        INY
6590        INX
6600        CPX #12      STACK 12 BYTES
6610        BCC .1
6620        STY STACK.PNTR
6630        RTS
6640 .2     JMP AS.FRMCPX  FORMULA TOO COMPLEX
6650 *--------------------------------
6660 *      POP EXPRESSION STACK INTO ARG
6670 *--------------------------------
6680 POP.STACK.ARG
6690        LDY STACK.PNTR
6700        BEQ DP.SYNERR.3   STACK IS EMPTY
6710        LDX #11
6720 .1     DEY
6730        LDA STACK,Y
6740        STA ARG,X
6750        DEX
6760        BPL .1
6770        STY STACK.PNTR
6780        RTS
6790 *--------------------------------
6800 CON.ONE    .HS 411000000000000000000
6810 CON.PI     .HS 4131415926535897932385
6820 *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8409:DOS3.3:S.INDEX.MASK.txt
==========================================================================

1000   .LIF
1010  *SAVE S.INDEX --> MASK
1020  *--------------------------------
1030  TEST   LDY #"0"
1040  .1     TYA
1050         JSR $FDED
1060         TYA
1070         JSR TRICKY.WAY
1080         JSR HEX
1090         TYA
1100         JSR TRICKIER.WAY
1110         JSR HEX
1120         TYA
1130         JSR SHIFT.LOOP
1140         JSR HEX
1150         TYA
1160         JSR TABLE.LOOKUP
1170         JSR HEX
1180         TYA
1190         JSR SILLY.WAY
1200         JSR HEX
1210         TYA
1220         JSR NOT.SO.SILLY.WAY
1230         JSR HEX
1240         TYA
1250         JSR TRICKY.WAY.R
1260         JSR HEX
1270         TYA
1280         JSR TRICKIER.WAY.R
1290         JSR HEX
1300         TYA
1310         JSR SHIFT.LOOP.R
1320         JSR HEX
1330         TYA
1340         JSR TABLE.LOOKUP.R
1350         JSR HEX
1360         JSR $FD8E
1370         INY
1380         CPY #"8"
1390         BCC .1
1400         RTS
1410  *--------------------------------
1420  HEX    PHA
1430         LDA #"-"
1440         JSR $FDED
1450         PLA
1460         JMP $FDDA
1470  *--------------------------------
1480  TRICKY.WAY.R
```

```
1490        EOR #7
1500 TRICKY.WAY
1510        AND #7
1520        CMP #2
1530        BCC .5       1+1=$02
1540        BEQ .5       2+1+CARRY=$04
1550        CMP #4
1560        BCC .4       3+1+4=$08
1570        BEQ .3       4+1+4+CARRY+6=$10
1580        CMP #6
1590        BCC .2       5+1+4+6+$10=$$20
1600        BEQ .1       6+1+4+6+$10+CARRY+$1E=$40
1610        ADC #$3F     7+1+4+6+$10+$1E+CARRY=$80
1620 .1     ADC #$1E
1630 .2     ADC #$10
1640 .3     ADC #6
1650 .4     ADC #4
1660 .5     ADC #1
1670        RTS
1680 *--------------------------------
1690 TRICKIER.WAY.R
1700        EOR #7
1710 TRICKIER.WAY
1720        AND #7       00-01-02-03-04-05-06-07
1730        SEC
1740        ROL          01-03-05-07-09-0B-0D-0F
1750        CMP #3
1760        BCC .0       TURN 00 INTO 01
1770        CMP #7
1780        BCC .12      TURN 03 INTO 02, 05 INTO 04
1790        ADC #6       ..-..-..-0E-10-12-14-16
1800        CMP #$12
1810        BCC .34      TURN 0E INTO 08, 10 INTO 10
1820        ADC #$2B     ..-..-..-..-..-..-3E-40-42
1830        CMP #$42
1840        BCC .56      TURN 3E INTO 20, 40 INTO 40
1850        ASL          TURN 42 INTO 84
1860 .56    AND #$E0     MASK 3E-40-84 TO 20-40-80
1870 .34    AND #$F8     MASK 0E-10-20-40-80 TO 08-10-20-40-80
1880 .12    AND #$FE     MASK 03-05... TO 02-04...
1890 .0     RTS
1900 *--------------------------------
1910 SHIFT.LOOP
1920        AND #7
1930        TAX
1940        LDA #1
1950 .1     ASL
1960        DEX
1970        BPL .1
1980        ROR
1990        RTS
2000 *--------------------------------
2010 TABLE.LOOKUP
2020        AND #7
```

```
2030        TAX
2040        LDA TABLE,X
2050        RTS
2060  TABLE  .HS 0102040810204080
2070  *--------------------------------
2080  SHIFT.LOOP.R
2090        AND #7
2100        TAX
2110        LDA #$80
2120  .1    LSR
2130        DEX
2140        BPL .1
2150        ROL
2160        RTS
2170  *--------------------------------
2180  TABLE.LOOKUP.R
2190        AND #7
2200        TAX
2210        LDA RTABLE,X
2220        RTS
2230  RTABLE .HS 8040201008040201
2240  *--------------------------------
2250  SILLY.WAY
2260        AND #7
2270        BEQ .0
2280        CMP #1
2290        BEQ .1
2300        CMP #2
2310        BEQ .2
2320        CMP #3
2330        BEQ .3
2340        CMP #4
2350        BEQ .4
2360        CMP #5
2370        BEQ .5
2380        CMP #6
2390        BEQ .6
2400        LDA #$80
2410        RTS
2420  .6    LDA #$40
2430        RTS
2440  .5    LDA #$20
2450        RTS
2460  .4    LDA #$10
2470        RTS
2480  .3    LDA #$08
2490        RTS
2500  .2    LDA #$04
2510        RTS
2520  .1    LDA #$02
2530        RTS
2540  .0    LDA #$01
2550        RTS
2560  *--------------------------------
```

```
2570  NOT.SO.SILLY.WAY
2580         AND #7
2590         BEQ .0
2600         CMP #4
2610         BEQ .4
2620         BCS .60
2630         CMP #2
2640         BEQ .2
2650         BCS .3
2660         LDA #$02
2670         RTS
2680  .60    CMP #6
2690         BEQ .6
2700         BCS .7
2710         LDA #$20
2720         RTS
2730  .7     LDA #$80
2740         RTS
2750  .6     LDA #$40
2760         RTS
2770  .4     LDA #$10
2780         RTS
2790  .3     LDA #$08
2800         RTS
2810  .2     LDA #$04
2820         RTS
2830  .0     LDA #$01
2840         RTS
```

```
=======================================================================
DOCUMENT :AAL-8409:DOS3.3:TWIRLERS.txt
=======================================================================

   1   .LIF
1000  *SAVE TWIRLERS
1010  *--------------------------------
1020  T
1030  .1     TXA
1040         AND #3
1050         TAX
1055         LDA CHARS,X
1060         JSR FILL
1070         INX
1075         LDA #180
1080         JSR $FCA8
1090         LDA $C000
1100         BPL .1
1110         STA $C010
1120         RTS
1130  CHARS  .HS A1AFADDC
1140  *--------------------------------
1150  FILL   LDY #4
1160         STY 1
1170         STY 2
1180         LDY #0
1190         STY 0
1200  .1     STA (0),Y
1210         INY
1220         BNE .1
1230         INC 1
1240         DEC 2
1250         BNE .1
1260         RTS
1270  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8410:Articles:Arctec.Ad.txt
==============================================================================

=============================================================================
DOCUMENT :AAL-8410:Articles:DP18.Correction.txt
=============================================================================

Correction to DP18, Part 5.......................Paul Schlyter

The following comments relate to the listing on page 13 of the September 1984 issue.

It appears to me that lines 4610-4620 and 4650-4660 can be deleted. They check for the non-tokenized forms of "+" and "-", which I believe will never be presented to DP18.

There is a definite bug at line 4460: the "LDA #$02" should be "LDA #$04". Compare with lines 4370 and 4410, and you will see how I caught it. Also the comment on line 4170, which says the bit map is in the form "00000<=>".

-----------------------------------------------------------------------------
DOCUMENT :AAL-8410:Articles:DP18.txt
-----------------------------------------------------------------------------

18-Digit Arithmetic, Part 6................Bob Sander-Cederlof

This month's installment will cover some of the elementary functions: VAL, INT, ABS, SGN, and SQR. I will also introduce a general polynomial evaluator, which will be used by most of the other math functions.

Most of the functions expect a single argument, which will be loaded into DAC by the expression evaluator just before calling the function code. The function code will compute a value based on the argument, and leave the result in DAC. As the expression evaluator calls with JSR, the function code returns with RTS.

One exception to the above paragraph is the VAL function. VAL processes a string expression, and converts it into a value in DAC. The code in lines 1350-1610 of the listing closely parallels the VAL code in the Applesoft ROMs. Lines 1350-1370 evaluate the string expression. Lines 1380-1460 save the current TXTPTR value (which points into your Applesoft program), and makes TXTPTR point instead at the resulting string. Lines 1470-1520 save the byte just past the end of the string and store a 00 terminator byte in its place. FIN will evaluate the string, placing the numeric value into DAC. Then lines 1540-1600 restore the byte after the string and TXTPTR.

The INT function zeroes any digits after the decimal point in a number. A number in DAC has 20 digits. The exponent will be $00 if the value is zero, $01-40 if the value is all fractional, $41-53 if the value has from 1 to 19 digits before the decimal point, or $54-7F if the value has no fractional digits.

Lines 1650-1700 remove the $40 bias from the exponent. If the exponent was $00-40, DP.ZERO will force DAC to zero. Lines 1730-1740 check for the case of no fractional digits, and exit immediately. Lines 1750-1860 zero the digits after the decimal point. If the exponent was odd, there is one digit to be removed in the first byte to be cleared; the rest get both digits zeroed.

The simplest function is ABS, or absolute value. All it requires is forcing the sign positive, handled at lines 1910-1930.

Almost as simple is SGN, or sign function. SGN returns -1, 0, or +1, according as DAC was negative, zero, or greater-than- zero. Lines 1970-1980 check DAC.EXPONENT, which will be zero if-and-only-if DAC is zero. If the value is not zero, lines 1990-2030 force the value to be 1.0, while retaining the original sign.

SQR, the square root function, is more interesting. Do you remember the way you learned to take square roots in high school? Neither do I, but there is a handier way in computers anyway.

Suppose I want to find square root of 25. I could start with a wild guess, check it to see if I am close by squaring and comparing with 25, and then refining my guess until it is as accurate as I need. Suppose my wild guess is 7 (pretty wild!).

7*7 is 49, which is bigger than 25, so my next guess should be less than 7. Instead of just guessing wildly for the next one, why not take the average between 7 and 25/7? That average is 5.286. The average of 5.286 and 25/5.286 is 5.0076. The next one is 5.0000079. You can see that I am rapidly approaching the answer of 5.0.

The method of refining an approximation as exemplified above was derived originally be Sir Isaac Newton. His method involves calculus, can get quite complex, and applies to all sorts of problems. But in the case of the square root, it is as simple as averaging an approximation with the argument divided by the approximation.

It turns out that it is a very good method, because if you can get an initial approximation that has the first few digits right, the number of digits that are correct will slightly more than double each time you run through Newton's improver.

The next trick is to reduce the range of possible arguments from the full range of zero to 10^63 down to the range from .1 to 1.0. The zero case is easy, because SQR(0) = 0, and is handled at lines 2100-2110. Notice that lines 2120-2130 weed out negative arguments, which are not allowed.

Remember that the square root of X*10^n is equal to SQR(X)*10^(n/2). Lines 2150-2190 save the exponent, and change it to $40. This changes the value in DAC to the range .1 to 1.0. I have a book which gives polynomial approximations to the square root in that range. One with the form aX^4+bX^3+...+e gives an approximation with is accurate in the first 2.56 digits. Three iterations by Newton yield more than 22 accurate digits. The same book shows a cubic polynomial which gives 2.98 accurate digits if we can get the value into the range between .25 and 1.0.

Lines 2200-2280 fold the values between .1 and .25 up to the range .4 through 1.0 by multiplying the value by 4. (This multiplication goes pretty fast, since most of the bytes are zero.) The fact that we quadrupled the value is remembered, so that we can later halve the approximate root at lines 2350-2410. The cubic polynomial is evaluated in lines 2290-2340, by calling POLY.N. The result, by the time we reach line 2420, is an approximate square root of the number between .1 and 1; now we need to make it an approximate root of the original argument.

Lines 2420-2480 compute the exponent of the square root, by simply dividing the original exponent by two. If there is a remainder, meaning the original exponent was odd, then we also need to multply the trial root by SQR(10). This is handled in lines 2490-2550. The halved original exponent next is added to the trial exponent, giving a good first approximation to the square root of the original argument. Lines 2600-2740 run through three cycles of the Newton iteration, giving plenty of precision. If we were carrying enough digits along, the 2.98 digits of precision our polynomial produced would be refined to a full 26 digits, according to my book.

Speaking of the book, it is one I bought a number of years ago when working on double precision math for a Control Data 3300 time sharing system. As far as I know, it is still the best book in its field. "Computer Approximations", by J. F. Hart and about seven other authors, was published in 1968 by John Wiley & Sons. I don't know if it is still in print or not, but if you ever need to create some high precision math routines, you ought to try to find a copy.

A very common element in the evaluation of many math functions is an approximation to the function over a limited range by a polynomial, or by the quotient of two polynomials. Therefore it is handy to have an efficient subroutine to evaluate a polynomial. Two different entry points allow efficient evaluation of two kinds: those whose first coefficient is 1, and the rest. POLY.N evaluates those whose first coefficient is not one, and POLY.1 does those whose first is 1.

```
       POLY.N --  a*x^n + b*x^n-1 + ...
       POLY.1 --    x^n + a*x^n-1 + ...
```

In both cases, you enter with the address of a table of coefficients in the Y- and A-registers (hi-byte in Y, lo-byte in A), and the degree of the polynomial in the X-register. Thus you see that in lines 2290-2340 the table P.SQR is addressed, and the degree of polynomial is 3 (cubic). Both POLY.N and POLY.1 assume that the value of x is in TEMP2. Where all terms have been computed and added, the result will be in DAC.

Actually, I may have misled you a little in the last sentence. The terms of the polynomial are not separately computed and added, but rather they are accumulated in a simple serial fashion:

```
       poly = ((( a * x + b) * x + c) * x + d) * x + e
```

The coefficients and other constants shown in lines 2770-2830 are in a special format which includes an extra two digits. You will remember that the basic operations (+-*/) are carried out to 20 digits. Therefore these constants are carried out to 20 digits. They are not critical in the square root computation, thanks to Sir Isaac, but the log and trig functions will need them.

```
==============================================================================
DOCUMENT :AAL-8410:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 5 -- Issue 1                     October, 1984

In This Issue...



Index to Volume 4

This time last year we published a cumulative index to the first three years of Apple Assembly Line.  In this issue we add a separate index to Volume 4, covering October 83 through September 84.  Perhaps in another year or two we can do another complete index.

65802 is Here!

After nearly a year of more or less patient waiting, we finally have a sample 65802 microprocessor.  It does indeed plug right into an Apple //e, and works just fine.  See Bob's story inside for all the details.

Blind Word Processor

Subscriber Larry Skutchan, of Little Rock, Arkansas, has adapted the S-C Word Processor to work with the Echo Two Speech Synthesizer.  He now has a special word processor for the blind, which he says is the best available.  The price will be $95.50.  Larry is a blind university student, majoring in Computer Science.  You can reach him at (501) 568-2172.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.50 each (other countries add $1 per back issue for postage).

```
==================================================================
DOCUMENT :AAL-8410:Articles:Graphics.SW.txt
==================================================================

Apple II Graphics Software
--------------------------

Accent Software        The Graphic Solution    $150
(415) 949-2711

Avant Garde            Graphics Applications   $10
(503) 345-3043         Hi-Res Secrets          $130
                       Paintmaster             $35
                       Starsprite (with board) $395
                       Super Shape Draw        $35
                       Ultra Plot              $70

Beagle Brothers        Alpha Plot              $40
(619) 296-6400         Apple Mechanic          $30
                       Flex Type               $30
                       Type Faces              $20

Broderbund             Arcade Machine          $60
                       Paint Shop              $50

Business and Pro-      Business Graphics       $350
  fessional Software
(617) 491-3377

Calif Pacific Comp.    Bill Budge's 3-D Animation  $40

Castle Designs         Fastdraw 1.1            $??

Chalkboard             Chalkboard w/ software      $100

Computer Station       Enhanced Graphic Soft.  $35
(314) 432-7019         Graphic Writer          $55
                       Ultra Hi-Res Graphics   $50

Datasoft               Micropainter            $35
(818) 701-5161

Decision Resources     Micropainter            $375
(203) 222-1974

Desktop Computer Soft. Graph 'n' Calc          $195
(408) 458-9095

Dickens Data Syst.     Super Plotter           $70
(404) 448-6177         Wall Street Plotter     $125

Ferox Microsyst.       Graphpower              $295
(703) 841-0800
```

| | | |
|---|---|---|
| Frank Belanger<br>(512) 451-6868 | Shapemaker | $?? |
| Funk Software<br>(617) 497-6339 | Sideways | $60 |
| Graphware<br>(513) 424-6733 | Charts Unlimited | $195 |
| Insoft | GraFORTH | $90 |
| Interactive Microware<br>(814) 238-8294 | Curve Fitter<br>Quick-Draft<br>Scientific Plotter | $35<br>$50<br>$25 |
| Koala Technologies | Koala Tablet w/ soft.<br>Gibson Light Pen | $125<br>$250 |
| Laumer Research<br>(214) 245-3927 | Mike's Magic Matrix | $?? |
| Mad West Software | Ampergraph | $45 |
| Micro Lab<br>(312) 433-7550 | Multigraph<br>Painter Power | $150<br>$40 |
| Muse Software | Data Plot | $60 |
| Penguin Software<br>(312) 232-1984 | Complete Graphics Sys.<br>Graphics Magician<br>Gr. Mag. Pict. Painter<br>Map Pack<br>Additional Type Sets<br>Paper Graphics<br>Special Effects | $80<br>$60<br>$50<br>$20<br>$20<br>$50<br>$40 |
| Primesoft<br>(301) 229-4229 | The Prime Plotter | $240 |
| Scarborough Syst. | PictureWriter | $40 |
| Sensible Software<br>(313) 399-8877 | Image Printer<br>Graphics Department | $50<br>$125 |
| Softalk Publishing | SoftGraph (David Durkee) | free |
| Software Publishing | PFS:Graph | $125 |
| Synergistic Software | Higher Graphics II<br>Higher Text<br>Higher Fonts | $35<br>$??<br>$15 |
| Visicorp | Visitrend/Visiplot | $99 |

(408) 946-9000

| | | |
|---|---|---|
| Zoom Software | Zoom Grafix | $50 |

=================================================================
DOCUMENT :AAL-8410:Articles:Index.2.Vol.4.txt
=================================================================

Index to Articles in "Apple Assembly Line", Volume 4

```
===========================================================================
DOCUMENT :AAL-8410:Articles:LCR.Correx.txt
===========================================================================
```

Corrections to Line Number Cross Reference..........Bill Morgan

Allen Miller just called up from Hong Kong (at 3:30 AM his time!) to report a problem with the Line Number Cross Reference program in the August issue. It seems that as published it only prints out the first line number list in each chain. The troublemaker is line 4560, which says BNE .1. Well .1 is the next line, so the routine is always exiting after only one pass. Line 4560 should read BNE PRINT.CHAIN, to go back to the beginning rather than on to the end.

Then Chuck Welman called to point out yet another problem. It seems that an undefined line number greater than the last line of the program caused LCR to head off into the wilderness. When I investigated this one it proceeded to get even stranger. LCR would hang only if the undefined line number was greater than 19668! Less than 19668 came out just right, and equal to 19668 worked, but LCR mistakenly said the line was defined. Now here was a real creepy crawler of a bug!

Well the problem turned out to be in the CHECK.DEFINITION routine. Here are the offending lines:

```
4790 .4    LDY #0
4800       LDA (PNTR),Y    lo-byte of next line address
4810       PHA
4820       INY
4830       LDA (PNTR),Y    and hi-byte
4840       STA PNTR+1
4850       PLA
4860       STA PNTR
4870       JMP CHECK.DEFINITION
```

This code is called when CHECK.DEFINITION wants to get the next line of the Applesoft program. The trouble comes up because there is no check for end-of-program. Sooner or later we come to the zero bytes that mark the end, load up PNTR with zeroes, and go back to CHECK.DEFINITION to try what seems to be the next line. That routine then compares the line number we are checking to the contents of locations 2 and 3 of memory, which Applesoft has loaded with D4 and 4C. Now $4CD4 equals 19668, so that's where that funny number came from!

Here is a slightly rearranged, working version of lines 4790-4870. Note that we have reversed the hi-lo byte sequence and added a check for a zero hi-byte:

```
4790 .4    LDY #1
4800       LDA (PNTR),Y    hi-byte of next line address
4805       BEQ .2          end of program?
```

```
4810        PHA
4820        DEY
4830        LDA (PNTR),Y    and lo-byte
4840        STA PNTR
4850        PLA
4860        STA PNTR+1
4870        JMP CHECK.DEFINITION
```

```
=============================================================================
DOCUMENT :AAL-8410:Articles:Mac.Assemblers.txt
=============================================================================
```

Macintosh Assemblers................................Lane Hauck
San Diego, CA

I have the privilege* of Beta testing two 68000 assemblers for the Macintosh -- the one from Apple (Workshop), and the one from Mainstay. Mainstay is the "serious" side of Funsoft.

* (If you are masochistic, and enjoy little surprises like alert boxes with no messages or GoAwayButtons in them, frequent crashes, and system fonts abruptly changing; you too might want to become a Beta Tester.)

I've gotten permission from both Apple and Mainstay to talk about these products. The versions I'm testing are preliminary, and therefore subject to change.

The Workshop is in "version 0.6" release, and is expected to be available about October (I'd guess November). The Mainstay product is scheduled for early October release, and judging from their staff and working hours, I think they'll make it. (I visited them in Agoura, CA, and found a very smart and hard working group of programmers.)

Although both assemblers do the same thing -- translate 68000 source programs into runnable programs on the Macintosh -- they couldn't be more different in how they operate!

The Apple Assembler

The Workshop has several parts. EDIT, ASM, LINK and EXEC are four applications that do the actual code development. Additionally, RMAKER creates resource files from text source files created by EDIT. And finally, MacDB and its associated "Nub" programs provide debug support for when your code doesn't run.

The development system can run on one drive, but two are highly recommended.

EDIT: This is a DISK BASED editor, so the short document frustrations of MacWrite are avoided. Additionally, you can open up to four documents, and cut and paste between them (a la Lisa)! This is a bare bones (but wonderful) editor, without fancy fonts or formatting. One improvement over the Lisa editor: it has a "reverse tab" -- hitting backspace from a tab stop takes you back not one space, but back one tab position. This is a great convenience when you're entering formatted source code.

ASM: Supports conditional assembly, macros (both "Lisa-type" and new "Mac-type"). It's tailored to the Mac development environment (for example it helps you write relocatable code).

Toolbox support is provided by special, compressed equate files (they are compressed by a program called PacSyms, which you can use to compress your own equate files). The Workshop provides all the Trap and symbol equates mentioned in Inside Macintosh.

LINK: Links ".REL" code modules produced by the Assembler, and (eventually -- not working yet) output files from RMAKER to produce a final file, complete with your code and resources. Takes its direction from a ".LINK" text file.

EXEC: Lets you automate the entire ASM-LINK process. One great improvement over the Lisa version: you can direct EXEC to reenter the editor if any assembly or link errors occur.

FIVE debuggers are supplied. MacDB is the best, most visible debugger I'ver ever seen. It requires two Macs (or one Mac and a Lisa running MacWorks). The Workshop will be supplied with an interconnect cable for two Macs. Other debugger versions (which don't require the second Mac) let you debug from an 8-line onscreen window on the Mac, and from any remote terminal.

This is a professional, complete, "industrial strength" 68000 assembly language development package. Its utilization of the Macintosh environment is total and outstanding. My only real quibble is that it takes a fair amount of time (a few minutes) to "turn" one cycle from EDIT to running the new code. A hard disk would presumably improve this greatly.

If you're an "interactive" programmer who likes to make changes and see their results QUICKLY, you might be interested in the Mainstay Assembler.

The Mainstay Assembler

If you've ever used any of the assemblers for the Apple II from S-C Software, you'll feel right at home with the Mainstay environment. It's patterned after the S-C 68000 Cross Assembler, and it looks and feels just like you're running on an Apple //e!

The fact that none of the Macintosh interface is used will bother some, especially the Mac purists. Mainstay's intention is to get a quality assembler to market quickly, and the approach they've taken allows this to happen. I don't mind non-fidelity to the Mac interface in a DEVELOPMENT product -- we developers are EXPECTED to put up with all sorts of indignities!

This is an absolute assembler, meaning that your code module is produced with an address origin, and it is loaded and run at that address. It does not produce "linkable" code modules, as does the Apple Workshop Assembler. In fact no linker is supplied or required.

The Editor is built in, and it functions much like the Apple II. The cursor is moved around with keyboard commands. The Editor has BASIC-like line numbers and the normal complement of line-number oriented commands (RENumber, COPY, MOVE, etc.).

Resources are handled right inside your source code (remember there is only one code "module"). This is more convenient than the Apple "RMAKER" approach.

The Assembler supports conditional assembly, macros, and local labes. It takes a novel approach in how it is installed and run on the Macintosh.

When you start the Assembler, it grabs a large chunk of memory from the application heap, and uses it for storing the symbol table, source code, and object code. Typing MEM shows you exactly where these three memory areas are. While you're in the Assembler environment, your code "stays put", so you can deal with absolute addresses without fear that the memory manager will move things around on you.

This means that you can edit, assemble, and test your code IMMEDIATELY, without goin through a linking and (optionally) a resource compiling step. This is the primary strength of this assembler -- it allows "quicklook" programming which is ideal for experimentation and learning the Macintosh system.

Eventually you will want to make your application an "installable" Macintosh program, so you should get into the habit of writing position independent code. The Mainstay package will supply the tools necessary to make your application runnable on the Mac. It will also contain Toolbox and Operating System equate files.

There are some nice "Apple II-like" features, such as typing DIR to look at the disk catalog. In the Mac environment, you have to exit the application and get back to the desktop to see your files. You can also type "EJECT" and eject a disk immediately. I like to do this just before running new code, to protect disks from my runaway test programs that mysteriously fire up the disk drive.

Having this assembler, a Mac, and a copy of INSIDE MACINTOSH might just be the most efficient way to learn the Macintosh. The prime benefit of this assembler is its very high speed in moving between editing, assembling, and running your test code.

Which One?

Which assembler would I recommend? At this stage I'd have to give the universal Computer Salesman answer: "It depends."

The Apple one allows you to write separate code modules, assemble them, and then link them together later. This allows you to utilize already written and debugged modules in new programs.

Another advantage of the "linker" approach is that a single module can be changed and reassembled, and then linked to other already-debugged modules. This saves reassembling the whole shebang every time you make a change.

If you like this "relocatable assembler" approach, you'll want the Apple Assembler. (If you're comfortable with the Lisa Assembler, ditto.)

The Mainstay Assembler, by contrast, is an absolute assembler - it puts code at a particular place in memory (set by an ORG - origin statement), and allows only one "module" -- your entire program. (Better write relocatable code if you want it to run as an application, though!)

The Mainstay Assembler is so fast (especially if you put a "LIST OFF" directive at the beginning of your code), that it negates the speed advantage of the linked module approach. I would guess that it takes you from source code edit to running reassembled code in about one-twentieth the time required by the Apple Assembler. if you're an "interactive" programmer who likes to see results of program changes FAST, the Mainstay Assembler is for you.

If time is a factor, the Mainstay product will ship within a week; the Apple Assembler is supposed to come out in October, but I doubt it.

If you're unhappy with "non-Mac-user-interface" products, you're better off with the Apple version. The operation of the Mainstay assembler is a bit strange at first, but anyone with Apple II roots will adjust quickly.

Here's a factor I consider very important: Apple is a "Pascal house" with almost no support given to assembly language programming of the Macintosh. I've found their support in this area dismal.

The Mainstay Assembler is a major committment by this small company. I've had quite a bit of technical interaction with them, and have found them to be very intelligent, motivated, and responsive. I've had indications that you'll be able to expect not only Assembler support from Mainstay, but also some Macintosh support as well.

--------

[ 10/15 -- The folks at Mainstay tell me they started shipping last week, so we should have some copies for sale by the time you read this. The introductory price is $100. -- Bill ].

```
=================================================================
DOCUMENT :AAL-8410:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1..................................$92.50
Version 1.1 Update...............................................$12.50
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM..................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Quick-Trace (Anthro-Digital)..............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor......................$60
Amper-Magic (Anthro-Digital)..............................(reg. $50)  $45
Amper-Magic Volume 2 (Anthro-Digital).....................(reg. $30)  $25
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                         or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
QuikLoader EPROM System (SCRG)...............................($179) $170
D MAnual Controller (SCRG)....................................($90)  $85
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32

Books, Books, Books.........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
```

```
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
```

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

=============================================================================
DOCUMENT :AAL-8410:Articles:Odd.Ways.txt
=============================================================================

Another Tricky Way....Bruce Love
Hamilton, New Zealand

Here is my effort to improve your version of turning an index into a mask. It uses (shudder!) self-modifying code, but it is shorter and faster and I think easy to understand.

```
LOVE.VERSION
      AND #7
      EOR #7
      STA .1+1
      LDA #1
.1    BNE .1  (OFFSET FILLED IN)
      ASL
      ASL
      ASL
      ASL
      ASL
      ASL
      ASL
      RTS
```

And Still Another...David Eisler
Littleton, Colorado

With reference to "Turn Index into a Mask" (AAL Sept 84), here is another tricky alternative. It uses only the A-register, is only 16 bytes long, and takes 9 to 23 cycles.

```
EISLER.VERSION
      AND #7
      STA .1+1
      LDA #$80
.1    BNE .1  (OFFSET FILLED IN)
      LSR
      LSR
      LSR
      LSR
      LSR
      LSR
      LSR
      RTS
```

==============================================================================
DOCUMENT :AAL-8410:Articles:Out.Of.Print.txt
==============================================================================

Out of Print..............................Bob Sander-Cederlof

After printing the mini-review of Gene Zumchak's "Microprocessor Design and Troubleshooting" last month, we naturally started receiving orders for the book. I had some on order from Sams, but Lo! It is now out-of-print! I talked with someone inside Sams and they said it will probably remain out-of-print.

I talked with the author directly, and I believe that if necessary he will re-publish the book himself. It is a worthy book, and needs to be available. He wants to update some of the material, too. We'll let you know when we can get it again.

You may have noticed that "computer" books are now the "in" thing to publish. I would not be surprised if some publishers began having serious difficulties because of their eagerness to grab this market. They are publishing fluff for the neophytes, forgetting the really useful technical titles. I hope Sams does not forget how it got where it is today.

Meanwhile, as Art Carlson says, "If you see a book you need you had better get while it is still available."

On this same subject, let's see if we can put some pressure on Apple to make their reference manuals more readily available. I find that very few (hardly any) Apple dealers will stock or even special order the ProDOS, //e, and //c Reference Manuals. More than twice I have been told that (for example) the //e manual had never been published, even though I bought a copy at a store many moons ago. It seems that Apple will only sell the books in bundles of five or more of the same title, and then only to Apple dealers. Apple dealers seem to not want to order five or more of what are a relatively slow moving item. After all, they are not book stores. And consequently, Apple gets the erroneous impression that they really do not need to publish the manuals, because no one is buying them! If you know anyone in Apple, pass the word to them: WE DO WANT REFERENCE MANUALS. Maybe it does make sense not to ship a copy of every manual with every computer, but some means MUST be available for EVERY owner to buy the manuals he needs.

```
==============================================================================
DOCUMENT :AAL-8410:Articles:Putneys.Way.txt
==============================================================================
```

An Even Trickier "Index to Mask"...............Charles Putney
Dublin, Eire

I got AAL today (September 1984 issue), and pored through it as usual. The "index" article on page 18 caught my eye.  Naturally I tried to think of a smaller way of coding the routine like your "TRICKIER.WAY" of 32 bytes.  Here it is, in only 23 bytes!

```
1000 *--------------------------------
1010 *      PUTNEY'S WAY
1020 *--------------------------------
1030 PUTNEY.WAY
1040        AND #7       .....421
1050        LSR          ......42, 1 IN CARRY
1060        PHA          SAVE FOR LATER PLP
1070        LDA #1       INITIAL MASK VALUE
1080        BCC .1       NO NEED TO SHIFT 1
1090        ASL
1100 .1     PLP          GET 1.....42 AS NV.BDIZC
1110        BCC .2       NO NEED TO SHIFT 2
1120        PHP
1130        ASL
1140        ASL
1150        PLP
1160 .2     BNE .3       NO NEED TO SHIFT 4
1170        ASL
1180        ASL
1190        ASL
1200        ASL
1210 .3     RTS
```

The timing, not including a JSR to it nor the RTS at the end, varies from a best case of 21 cycles to a worst case of 39 cycles.

[One note of warning:  the PLP pulls a status of 000000xx, setting the I-status to zero.  This enables IRQ interrupts, which might be very dangerous if you have an interrupting source connected and were otherwise unprepared.]

Another Tricky Way................................Bruce Love
Hamilton, New Zealand

Here is my effort to improve your version of turning an index into a mask.  It uses (shudder!) self-modifying code, but it is shorter and faster and I think easy to understand.

```
LOVE.VERSION
       AND #7
       EOR #7
```

```
       STA .1+1
       LDA #1
.1     BNE .1  (OFFSET FILLED IN)
       ASL
       ASL
       ASL
       ASL
       ASL
       ASL
       ASL
       RTS
```

And still another................................David Eisler
Littleton, Colorado

With reference to "Turn Index into a Mask" (AAL Sept 84), here is another tricky alternative. It uses only the A-register, is only 16 bytes long, and takes 9 to 23 cycles.

```
EISLER.VERSION
       AND #7
       STA .1+1
       LDA #$80
.1     BNE .1  (OFFSET FILLED IN)
       LSR
       LSR
       LSR
       LSR
       LSR
       LSR
       LSR
       RTS
```

==============================================================
DOCUMENT :AAL-8410:Articles:V5N1.65802.txt
==============================================================

The 65802 is Here!........................Bob Sander-Cederlof

I think it was last December that I learned of the new 16-bit versions of our old friend, the 6502. You will remember my enthusiastic description in the Jan 84 issue. People at Western Design Center were optimistic about shipping chips in a month or so. Very optimistic. Way too optimistic. Nevertheless, they followed the tradition of our whole industry by continuing to stick by their commitment. Every time we called, it was always "in a month or so"!

But yesterday (Oct 12th) it arrived. Nice shiny new COD sticker on top, for $98.05, and nice new 40-legged bug inside. I plugged the 65802 into my //e, after carefully removing the 65C02 I had just put in a week before. Power on, the drive whirrs, RESET works, hurray!

So far I have spent about six hours exploring the new opcodes. I used the new but yet unreleased version 2.0 of the S-C Macro Assembler, naturally. The literature available up till now has been very sketchy on the details of some of the new opcodes and addressing modes. Anyway, no matter how well the printed word is used, the chip itself will always have the final say, the last word.

Which reminds me that I have already had to correct one mis-understanding (bug?). I was not computing the relative offsets for the 16-bit relative address mode. There are two opcodes which use this mode: BRL, Branch Relative Long; and PER, Push Effective address Relative.

BRL can branch anywhere within a 64K memory, using an offset of 16-bits. Compare this with the other relative branches, which use only an 8-bit offset and can only branch inside a 256-byte space centered around the instruction. BRL's offset ranges from -32768 to +32767.

PER pushes two bytes onto the stack. The two bytes pushed are the high byte and then the low byte of the address calculated by adding the 16-bit offset to the current PC-register. For example,

```
       0800- 62 FD FF    PER $0800
       0803-
```

pushes first $08 and then $00 onto the stack. Voila! Now we really can write position independent code! Using the 16-bit mode, I can PER the address of a data item or table onto the stack, and then PLX (Pull to X-register) that address, and access data by LDA 0,X or the like.

Another favorite pair are the two block move instructions: MVN and MVP. With these I can move any block of memory from 1 byte up to 64K bytes from anywhere to anywhere. With the 65802, anywhere is still limited to the 64K address space, but with the 65816 it can be anywhere in 16 megabytes.

To get full advantage of MVP and MVN, you need to be in the 16-bit mode. You get there in two steps: first you turn on the 65802 mode, as opposed to the 6502-emulation mode; and then you set some status bits which select 16-bit memory references and 16-bit indexing.

You turn on the 65802 mode by clearing the new E-bit in the status register. The E-bit hides behind the Carry bit, and you access it with the XCE (Exchange C and E) instruction.

```
       CLC
       XCE     turns on 65802 mode

       SEC
       XCE     turns on 6502 emulation mode
```

Then REP #$30 turns on the 16-bit mode. REP stands for Reset P-bits. Wherever there are one bits in the immediate value, the corresponding status bits will be cleared. Where there are zero bits in the immediate value, the corresponding status bits will be unaffected. The two bits cleared by REP #$30 are the M- and X-bits. If either of these, or both, are zero, the immediate mode of LDA, LDX, LDY, CMP, ADC, SBC, AND, ORA, and EOR become three byte instructions. For example,

```
       LDA ##$1234
```

loads $1234 into the extended 16-bit A-register. The long A-reg gets a new name or two. The high byte is called the B-register, the low byte is still the A-register, and the pair together are called the C-register.

Okay. Now back to the block movers. Both of the moves require some setting up first. You put the 16-bit address of the source block into the X-register, the destination address in Y, and the move count in C. For example, suppose I want to move the block $0800-$0847 up to $0912:

```
       LDX ##$0800     source
       LDY ##$0912     destination
       LDA ##$0047     # bytes - 1
       MVN 0,0         move it
```

As each byte is moved, X and Y are incremented and A is decremented. After all is complete, A will have $FFFF, X=$0848, and Y=$095A.

MVP, on the other hand, decrements the A-, X- and Y-registers for each byte moved. If the block source and destination overlap, you can use the one which moves in the order that prevents mis-copying.

Those two zeroes after the MVN instruction above are two 8-bit values. In the 65802 they don't mean anything, but in the 65816 they are the high 8-bits of the 24-bit addresses of source and destination. In the

65816, you could copy one entire 64K bank to another with just four instructions! And it only takes 3 cycles per byte moved!

The 65802 plugs directly into the 6502 socket in your Apple //e. It may or may not work in older Apples ... I haven't tried it yet. The 65816 will not plug into any current Apple II, even though it also has forty pins. The extra 8-bits of address are multiplexed on the 8 data lines, and the meaning of the other pins is somewhat changed.

Please don't get the idea that plugging in this new chip will speed up your old software. Old software will stay in the 6502 emulation mode, and will run at exactly the same pace as before. New software can be written which will take advantage of the new features, and it can be a little faster, more compact, and so on. The exciting future of the 65802 and 65816 lies not inside old Apples, but in the Apples yet to be born. I am dreaming of a 4-megahertz, 1- to 8-megabyte Apple ...

Meanwhile, here is a REAL example. Way back in the January 1981 issue of Apple Assembly Line I published a General Move Subroutine. It was set up as a control-Y command for the monitor. As an improvement over the monitor M-command, it could move blocks which overlapped either up or down in memory without repeating the leading bytes.

The following program takes advantage of the MVN and MVP commands to greatly speed up and shrink my previous effort. The old one took 149 bytes, the new one only 80. Disregarding all the setup time, which also improved, the time to move a single byte changed from a minimum of 16 cycles to a consistent 3 cycles.

Lines through 1090 describe how to set up and run the program, but don't even TRY it until you get a 65802 chip into your Apple! The new opcodes will do amazing things in an old 6502 chip, but nothing at all like intended.

Line 1100, the .OP 65816 directive, tells version 2.0 that it should allow and assemble the full 65816 instruction set.

Lines 1180-1250 are executed if you use $300G after assembling, or if you BRUN it from a type-B file.

A1, A2, and A4 are monitor variables which are setup by the control-Y command. When you type, for example, 800<900.957^Y (where by ^Y I mean control-Y), $800 is stored in A4, $900 in A1, and $957 in A2.

Lines 1270-1290 save the three registers, and these will be restored later at lines 1500-1520. Lines 1320-1340 get us unto the 16-bit mode described above. Just before returning to the monitor we will switch back to 6502 emulation mode, at lines 1480-1490.

Lines 1360-1390 calculate the "#bytes-1" to be moved, by using 16-bit subtraction. Note that the opcodes assembled are exactly the same as they would be for 8-bit operations; the cpu does 16-bit steps here because we set the 16-bit mode.

Lines 1410-1460 determine which direction the block is to be moved: up toward higher memory addresses, or down toward lower addresses. By using two separate routines we prevent garbling the move of an overlapping block.

Lines 1610-1660 move a block down. It is as easy as rolling off a log.... Just load up the registers, and do an MVN command.

Lines 1680-1760 move a block up. Here we need the addresses of the ends of the blocks, so lines 1690-1720 calculate the end address for the destination. Then we do the MVP command, and zzaappp! it's done.

=====================================================================
DOCUMENT :AAL-8410:DOS3.3:S.DP18.FUNC.1.txt
=====================================================================

```
 1000  *SAVE S.DP18 FUNC 1
 1010  *--------------------------------
 1020  AS.CHRGOT   .EQ $00B7
 1030  AS.FRMEVL   .EQ $DD7B
 1040  AS.CHKSTR   .EQ $DD7B
 1050  AS.FRESTR   .EQ $E600
 1060  AS.ILLERR   .EQ $E199
 1070  *--------------------------------
 1080  DMULT       .EQ $FFFF
 1090  DDIV        .EQ $FFFF
 1100  DADD        .EQ $FFFF
 1110  FIN         .EQ $FFFF
 1120  DP.TRUE     .EQ $FFFF
 1130  DP.ZERO     .EQ $FFFF
 1140  MOVE.DAC.TEMP3 .EQ $FFFF
 1150  MOVE.DAC.TEMP2 .EQ $FFFF
 1160  MOVE.TEMP2.DAC .EQ $FFFF
 1170  MOVE.YA.DAC.1  .EQ $FFFF
 1180  MOVE.YA.ARG.1  .EQ $FFFF
 1190  MOVE.TEMP3.ARG .EQ $FFFF
 1200  MOVE.TEMP2.ARG .EQ $FFFF
 1210  *--------------------------------
 1220  TXTPTR      .EQ $B8,B9
 1230  DEST        .EQ $60,61
 1240  *--------------------------------
 1250  TEMP2       .BS 1
 1260  TEMP3       .BS 1
 1270  P1          .BS 2
 1280  DAC.EXPONENT .BS 1
 1290  DAC.HI        .BS 10
 1300  DAC.SIGN      .BS 1
 1310  *--------------------------------
 1320  *      VAL (X$) FUNCTION
 1330  *--------------------------------
 1340  DP.VAL JSR AS.CHRGOT
 1350         JSR AS.FRMEVL     GET STRING
 1360         JSR AS.CHKSTR MAKE SURE IT IS A STRING
 1370         LDA TXTPTR   SAVE TXTPTR
 1380         PHA          ...ON STACK
 1390         LDA TXTPTR+1
 1400         PHA
 1410         JSR AS.FRESTR FREE THE STRING;GET ADR IN
 1420         STX TXTPTR   Y,X AND LEN IN A
 1430         STX DEST    SAVE BEGINNING OF STRING
 1440         STY TXTPTR+1
 1450         STY DEST+1
 1460         TAY          LENGTH TO Y
 1470         STA TEMP2    SAVE LENGTH
 1480         LDA (TXTPTR),Y
```

```
       PHA          SAVE CHAR AT END OF STRING
       LDA #0
       STA (TXTPTR),Y  PUT 0 AT END OF STRING
       JSR FIN      GET THE NUMBER
       PLA          GET CHAR
       LDY TEMP2    GET LENGTH
       STA (DEST),Y
       PLA           RESTORE TXTPTR
       STA TXTPTR+1
       PLA
       STA TXTPTR
       RTS          VAL IS IN DAC
*-------------------------------
*      INT FUNCTION
*-------------------------------
DP.INT LDA DAC.EXPONENT
       SEC
       SBC #$40     REMOVE OFFSET
       BPL .1       POSITIVE EXP
*---ALL FRACTION, MAKE = 0-------
.0     JMP DP.ZERO
*---SOME INTEGER, TRUNCATE-------
.1     BEQ .0       ...ALL FRACTION
       CMP #20      ALL INTEGER?
       BCS .4       ...YES, NONTHING TO LOP
       LSR          DIVIDE BY 2
       TAY          BYTE INDEX
       BCC .3       ...NO NYBBLE TO CLEAR
       LDA DAC.HI,Y ...CLEAR A NYBBLE
       AND #$F0
       STA DAC.HI,Y
.2     INY          ...NEXT BYTE
       CPY #10      FINISHED?
       BCS .4       ...YES
.3     LDA #0       CLEAR A BYTE
       STA DAC.HI,Y
       BEQ .2       ...ALWAYS
.4     RTS
*-------------------------------
*      ABS (DAC)
*-------------------------------
DP.ABS LDA #0       STORE 0 IN
       STA DAC.SIGN SIGN
       RTS
*-------------------------------
*      SGN (DAC)
*-------------------------------
DP.SGN LDA DAC.EXPONENT
       BEQ .1       IT IS 0, SO LEAVE IT
       LDA DAC.SIGN
       PHA          SAVE SIGN
       JSR DP.TRUE  PUT 1 IN DAC
       PLA
       STA DAC.SIGN RESTORE SIGN
```

```
2030  .1     RTS
2040  *-------------------------------
2050  *      SQR (DAC)
2060  *      #0072 IN HART, ET AL
2070  *-------------------------------
2080  ERR.SQ JMP AS.ILLERR  ILLEGAL QUANTITY
2090  DP.SQR LDA DAC.EXPONENT
2100         BEQ .3        SQR(0)=0
2110         LDA DAC.SIGN
2120         BMI ERR.SQ   MUST BE POSITIVE
2130         JSR MOVE.DAC.TEMP3 SAVE X
2140  *---REDUCE RANGE TO .1 - 1-------
2150         LDA DAC.EXPONENT
2160         PHA          SAVE EXPONENT
2170         LDA #$40     CHANGE RANGE TO .1 THRU .9999...9
2180         STA DAC.EXPONENT
2190  *---REDUCE RANGE TO .25 - 1------
2200         LDA DAC.HI
2210         CMP #$25     LESS THAN .25?
2220         PHP          SAVE ANSWER
2230         BCS .4       ...NO
2240         LDA #CON.FOUR
2250         LDY /CON.FOUR
2260         JSR MOVE.YA.ARG.1
2270         JSR DMULT
2280  *---CALC FIRST APPROX.-----------
2290  .4     JSR MOVE.DAC.TEMP2
2300         LDA #P.SQR   GET FIRST APPROXIMATION
2310         LDY /P.SQR   FROM AX^3+BX^2+CX+D
2320         LDX #P.SQR.N
2330         JSR POLY.N
2340  *---ADJUST APPROX FOR FOLDING----
2350         PLP          WAS X<.25?
2360         BCS .5       ...NO
2370         LDA #CON.HALF
2380         LDY /CON.HALF
2390         JSR MOVE.YA.ARG.1
2400         JSR DMULT
2410  *---COMPUTE SQR EXPONENT---------
2420  .5     PLA          GET EXPONENT FROM BEGINNING
2430         SEC
2440         SBC #$40     REMOVE OFFSET
2450         ROR          DIVIDE BY TWO (KEEP SIGN)
2460         EOR #$80
2470         BCC .1       DON'T MULT BY SQR(10)
2480  *---ADJUST APPROX FOR ODD EXP----
2490         PHA          SAVE EXPONENT/2
2500         LDA #CON.SQR10
2510         LDY /CON.SQR10
2520         JSR MOVE.YA.ARG.1
2530         JSR DMULT
2540         PLA
2550  *---INSTALL NEW EXPONENT---------
2560  .1     CLC
```

```
2570        ADC DAC.EXPONENT
2580        STA DAC.EXPONENT
2590 *---THREE NEWTON ITERATIONS------
2600        LDA #3
2610        STA TEMP3
2620 .2     JSR MOVE.DAC.TEMP2     TEMP2 = Y
2630        JSR MOVE.TEMP3.ARG     GET X
2640        JSR DDIV               X/Y
2650        JSR MOVE.TEMP2.ARG
2660        JSR DADD               X/Y+Y
2670        LDA #CON.HALF
2680        LDY /CON.HALF
2690        JSR MOVE.YA.ARG.1
2700        JSR DMULT              (X/Y+Y)/2
2710        DEC TEMP3              ANY MORE?
2720        BNE .2                 ...YES
2730 .3     RTS                    ...DONE
2740 *--------------------------------
2750 P.SQR.N  .EQ 3
2760 P.SQR      .HS 4028736982400000000000
2770            .HS C0825888891000000000000
2780            .HS 4113225638600000000000
2790            .HS 4021701867200000000000
2800 CON.SQR10  .HS 4131622776601683793320
2810 CON.HALF   .HS 4050000000000000000000
2820 CON.FOUR   .HS 4140000000000000000000
2830 *--------------------------------
2840 *      POLYNOMIAL EVALUATOR ROUTINES
2850 *      (Y,A) = ADDRESS OF COEFFICIENT TABLE
2860 *      ARRANGED HIGHEST POWER TO LOWEST
2870 *      CONSTANTS DO USE GUARD BYTE (11 TOTAL)
2880 *--------------------------------
2890 *   DO A POLYNOMIAL WITH 1ST CONSTANT 1
2900 *      (TEMP2) IS X-VALUE
2910 *      (X-REG) IS N
2920 *           WHERE N = POWER OF X
2930 *           FOR EXAMPLE, IF N=2 : X^2+AX+B
2940 *                           N=4 : X^4+AX^3+BX^2+CX+D
2950 *--------------------------------
2960 POLY.1
2970        STA P1
2980        STY P1+1
2990        STX TEMP3
3000        JSR MOVE.TEMP2.DAC
3010 POLY   LDA P1
3020        LDY P1+1
3030        JSR MOVE.YA.ARG.1
3040        JSR DADD
3050        DEC TEMP3    FINISHED YET?
3060        BNE POLY2    ...NO
3070        RTS          ...YES
3080 *--------------------------------
3090 *      DO A POLYNOMIAL WITH 1ST CONSTANT <> 1
3100 *      (TEMP2) IS X-VALUE
```

```
3110  *      (X-REG) IS N
3120  *           WHERE N = POWER OF X
3130  *           FOR EXAMPLE, IF N=2 : AX^2+BX+C
3140  *                           N=3 : AX^3+BX^2+CX+D
3150  *-------------------------------
3160  POLY.N
3170         STA P1
3180         STY P1+1
3190         STX TEMP3
3200         JSR MOVE.YA.DAC.1
3210  POLY2  JSR MOVE.TEMP2.ARG
3220         JSR DMULT
3230         CLC
3240         LDA P1
3250         ADC #11      NUMBER OF BYTES
3260         STA P1
3270         BCC POLY
3280         INC P1+1
3290         BNE POLY     ...ALWAYS
3300  *-------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8410:DOS3.3:S.GENERAL.MOVER.txt
=============================================================================

 1000  *SAVE S.GENERAL MOVER
 1010  *--------------------------------
 1020  *   BRUN the program to set it up as
 1030  *      a control-Y monitor command.
 1040  *--------------------------------
 1050  *   Use like the Monitor M-command:
 1060  *      A1 -- Source start address
 1070  *      A2 -- Source end address
 1080  *      A4 -- Destination start address
 1090  *--------------------------------
 1100         .OP 65816
 1110         .OR $300
 1120  *--------------------------------
 1130  A1     .EQ $3C,3D
 1140  A2     .EQ $3E,3F
 1150  A4     .EQ $42,43
 1160  BLKSIZ .EQ $00,01
 1170  *--------------------------------
 1180  CONTROL.Y.SETUP
 1190         LDA #$4C
 1200         STA $3F8
 1210         LDA #GENERAL.MOVER
 1220         STA $3F9
 1230         LDA /GENERAL.MOVER
 1240         STA $3FA
 1250         RTS
 1260  *--------------------------------
 1270  GENERAL.MOVER
 1280         PHA
 1290         PHY
 1300         PHX
 1310  *--------------------------------
 1320         CLC          65816 MODE
 1330         XCE
 1340         REP #$30     16-BIT MODE
 1350  *--------------------------------
 1360         SEC          Compute block length - 1
 1370         LDA A2
 1380         SBC A1
 1390         STA BLKSIZ
 1400  *--------------------------------
 1410         LDA A1
 1420         CMP A4       Determine direction of move
 1430         BCC .1       ...UP
 1440         JSR MOVE.DOWN
 1450         BRA .2       ...ALWAYS
 1460  .1     JSR MOVE.UP
 1470  *--------------------------------
 1480  .2     SEC          RETURN TO 6502 MODE
```

```
1490        XCE
1500        PLX
1510        PLY
1520        PLA
1530        RTS
1600 *---------------------------------
1610 MOVE.DOWN
1620        LDX A1        Source start address
1630        LDY A4        Destination start address
1640        LDA BLKSIZ    # Bytes - 1
1650        MVN 0,0
1660        RTS
1670 *---------------------------------
1680 MOVE.UP
1690        CLC
1700        LDA A4
1710        ADC BLKSIZ
1720        TAY           Destination end address
1730        LDX A2        Source end address
1740        LDA BLKSIZ    # Bytes - 1
1750        MVP 0,0
1760        RTS
```

=============================================================================
DOCUMENT :AAL-8410:DOS3.3:S.PUTNEYS.WAY.txt
=============================================================================

```
 .LIF
*SAVE S.PUTNEY'S WAY
*---------------------------------
*      PUTNEY'S WAY
*---------------------------------
PUTNEY.WAY
       AND #7       .....421
       LSR          ......42, 1 IN CARRY
       PHA          SAVE FOR LATER PLP
       LDA #1       INITIAL MASK VALUE
       BCC .1       NO NEED TO SHIFT 1
       ASL
.1     PLP          GET 1.....42 AS NV.BDIZC
       BCC .2       NO NEED TO SHIFT 2
       PHP
       ASL
       ASL
       PLP
.2     BNE .3       NO NEED TO SHIFT 4
       ASL
       ASL
       ASL
       ASL
.3     RTS
*---------------------------------
*   BRUCE LOVE'S METHODS
*---------------------------------
LOVE.1 AND #7
       LSR
       PHP
       LSR
       EOR #1
       BNE .1
       LDA #$10
.1     BCC .2
       ASL
       ASL
.2     PLP
       BCC .3
       ASL
.3     RTS
*---------------------------------
LOVE.2 AND #7
       EOR #7
       STA .1+1
       LDA #1
.1     BNE .2
.2     ASL
       ASL
```

```
1490        ASL
1500        ASL
1510        ASL
1520        ASL
1530        ASL
1540        RTS
1550  *--------------------------------
1560  *   MASK --> INDEX
1570  *      19 BYTES (NOT COUNTING RTS)
1580  *------------------80-40-20-10-08-04-02-01
1590  RBSC.1 LSR          40 20 10 08 04 02 01 00
1600         CMP #4
1610         BCC .2
1620         BEQ .1
1630         LSR          20 10 08 04
1640         LSR          10 08 04 02
1650         LSR          08 04 02 01
1660         ADC #8       10 0C 0A 09
1670         LSR          08 06 05 04
1680         CMP #8
1690         BCC .2
1700  .1     SBC #1       07 .. .. .. 03 .. .. ..
1710  .2     RTS          07 06 05 04 03 02 01 00
1720  *       CYCLES:     24 23 23 23 10 07 07 07 (WITHOUT RTS)
1730  *                   AVERAGE = 15.5 CYCLES
1740  *--------------------------------
1750  *   MASK --> INDEX VIA X-LOOP
1760  *--------------------------------
1770  RBSC.2 LDX #8
1780  .1     DEX
1790         ASL
1800         BCC .1
1810         TXA
1820         RTS
1830  *--------------------------------
1840  TESTMX LDA #1
1850  .1     PHA
1860         JSR RBSC.1
1870         JSR $FDDA
1880         LDA #"-"
1890         JSR $FDED
1900         PLA
1910         PHA
1920         JSR $FDDA
1930         JSR $FD8E
1940         PLA
1950         ASL
1960         BCC .1
1970         RTS
```

================================================================
DOCUMENT :AAL-8411:Articles:Alliance.CPUs.txt
================================================================

New Source for 65802's

I talked to Constantine Geromnimon at Alliance Computers this morning. His company has ordered hundreds of 65802's, and offers them to you at $49.95 each. They expect their next shipment to come in around the middle of January, so now is the time to order. Call them at (718) 672-0684, or write to P. O. Box 408, Corona, NY 11368.

```
=============================================================================
DOCUMENT :AAL-8411:Articles:Annc.2.0.txt
=============================================================================
```

S-C Macro Assembler Version 2.0....................Bill Morgan

We are now accepting orders for the upgrade to S-C Macro Assembler Version 2.0.  Here is a summary of the new features:

o The big news, of course, is the ability to assemble 65C02, 65802, and 65816 opcodes.  The new .OP directive switches between the 6502, Sweet-16, 65C02, and 65816 opcode sets.

o All screen output now passes through one driver routine, which will be much easier to modify for other displays.  Drivers are included for 40-column, //e and //c 80-column, and STB-80.

o Typing a Control-C at the command prompt (:) emits CATALOG, leaving the cursor at the end of the line, to add slot and drive specifiers if needed.

o There is a sort of Auto-SAVE function.  Once you have created a comment line near the beginning of your source file containing the phrase "SAVE filename", typing ESC-S will emit that phrase and position the cursor at the end, so you can add a suffix or just press RETURN.

o The COPY command asks "DELETE ORIGINAL?"  If you type "Y", the effect will be that of a MOVE command.

o The tape LOAD and SAVE commands have been removed, to make room for new features.

o All operand expressions are calculated to 32 bits and .DA data values may be larger, to allow for the 65816's extended addressing capabilities.

o You can force Zero Page or Absolute addressing modes by prefixing the operand with < or >.

o Operand expressions may include bitwise logical operations.  &, ! (or |), and ^ are AND, OR, and EOR.

o Control-S functions as a case lock key, toggling upper/lower case entry.

o The .BS directive allows you to specify the value of the fill byte generated.  This directive now creates fill bytes in assemblies into memory, rather than to disk only.

o The assembler tests for the "/" command character, to simplify use of the Laumer Research Full Screen Editor.

o All object code bytes are vectored through a standard location, so you can intercept the assembler's output for special purposes.

Registered owners of S-C Macro Assembler will be able to purchase the upgrade to Version 2.0 for only $20.00.  Just send us a check or charge card number, and you will be among the first to have the new version.

==============================================================================
DOCUMENT :AAL-8411:Articles:Disasm.Patches.txt
==============================================================================

Generating Cross Reference Text File with DISASM...Bob Kovacs

I received a phone call from Don Lancaster the other day. He had been using DISASM to probe the mysteries of AppleWriter, and was now preparing to document his findings. Although he liked the way DISASM generated a triple cross reference table, he preferred to have it in a form that could be used by his word processor (that is, on a text file). The cross reference table generated by DISASM is normally output to either the screen or a printer, so Don's only alternative was to manually type it into his word processor. There were hundreds of labels....

It turned out that a simple patch to DISASM will do the trick. All that is necessary is to change the JSR PASS2 which normally generates the source code listing to JSR XREF.

The following patch outputs the cross reference table to your file after responding "Y" to the prompt "GENERATE TEXT FILE?":

```
       $09A1:20 F1 0A
```

Back in the April issue of AAL, I described a method of using EXEC files with DISASM. A patch was required to the "YES/NO" routine to input the response via KEYIN rather than directly from the keyboard. Although the patch I gave in April works, KEYIN uses the Y-register as an index to the screen. My patch did not always wind up in the right place. So I have expanded the patch as follows:

```
       $0C57:EA A4 24 20 18 FD 09 80
```

I hope that this has not caused any inconvenience.

```
===============================================================================
DOCUMENT :AAL-8411:Articles:DP18.Func.2.txt
===============================================================================
```

18-Digit Arithmetic, Part 7................Bob Sander-Cederlof

Last month we began the implementation of math functions, so it seems appropriate to continue in the same direction. This month we will reveal the LOG and EXP functions.

As always, I turned to "Computer Approximations" for some good algorithms. I mentioned this book last month, and several of you have tried to find copies.

Thanks to Trey Johnson, of Monolith Inc. in San Antonio, for the following information: John Wiley & Sons stopped publishing the book "Computer Approximations" in 1977. They sold the rights to Krieger Publishing Co., and it is now being published under the same title. Trey was quoted a price of $22.50 + shipping. Krieger's address is P. O. Box 9542, Melbourne, FL 32901; phone is (305) 724-9542.

"Computer Approximations" is the only book I have found which lists all the actual coefficients needed to produce good approximations for the whole variety of standard functions. Pages 189-339 are packed solid with nothing by numbers. For example, there are ten pages of numbers for the EXP function alone, providing over 100 different approximation formulas for the EXP function. The chapter covering EXP describes the math behind the approximations. You pick an algorithm according to the precision you need, the number base you are using (2, 10, or whatever), the tradeoff between speed and size, and the range of arguments you will be using. Each algorithm in the book has a number, and I indicate that number in the comments to the programs which follow.

Almost all of the approximations involve these steps:

```
     SIFT:  Check the argument for legal range and
            easy arguments.
     FOLD:  Reduce the range of the argument.
     POLY:  Use a polynomial or a ratio of polynomials
           to approximate the function in the reduced
           range.
   UNFOLD:  Expand the result by the reverse of the
            processes used to reduce the range.
```

When we first learned about logarithms in high school, we used tables in books. One set of tables converted normal numbers to logs, and the other converted logs back to normal numbers. The LOG function takes the place of the first set of tables, and the EXP function replaces the second. By the way, those high school logarithms were base 10 logs. The log of a number is the power to which you would have to raise 10 to equal the number. For example, the log base 10 of 1000 is 3; of the square root of 10 is .5.

Scientists prefer base "e" logs. "e" is an irrational number (as is pi) approximately equal to 2.71828182845904523536. Did the original scientists have 2.718281828... fingers? Maybe, if they had to chop firewood (logs?)! Anyway, EXP and LOG in Applesoft work with base e. LOG tells you to what power you would raise e to equal the argument, and EXP raises e to the power of the argument.

One great application of LOG and EXP is to raise any number to any power. Applesoft (as well as DP18) has an exponentiation operator "^" for this purpose, but the code inside does it by calling on EXP and LOG. Here are some mathematical symbols to indicate how it is done:

```
       let          z = x^y
      then      log z = log (x^y)
                log z = y log x
         exp (log z) = exp (y log x)
                  x^y = exp (y log x)
```

Here is the code for the exponentiation operator in DP18:

```
*--------------------------------------
*   EXPONENTIATION:  X ^ Y
*      (DAC) = Y
*      (ARG) = X
*--------------------------------------
DP.POWER
       JSR MOVE.DAC.TEMP3   SAVE DAC (POWER) IN TEMP3
       JSR SWAP.ARG.DAC
       JSR DP.LOG10         GET LOG X
       JSR MOVE.TEMP3.ARG   GET Y IN ARG
       JSR DMULT            Y LOG X
       JMP DP.EXP10         X ^ Y
```

Notice I used base 10 log and exp? That is because DP18 is basically decimal. In a binary floating point scheme such as is internal to Applesoft, base 2 log and exp would probably be used. After all, floating point notation is a kind of half-log half-normal notation.

Which leads to the topic of converting from one logarithmic base to another. If my internal subroutines work in base 10, how do I get LOG and EXP to base e? Some more math is due:

```
       suppose         e^x = 10^y
       then    log10 (e^x) = log10 (10^y)
                x log10(e) = y log10(10)
                x log10(e) = y
```

Log10(e) is a constant, approximately 0.43429448190325182765. So if I want to know what EXP(3) is, I can first get 3*log10(e) = 1.302..., and 10^1.302... = 20.0855...

EXP Function

Lines 1640-1660 of the program check for a zero argument, which is an easy case:  e^0 = 1.  Lines 1670-1700 multiply the argument by log10(e), so that EXP10 can be used.

Lines 1730-1740 again sift out the easy case of 10^0, in case DP.EXP10 was called directly.

Lines 1750-1790 begin the folding process.  We can cut the range in half by folding all negative arguments on top to the positive range: EXP(-x) = 1/EXP(x).

Lines 1810,1820 further sift, by eliminating arguments larger than 99. If the exponent of the argument is $43 or more, then the argument is 100 or more.  Arguments that large are too large.  (Indeed, any argument above 63 is too large.)  The Applesoft ROM routine for OVERFLOW ERROR will let you know you tried it.

The arguments we have left will be in the range 0 < x < 100.  We can further subdivide the range by separating the integer and fractional parts of the argument.  Remember that 10^(x+y) = (10^x)*(10^y)?  For illustration, suppose the argument is 3.75.  Then 10^3.75 = 10^3 * 10^.75 = 5623.4132....  Lines 1830-2100 perform the separation.  The variable INTPWR will get the integer part, which may range from 0 to 99.  The corresponding digits are zeroed in DAC, and the resulting fraction is re-normalized.  If the fractional part is zero, then the log of the fractional part is 1; lines 2080-2100 sift out this special case.  This section could be accomplished by using previously covered subroutines, such as DP.INT to get the integer part, and DSUB to get the fractional part.  However, that would take considerably longer for only a slight savings in space.

The active part of the argument has now been reduced to the range 0<x<1.  The next adjustment will cut that in half.  If the argument x<.5, this adjustment will be skipped.  Lines 2120-2160 perform the test, and line 2170 saves the result of the test on the stack.  We need the result later when we are unfolding.  If x >= .5, then lines 2190-2210 subtract .5 from it.  If x = .5, then the result after subtraction will be zero.  In this case, the correct answer is a known constant, the square root of 10.  Lines 2230-2270 load up that value and skip over the POLY part on down to the UNFOLDing.  If not exactly .5, we now have a folded argument in the range 0<x<.5, with a flag on the stack indicating whether or not we subtracted .5 to get there. Later, if we DID subtract .5, we will multiply the result of POLY by the square root of 10 to unfold the answer.

We could have arbitrarily subtracted .5, changing the range from 0<x<1 to -.5<x<.5, with the same result.  This would have saved the trouble of determining which side of .5 we were on, and of later deciding whether or not to multiply by SQR(10).  However, it would also take longer for those cases already under .5, so I decided against it.

The POLY part is lines 2280-2520.  This is a ratio of two polynomials, both 8th degree.  However, because of derivational and computational reasons, it is actually written and calculated in a different form:

```
                 Q(x^2) + xP(x^2)
       POLY(x) = ----------------
                 Q(x^2) - xP(x^2)
```

Lines 2290-2320 save x and compute x^2. Lines 2330-2380 call on POLY.N (covered last month) to compute the P polynomial, and then multiply the result by x. The constants are given in lines 1440-1490. So that you see the form, I will give it here with the coefficients rounded off:

```
       xP = 31x^7 + 4562x^5 + 134331x^3 + 760254x
```

Lines 2400-2430 compute the Q-polynomial, by calling POLY.1 (also covered last month). POLY.1 is used when the coefficient of the highest degreed term is 1. We get, approximately,

```
       Q = x^8 + 477x^6 + 29732x^4 + 408437x^2 + 660349
```

Lines 2440-2520 form the numerator and denominator and divide, giving us a very nice approximation to the function for the folded argument.

Lines 2530-2590 begin the unfolding process, by multiplying by SQR(10) if we previously folded .5<x<1 down to 0<x<.5.

Lines 2600-2660 take care of the integral portion of the original argument, by adding it to the EXPONENT of the result so far. This is equivalent to multiplying by the integral power of ten, but much faster. Isn't base ten nice?

The final adjustment is to take the reciprocal if the original argument was negative, done in lines 2670-2730.

LOG Function

The LOG function is the inverse of the EXP function. Now if we could just run the 6502 backwards....

Log base e is related to log base 10 the same way the exp functions were:

```
       loge x = loge(10) * log10 (x)
```

Lines 2990-3040 call on the LOG10 subroutine and then multiply the result by the log base e of 10.

The LOG10 routine begins by sifting out the objectionable argument values, at lines 3100-3130. The argument MUST be positive, and MUST NOT be zero. Negative or zero arguments send you to Applesoft's ILLEGAL QUANTITY ERROR.

Lines 3140-3170 separate the exponent from the mantissa of the argument. The exponent represents the power of 10 multiplier, so as an integer it can just be added to the logarithm of the mantissa viewed as a fraction. The exponent is saved in INTPWR, to be processed later. Stuffing $40 in its place in DAC makes the range now .1<=x<1.

Lines 3180-3210 multiply the fraction by SQR(10), which changes the range to

```
       1
    -------  <= x < SQR(10)
    SQR(10)
```

This can be compensated for later by subtracting .5 from the logarithm of the folded argument.

Lines 3220 further thrash the argument by forming an intermediate argument z = (x-1)/(x+1). This value z will be in the range -.52 < z < +.52, which is a nice symmetrical value to run through a ratio of polynomials. I get lost in the math that motivates this step.

The POLY part is again a ratio of two polynomials. Lines 3330-3440 calculate the numerator, which is approximately

```
     -15z^11 + 301z^9 - 1726z^7 + 4060z^5 - 4192z^3 + 1576z
```

The denominator, formed in lines 3450-3500, is approximately

```
     z^12-68z^10+764z^8-3200z^6+6122z^4-5432z^2+1815
```

Dividing at line 3510 gives the logarithm of the value x. To unfold, we need to subtract .5, handled by lines 3860-3920. We also need to add as an integer the power of ten we saved in INTPWR. The latter is trickier, because we must convert a biased binary integer to a signed decimal floating point value.

Lines 3530-3600 un-bias INTPWR. If the exponent happens to be exactly $40, which in un-biased terms is 0, the rest of this step can be skipped (because the log of 10^0 is zero, adding nothing). If not, it is time to build a DP18 value in ARG. Line 3570 saves the sign in ARG.SIGN.

Lines 3610-3620 pre-clear ARG.HI, which is where we will be putting the one or two digits of INTPWR. Line 3630 assumes it will be a one-digit value, and lines 3640-3650 test that assumption. If it is one digit, lines 3730-3780 will shift the digit to the left nybble and store it in ARG.HI. If two digits, lines 3660 will divide by ten to get the high digit as quotient and low digit as remainder. Then lines 3730-3780 will merge the two digits into ARG.HI.

Lines 3790-3840 complete the construction of ARG by storing the exponent and clearing the remaining mantissa bytes. Line 3850 adds the value to the results of the POLY step, lines 3870-3920 subtract .5, and the answer is ready.

=============================================================================
DOCUMENT :AAL-8411:Articles:DP18.New.SQRT.txt
=============================================================================

New DP18 Square Root Subroutine............Bob Sander-Cederlof

Even after bending over backwards to be certain I had the best possible SQR implementation in the October AAL, I still found some ways to improve it. Last night I found some more information in a book called "Software Manual for the Elementary Functions", by William Cody and William Waite, Prentice-Hall, 1980.

They pointed out that in general an extra Newton iteration took less time than a complex method of getting an initial approximation which would be accurate enough to avoid one iteration. In other words, using a cubic polynomial like I did in October is just not worth it. Not worth the time, and not worth the space.

They further pointed out that it is best to compute the last Newton iteration in a slightly different fashion, to avoid shifting out the last significant digit. The normal iteration computes (x/y + y)*.5. Re-arrangement to y+(x/y-y)*.5 is better. Since it takes an extra step, it should only be used the last time.

To see the difference, consider the example below. I have used a precision of just 3 digits (instead of 18 or 20)to simplify the illustration:

```
     let x=.253, and y=.5
     then x/y=.506

     x/y+y=1.00 (truncating to 3 places)
     (x/y+y)*.5 = .500, which is wrong

     x/y-y=.006
     (x/y-y)*.5=.003
     y+(x/y-y)*.5 = .503, which is correct.
```

My new SQR version uses a much faster method for getting the first approximation. The first two digits of the argument (in DAC.HI) must be in the range from 10 to 99. I convert them to an index between $02 and $13 by shifting the first digit over three, and adding one if the second digit is 5 or more. In other words, 10-14 become $02, $15-19 become $03, on up to $95-99 becoming $13. Then I use that value as an index into a table which gives a good approximation to the first two digits of the square root. For example, any number between .10 and .19999...9 will get a first approximation of .35. I store those two digits into DAC.HI, letting the remaining digits stay as they were. This method gives a first approximation which in the worst case still has at least the first digit correct.

It turns out the worst case is for numbers with odd exponents and the mantissa=1, such as 1 (which is .1*10^1), 100 (which is .1*10^3), and so on.  Even in this worst case, four iterations give 20 digits of precision.

The end result of these changes is a faster and shorter program which is more accurate.  Here is the new listing:

================================================================
DOCUMENT :AAL-8411:Articles:Front.Page.txt
================================================================

Volume 5 -- Issue 2 November, 1984

In This Issue...



Apple II Troubleshooting Guide

We have just received a new book from Howard Sams: Apple II+/IIe Troubleshooting & Repair Guide, by Robert C. Brenner. At a glance, it looks like quite a good introduction to the Apple hardware and its potential problems. The first chapter is Basic Troubleshooting, followed by three chapters on Description, Operations, and Specific Troubleshooting for the II Plus, three more similar chapters on the //e, and two chapters on Preventive Maintenance and Advanced Troubleshooting. Here's a quote from the Introduction:

This book is a detailed troubleshooting and repair document. It is not a treatise on basic computer theory or a discussion of chip operation, registers, busses, and logic gates. It is an all "meat and potatoes" manual to enable the computer user to repair his or her own machine in those 95 percent of circumstances where knowledge and a good reference are enough to find and repair a failure.

List price of the Troubleshooting & Repair Guide is $19.95. Our price will be $18 + shipping.

Apple //e Reference Manual Source

We have located a mail- or phone-order source for the Apple manuals! A reader in New York City phoned to let us know that the McGraw-Hill Bookstore there carries the Apple publications. Apparently the bookstore is also a computer store and an Apple Dealer. The address is McGraw-Hill Bookstore, 1221 Sixth Ave., New York, NY 10020. The phone number is (212) 512-4100.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3

for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8411:Articles:Macro.Examples.txt
==============================================================================

Macro Information by Example...................Sandy Greenfarb

The following are three examples of macro use which I have found interesting and informative.

The first example, TEST, shows that you can use parameters in places other than the operand field. In this case, one of the parameters becomes part of an opcode name.

SETD shows how a macro can make more efficient code. If both bytes are the same, there is no need to have two LDA instructions.

MOVD copies two bytes from one variable to another. If you use MOVD to move two bytes one byte higher in RAM, MOVD will reverse the order the bytes are moved so that the data are not clobbered.

===============================================================================
DOCUMENT :AAL-8411:Articles:Mask2Index.txt
===============================================================================

Turning Bit-Masks into Indices.............Bob Sander-Cederlof

A few months ago I presented several ways to turn an index (0-7) into a bit mask (01, 02, 04,...,80). We got a lot of feedback, including some faster and better programs. Bruce Love suggested the possibility of the reverse transformation.

According to Bruce, who is a high school teacher in New Zealand, the method which uses the fewest bytes is the one I show in lines 1390-1450. In order to be fair in comparing different algorithms, I am going to count the RTS opcodes both for bytes and for cycles. With this in mind, Bruce's routine takes 8 bytes and from 16 to 65 cycles. This is certainly the smallest way, and it really is pretty fast.

Bruce mentioned that he had written several other programs to solve the same problem: one used the X-register, took 26 bytes with an average of 33.5 cycles; another without useing X or Y took 28 bytes and an average of 39 cycles. Unfortunately, he did not include a copy of either of these.

I worked out four more methods, shown in the listing after Bruce's. I wrote a test driver which is in lines 1000-1310. The test driver calls each routine, printing the results of each, for all possible values of the bit-mask.

The following table summarizes the data for the five algorithms:

| | | # of cycles | | |
|---|---|---|---|---|
| | bytes | min | max | ave |
| SMALLEST.WAY | 8 | 16 | 65 | 40.5 |
| WAY.WITH.X | 26 | 25 | 42 | 33.5 |
| WAY.WITHOUT.X | 23 | 14 | 30 | 22 |
| ANOTHER.WAY.W... | 32 | 14 | 24 | 18.375 |
| STRAIGHT.TEST... | 33 | 14 | 27 | 18.5 |

If the SMALLEST.WAY is not fast enough, I would probably go with the one named WAY.WITHOUT.X. It is almost as fast as the fastest, and is the shortest of the longer routines. Of course, some of you may come up with better and faster ones....

```
=================================================================
DOCUMENT :AAL-8411:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 1.0.......................................$80
S-C Macro Assembler Version 1.1....................................$92.50
Version 1.1 Update.................................................$12.50
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM....................................additional $30

S-C Word Processor (with complete source code)........................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39

Quick-Trace (Anthro-Digital)...........CLOSEOUT SPECIAL!..(reg. $50)  $45/// $35

Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                          or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers...........................  6 cents each
QuikLoader EPROM System (SCRG)..............................($179) $170
D MAnual Controller (SCRG)...................................($90)  $85
Switch-a-Slot (SCRG)........................................($190) $175
Extend-a-Slot (SCRG).........................................($35)  $32

Books, Books, Books..........................compare our discount prices!

    "Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95)  $18
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster...........($15.95)  $15
```

```
               Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

=============================================================================
DOCUMENT :AAL-8411:Articles:New.Dump.Rtn.txt
=============================================================================

Improvements to 80-column Monitor Dump.........Jan Eugenides

I found a little bug in the 80-column ASCII monitor dump, as presented in Sept 1983 AAL (page 27,28). It worked great in the 80-column mode, but if I happened to be in 40-column mode when I used the monitor dump command something strange happens.

Some time ago I incorporated the dump and Steve Knouse's monitor patches into an EPROM and installed it in my system. Everything seemed to be working fine, until one day.... I was working on a short Applesoft program, and I went into the monitor in 40-column mode to check a few program bytes. When I returned to Applesoft and listed the program, the first line had been changed. Huh?

I eventually figured out that the problem had to do with the tab to column 60. In 40-column mode this will be 20 characters beyond the bottom of the screen, which is $80C.

The solution was to just print a few spaces rather than attempting to tab. This approach makes for more compatibility among various 80-column devices, too.

While I was at it, I even squeezed a byte out of the code.

[And I squeezed some more, saving a total of 11 bytes. Bob S-C]

Here is the modified routine:

<<<<code here>>>>

Note the directions for installing the routine in a RAM card copy of the monitor, in lines 1020-1060. "$C083 C083 FCC9<CC9.CFFM" write enables the RAM area and copies the dump code over the top of cassette I/O stuff. "$FDBE:C9 FC N FDA6:F N FDB0:F" patches the monitor dump command code to call the new patch, and also patches to print 16 bytes per screen line.

If you want to use this routine in 40-column mode only, change line 1240 from "AND #$0F" to "AND #$07", line 1310 from "CPX #$10" to "CPX #$08", and leave out the patches at FDA6 and FDB0 in the previous paragraph.

```
================================================================
DOCUMENT :AAL-8411:Articles:News.65816.txt
================================================================
```

65816 News..........................................Bill Morgan

Did you see the Infoworld article a few weeks ago (November 5 issue) about the 65816? That story mentioned a plug-in board for the Apple II containing a 65816 processor and extra RAM. Well, I spoke today with Larry Hittel of Com Log, producers of that board, and it does sound very interesting.

Com Log intended their board, the Apple16, to be a developers' tool, rather than a consumer item, or an Apple hot-rod device. They were therefore a little surprised and overwhelmed by the response to the Infoworld story: When I talked to Larry they had exactly one board in stock, and it was waiting for purchase order paperwork from Apple Computer. They are a month or two away from full production quantities.

The Apple16 board uses DMA (Direct Memory Access) to take control of the Apple, shutting down the 6502 and taking over the address bus. They have found that the DMA does not function properly in Apples earlier than Revision 4, due to problems with the bus driver chips on the motherboard.

The 65816 chips are designed to operate at 8 MHz and are currently testing out at 2-4 MHz, but, in order to maintain compatibility with the Apple, the Com Log processor is clocked at 1 MHz.

To the '816, the 64K of Apple memory, both RAM and ROM, is bank 0. Bank 1 echoes the Apple from 0-DFFF, but contains space for new EPROM at E000-FFFF. Banks 2 and 3 are reserved for more new EPROM. Banks 4-7 are the on-board RAM, consisting of one set of either 64K or 256K chips. Banks 8-255 are available on an expansion connector, intended for a future separate memory board. There is abort logic to provide an interrupt on access to non-existent memory.

Com Log is selling the boards now with no EPROMs. They are working on an operating system and an Applesoft interpreter, but those are still some time away. No price has been set for the firmware yet.

The current price of the Apple16 board is $395 with no RAM, $450 with 64K, and $795 with 256K. They are not expecting to have them available in production quantities until January or later, by which time the prices might change. Contact Com Log Corporation at 11056 N. 23rd Dr., Suite 104, Phoenix, AZ 85029. Phone (602) 248-0769.

That Infoworld story quoted an Apple spokesman as saying that the 65816 was to be used in an earlier project that had been shelved. That project is being dusted off and revived, now that the 65816 chips are really coming through. We've been hearing of it as the Apple //x. According to an article in the November 19 issue of Infoworld about an interview with Woz, the //x is still not a fixed design and will not be ready for market until 1986.  There's always something new to look forward to!

=============================================================================
DOCUMENT :AAL-8411:Articles:Quick.DecHex.txt
=============================================================================

Convert Two Decimal Digits to Binary.......Bob Sander-Cederlof

I have recently been running into more and more uses for the decimal mode in the 6502. In the decimal mode, each byte contains a value from 0 to 99, with the ten's digit in the left nybble and the units digit in the right nybble.

The 6502 has built-in capability to add and subtract values in this format, with automatic carry when a nybble exceeds 9. If you have been following my series on 18-digit arithmetic, you have seen a lot of examples of its use.

A frequent problem that arises is conversion between the decimal form and the binary form of a number. I suppose I have written ten million different programs to do this kind of conversion, on at least a thousand different kinds of computers! (Ever notice that my exaggerations are always in decimal?)

For a small (byte-size) example, suppose a byte contains two decimal digits ($00-$99) and you want to convert it to binary ($00-$63). The first step is to separate the two digits into two different variables. The the ten's digit should be multiplied by ten, and the unit's digit added.

Lines 1390-1510 in the listing perform these steps, but there are a few tricks. Lines 1410-1420 strip out the unit's digit and save it in LOW, and lines 1440-1450 save the high digit in HIGH. Notice that I did not shift the high digit down, so it is really the ten's digit times 16 (call it "tens*16").

Lines 1460-1500 multiply the tens*16 by 10/16. Then line 1500 adds the unit's digit.

The program in lines 1010-1190 is a test driver, which calls the DEC.HEX.2 routine 100 times with successive values in the A-register between $00 and $99. DEC.HEX.2 returns with the converted value ($00-$63 in the A-register, and the test driver prints out the value. If everything is okay, the hexadecimal numbers from $00 through $63 will be displayed.

DEC.HEX.2 as written takes 18 bytes plus two variables in page zero. If the variables are not in page zero, the program will take an additional four bytes.

A faster program which takes only a few more bytes, and does not use any variables in RAM other than the stack, is shown in lines 1200-1340. Lines 1220-1260 convert the ten's digit into an index 0-9 in the X-register. Line 1270 retrieves the original number from the stack. Lines 1290-1300 add a value from the table, indexed by the ten's digit, giving a total which is the converted number.

The values in the table consist of one byte each, having selected so that they subtract out the hexadecimal value of the ten's digit and add back the value of that digit-times-ten in binary. For example, if the original number was $58 (meaning decimal 58 in BCD storage format), we will add the value $E2 (which is 50-$50). $58+$E2 = $3A, which is the correct hexadecimal conversion.

I recently worked on a consulting project which included a lot of mixed decimal and hexadecimal calculations. The project was implemented on a 6511 chip, which has only 192 bytes of RAM. That is total, including the stack! We also had 4096 bytes of EPROM. The system operates in a real-time mode with relatively high-speed interrupts occurring. With these constraints, every routine had to be written to use the minimum amount of RAM and to be as fast as possible. A few extra bytes of code would be all right, because 4096 bytes of EPROM was more than enough. In situations like this, programs like the one in lines 1200-1300 come in real handy.

=================================================================
DOCUMENT :AAL-8411:Articles:RAMWorks.MB.txt
=================================================================

A Whole Megabyte for your Apple //e........ Bob Sander-Cederlof

Both Applied Engineering and Saturn have announced 1 Mbyte cards for the //e. Saturn's, I understand, plugs into any slot 1-7; this of course makes it a little non-standard compared to other //e memory expanders when it comes to software access.

The new board from Applied Engineering, called RAM WORKS, fits in the //e auxiliary slot. You get 80 column text and double hi-res, with anywhere from 64K to 1 Megabyte of expansion RAM in 64K or 256K increments. You can buy RAM WORKS already expanded, or expand it yourself later. Prices: 64K = $179, 128K = $249, 256K = $449, 512K = $799, and 1Meg = $1499. The first 512K fits one a normal size card, about 6 inches long. The second 512K come in a piggy-back card which attaches to the main card. Another option, an RGB video generator ($129), attaches to the front of the memory card.

The megabyte is divided into 16 chapters of 64K each. You select which one is active by storing a value from $00 to $0F in a register at $C073. Then the normal //e maze of soft switches lets you access the active chapter the same way you would access Apple's standard 64K card.

RAM WORKS has some new design ideas, for which patents are pending, including a power saving circuit and a video refresh circuit. The latter eliminates the annoying screen flicker that normally occurs when you switch chapters with older expansion cards.

Low cost software options available with RAM WORKS include disk emulation for DOS and ProDOS, and workspace expansion for Appleworks. Standard ProDOS will turn Apple's RAM card into a half-size RAMdisk, but with RAM WORKS you get a full megabyte!

If you like the idea of souping up your //e, one of these boards plus a new 65802 processor may be just the ticket!
1

==============================================================================
DOCUMENT :AAL-8411:DOS3.3:Opcodes.65816.txt
==============================================================================

```
 1000  *SAVE NEW 65816 OPCODES
 1010         .OP CON
 1020  ROCKWELL    .EQ 0
 1030  *---------------------------------
 1040  ZP     .EQ $45
 1050  LONG   .EQ $300     <<<24-BIT VALUE>>>
 1060  *---------------------------------
 1070  TEST   BRA .1
 1080  *---------------------------------
 1090         ORA (ZP)
 1100         AND (ZP)
 1110         EOR (ZP)
 1120         ADC (ZP)
 1130         STA (ZP)
 1140         LDA (ZP)
 1150         CMP (ZP)
 1160         SBC (ZP)
 1170  *---------------------------------
 1180  .1     JMP (TEST),X
 1190  *---------------------------------
 1200         BIT #$45     IMMEDIATE
 1210         BIT ZP       ZERO PAGE
 1220         BIT LONG     ABSOLUTE
 1230         BIT ZP,X     ZP,X
 1240         BIT LONG,X   ABS,X
 1250  *---------------------------------
 1260         INC
 1270         DEC
 1280  *---------------------------------
 1290         PHX
 1300         PLX
 1310         PHY
 1320         PLY
 1330  *---------------------------------
 1340         STZ ZP
 1350         STZ LONG
 1360         STZ ZP,X
 1370         STZ LONG,X
 1380  *---------------------------------
 1390         TSB ZP
 1400         TSB LONG
 1410         TRB ZP
 1420         TRB LONG
 1430  *---------------------------------
 1440      .DO ROCKWELL
 1450         RMB 0,ZP
 1460         RMB 1,ZP
 1470         RMB 2,ZP
 1480         SMB 0,ZP
```

```
1490        SMB 1,ZP
1500        SMB 2,ZP
1510 *--------------------------------
1520        BBR 0,ZP,SS
1530        BBR 1,ZP,SS
1540        BBR 2,ZP,SS
1550 SS     BBS 0,ZP,SS
1560        BBS 1,ZP,SS
1570        BBS 2,ZP,SS
1580     .FIN
1590 *--------------------------------
1600     .DO WDM.65816
1610 *---ABSOLUTE LONG----------------
1620        ORA LONG     0F LL HH BB
1630        AND LONG     2F LL HH BB
1640        EOR LONG     4F LL HH BB
1650        ADC LONG     6F LL HH BB
1660        STA LONG     8F LL HH BB
1670        LDA LONG     AF LL HH BB
1680        CMP LONG     CF LL HH BB
1690        SBC LONG     EF LL HH BB
1700 *---ABSOLUTE INDEXED LONG--------
1710        ORA LONG,X   1F LL HH BB
1720        AND LONG,X   3F LL HH BB
1730        EOR LONG,X   5F LL HH BB
1740        ADC LONG,X   7F LL HH BB
1750        STA LONG,X   9F LL HH BB
1760        LDA LONG,X   BF LL HH BB
1770        CMP LONG,X   DF LL HH BB
1780        SBC LONG,X   FF LL HH BB
1790 *---DIRECT INDIRECT LONG---------------
1800 *      ADDRESS POINTED TO IS 3 BYTES LONG
1810 *      NEED A SYNTAX CHANGE HERE!!!
1820 *      I PROPOSE "ORA.L"
1830        ORA.L (ZP)     07 ZP
1840        AND.L (ZP)     27 ZP
1850        EOR.L (ZP)     47 ZP
1860        ADC.L (ZP)     67 ZP
1870        STA.L (ZP)     87 ZP
1880        LDA.L (ZP)     A7 ZP
1890        CMP.L (ZP)     C7 ZP
1900        SBC.L (ZP)     E7 ZP
1910 *---DIRECT INDIRECT INDEXED ZP--------
1920        ORA.L (ZP),Y   17 ZP
1930        AND.L (ZP),Y   37 ZP
1940        EOR.L (ZP),Y   57 ZP
1950        ADC.L (ZP),Y   77 ZP
1960        STA.L (ZP),Y   97 ZP
1970        LDA.L (ZP),Y   B7 ZP
1980        CMP.L (ZP),Y   D7 ZP
1990        SBC.L (ZP),Y   F7 ZP
2000 *---STACK RELATIVE---------------
2010 *      NEED A SYNTAX CHANGE HERE!!!
2020 *      I PROPOSE "ORA.S"
```

```
2030        ORA.S SR       03 SR
2040        AND.S SR       23 SR
2050        EOR.S SR       43 SR
2060        ADC.S SR       63 SR
2070        STA.S SR       83 SR
2080        LDA.S SR       A3 SR
2090        CMP.S SR       C3 SR
2100        SBC.S SR       E3 SR
2110  *---STACK RELATIVE INDIRECT INDEXED-----
2120  *     NEED A SYNTAX CHANGE HERE!!!
2130  *     I PROPOSE "ORA.S"
2140        ORA.S (SR),Y   13 SR
2150        AND.S (SR),Y   33 SR
2160        EOR.S (SR),Y   53 SR
2170        ADC.S (SR),Y   73 SR
2180        STA.S (SR),Y   93 SR
2190        LDA.S (SR),Y   B3 SR
2200        CMP.S (SR),Y   D3 SR
2210        SBC.S (SR),Y   F3 SR
2220  *--------------------------------
2230        PEA $HHLL    F4 LL HH
2240  *     EQUIVALENT TO: LDA /ABS;PHA;LDA #ABS;PHA
2250        PEI ZP       D4 ZP     PUSH (ZP+1),(ZP)
2260  *     EQUIVALENT TO: LDA ZP+1;PHA;LDA ZP;PHA
2270        PER ABS      62 LL HH
2280  *     PUSHES PC+HHLL
2290  *--------------------------------
2300        PHB          8B   PUSH DBR
2310        PLB          AB   PULL DBR
2320        PHD          0B   PUSH D-REG
2330        PLD          2B   PULL D-REG
2340        PHK          4B   PUSH PBR
2350        RTL          6B   RTS LONG
2360  *--------------------------------
2370        REP #BYTE    C2 XX   RESET BITS IN STATUS
2380  *     PHP;PLA;EOR#FF;ORA#XX;EOR#FF;PHA;PLP
2390        SEP #BYTE    E2 XX   SET BITS IN STATUS
2400  *     PHP;PLA;ORA#XX;PHA;PLP
2410  *--------------------------------
2420        TCS          1B   C --> S
2430        TSC          3B   S --> C
2440        TCD          5B   C --> D
2450        TDC          7B   D --> C
2460        TXY          9B   X --> Y
2470        TYX          BB   Y --> X
2480        WAI          CB   WAIT FOR INTERRUPT
2490        STP          DB   STOP UNTIL RESET
2500        XBA          EB   A <--> B (REGISTERS)
2510        XCE          FB   C <--> E (STATUS BITS)
2520  *--------------------------------
2530        COP #XX      02 XX  COPROCESSOR INTERRUPT
2540        JSL LONG     22 LL HH BB
2550        WDM          42  <<<RESERVED>>>
2560        BRL ADDR     82 LL HH  BRANCH RELATIVE
```

```
2570  *--------------------------------
2580         JMP (TEST,X) 7C LL HH
2590         JSR (TEST,X) FC LL HH
2600         JML (TEST)   DC LL HH   3 BYTES AT TEST ARE LONG ADDRESS
2610         JMP LONG     5C LL HH BB
2620  *--------------------------------
2630  *      LDA #NUMBER OF BYTES
2640  *      LDX #SOURCE ADDRESS
2650  *      LDY #DESTINATION ADDRESS
2660         MVP SBANK,DBANK  44 DB SB
2670         MVN SBANK,DBANK  54 DB SB
2680  *--------------------------------
2690      .FIN
2700  *--------------------------------
2710  *      WE EVIDENTLY NEED A NEW DIRECTIVE TO TELL
2720  *      THE ASSEMBLER WHETHER TO USE 8- OR 16-BIT OPERANDS
2730  *      IN IMMEDIATE MODE.
```

==============================================================================
DOCUMENT :AAL-8411:DOS3.3:S.DP18.FUNC.LOG.txt
==============================================================================

```
1000  *SAVE S.DP18 FUNC LOG
1010  *--------------------------------
1020  AS.OVRFLW  .EQ $E8D5
1030  AS.ILLERR  .EQ $E199
1040  *--------------------------------
1050  POLY.1     .EQ $FFFF
1060  POLY.N     .EQ $FFFF
1070  DADD       .EQ $FFFF
1080  DSUB       .EQ $FFFF
1090  DMULT      .EQ $FFFF
1100  DDIV       .EQ $FFFF
1110  DP.TRUE    .EQ $FFFF
1120  MOVE.YA.ARG.1  .EQ $FFFF
1130  MOVE.YA.DAC.1       .EQ $FFFF
1140  SWAP.DAC.ARG   .EQ $FFFF
1150  MOVE.TEMP1.ARG      .EQ $FFFF
1160  MOVE.TEMP2.ARG      .EQ $FFFF
1170  MOVE.TEMP3.ARG      .EQ $FFFF
1180  MOVE.DAC.ARG        .EQ $FFFF
1190  MOVE.TEMP3.DAC      .EQ $FFFF
1200  MOVE.DAC.TEMP1      .EQ $FFFF
1210  MOVE.DAC.TEMP2      .EQ $FFFF
1220  MOVE.DAC.TEMP3      .EQ $FFFF
1230  NORMALIZE.DAC       .EQ $FFFF
1240  *--------------------------------
1250  DAC.EXPONENT .BS 1
1260  DAC.HI       .BS 10
1270  DAC.SIGN     .BS 1
1280  *--------------------------------
1290  ARG.EXPONENT .BS 1
1300  ARG.HI       .BS 10
1310  ARG.SIGN     .BS 1
1320  *--------------------------------
1330  SIGN         .BS 1
1340  INTPWR       .BS 1
1350  *--------------------------------
1360  CON.ONE    .HS 41.10000.00000.00000.00000
1370  CON.1HALF  .HS 40.50000.00000.00000.00000
1380  CON.SQR10  .HS 41.31622.77660.16837.93320
1390  *--------------------------------
1400  *       EXP (DAC)    E^DAC
1410  *              OR    10^DAC
1420  *       #1446 IN HART, ET AL
1430  *--------------------------------
1440  P.EXP    .EQ *
1450  P.EXP.N  .EQ 3
1460           .HS 42.31341.17940.19730.48777
1470           .HS 44.45618.28316.94656.35848
1480           .HS 46.13433.11347.35855.59034
```

```
1490           .HS 46.76025.44794.41265.39434
1500  Q.EXP    .EQ *
1510  Q.EXP.N  .EQ 4
1520           .HS 43.47705.44030.08207.98775
1530           .HS 45.29732.60655.85996.83303
1540           .HS 46.40843.69796.67736.28236
1550           .HS 46.66034.86505.27141.54491
1560  *--------------------------------
1570  CON.LOGE .HS 40.43429.44819.03251.82765
1580  *--------------------------------
1590  DP.EXP.NULL
1600         JMP DP.TRUE  E^0 = 10^0 = 1.0
1610  DP.EXP.OVERFLOW
1620         JMP AS.OVRFLW
1630  *--------------------------------
1640  DP.EXPE
1650         LDA DAC.EXPONENT
1660         BEQ DP.EXP.NULL
1670         LDA #CON.LOGE
1680         LDY /CON.LOGE
1690         JSR MOVE.YA.ARG.1
1700         JSR DMULT    CHANGE TO 10^X
1710  *--------------------------------
1720  DP.EXP10
1730         LDX DAC.EXPONENT       10^0 = 1
1740         BEQ DP.EXP.NULL
1750  *---HANDLE NEGATIVE POWERS-------
1760         LDA DAC.SIGN SAVE FOR 1/EXP IF NEGATIVE
1770         STA SIGN
1780         LDA #0       GET ABS(X)
1790         STA DAC.SIGN
1800  *---SPLIT INTEGER & FRACTION-----
1810         CPX #$43     THREE OR MORE INTEGER DIGITS?
1820         BCS DP.EXP.OVERFLOW   YES, OVERFLOW
1830         LDA #0       ...ALL FRACTIONAL
1840         STA INTPWR
1850         CPX #$41
1860         BCC .3       ...NO INTEGRAL PART
1870         LDA DAC.HI   ...1 OR 2 DIGITS
1880         LSR
1890         LSR
1900         LSR
1910         LSR
1920         STA INTPWR
1930         LDA DAC.HI
1940         AND #$0F
1950         STA DAC.HI
1960         CPX #$41     ONE OR TWO DIGITS?
1970         BEQ .2       ...ONE DIGIT INTEGER
1980         LDA INTPWR    DIGIT*10
1990         ASL
2000         ASL
2010         ADC INTPWR
2020         ASL
```

```
2030        ADC DAC.HI
2040        STA INTPWR
2050        LDX #0
2060        STX DAC.HI
2070 .2     JSR NORMALIZE.DAC   ADJUST REMAINING FRACTION
2080        BNE .3              FRACTION NOT 0
2090        JSR DP.TRUE         10^0 = 1
2100        JMP .7
2110 *---ADJUST FRACTION SO < .5------
2120 .3     LDA DAC.EXPONENT
2130        CMP #$40
2140        BCC .4
2150        LDA DAC.HI
2160        CMP #$50
2170 .4     PHP          REMEMBER...
2180        BCC .5       ...ALREADY < .5
2190        SBC #$50
2200        STA DAC.HI
2210        JSR NORMALIZE.DAC
2220        BNE .5       ...REST OF FRACTION NOT 0
2230        PLA          POP SAVED STATUS
2240        LDA #CON.SQR10
2250        LDY /CON.SQR10
2260        JSR MOVE.YA.DAC.1
2270        JMP .7
2280 *---COMPUTE 10^.XXXX-------------
2290 .5     JSR MOVE.DAC.TEMP1    SAVE X
2300        JSR MOVE.DAC.ARG
2310        JSR DMULT             GET X^2
2320        JSR MOVE.DAC.TEMP2    SAVE X^2
2330        LDA #P.EXP            COMPUTE P(X^2)
2340        LDY /P.EXP
2350        LDX #P.EXP.N
2360        JSR POLY.N
2370        JSR MOVE.TEMP1.ARG    COMPUTE XP(X^2)
2380        JSR DMULT
2390        JSR MOVE.DAC.TEMP3    SAVE XP(X^2)
2400        LDA #Q.EXP            COMPUTE Q(X^2)
2410        LDY /Q.EXP
2420        LDX #Q.EXP.N
2430        JSR POLY.1
2440        JSR MOVE.DAC.TEMP2    SAVE Q(X^2)
2450        JSR MOVE.TEMP3.ARG    NUMERATOR = Q+XP
2460        JSR DADD              Q(X^2)+XP(X^2)
2470        JSR MOVE.DAC.TEMP1    SAVE UMERATOR
2480        JSR MOVE.TEMP2.ARG    DENOMINATOR = Q-XP
2490        JSR MOVE.TEMP3.DAC
2500        JSR DSUB              Q(X^2)-XP(X^2)
2510        JSR MOVE.TEMP1.ARG    10^.XXX = N/D
2520        JSR DDIV
2530 *---ADJUST BY SQR(10)------------
2540        PLP          SEE IF ADJUSTMENT NEEDED
2550        BCC .7       ...NO
2560        LDA #CON.SQR10
```

```
2570        LDY /CON.SQR10
2580        JSR MOVE.YA.ARG.1
2590        JSR DMULT
2600 *---ADD INTEGRAL POWER-----------
2610 .7     CLC
2620        LDA DAC.EXPONENT
2630        ADC INTPWR
2640        BPL .8       ...NO OVERFLOW
2650        JMP DP.EXP.OVERFLOW
2660 .8     STA DAC.EXPONENT
2670 *---ADJUST FOR SIGN--------------
2680        LDA SIGN     GET ORIGINAL SIGN
2690        BPL .9       POSITIVE, WE ARE DONE
2700        LDA #CON.ONE NEGATIVE, FORM RECIPROCAL
2710        LDY /CON.ONE
2720        JSR MOVE.YA.ARG.1
2730        JSR DDIV
2740 .9     RTS
2750 *--------------------------------
2760 *      LN (DAC)  LOG E (DAC)
2770 *           OR   LOG 10 (DAC)
2780 *      #2330 IN HART, ET AL
2790 *--------------------------------
2800 P.LOG    .EQ *
2810 P.LOG.N  .EQ 5
2820          .HS C2.14933.41871.23101.49868
2830          .HS 43.30132.34734.14748.46138
2840          .HS C4.17255.36265.00653.03387
2850          .HS 44.40598.33123.94476.21513
2860          .HS C4.41923.45602.07081.07911
2870          .HS 44.15764.33484.51127.69255
2880 Q.LOG    .EQ *
2890 Q.LOG.N  .EQ 6
2900          .HS C2.67696.41190.46224.52758
2910          .HS 43.76357.00230.09155.79877
2920          .HS C4.32000.87986.36664.12225
2930          .HS 44.61216.00041.77468.78069
2940          .HS C4.54315.94950.92575.25735
2950          .HS 44.18149.36120.76616.30282
2960 *--------------------------------
2970 CON.LN10 .HS 41.23025.85092.99404.56840
2980 *--------------------------------
2990 DP.LOGE
3000        JSR DP.LOG10
3010        LDA #CON.LN10     CONVERT LOG10 TO LN
3020        LDY /CON.LN10
3030        JSR MOVE.YA.ARG.1
3040        JMP DMULT
3050 *--------------------------------
3060 DP.LOG.ERR
3070        JMP AS.ILLERR
3080 *--------------------------------
3090 DP.LOG10
3100        LDA DAC.SIGN      CHECK RANGE
```

```
3110        BMI DP.LOG.ERR    ...NEGATIVE
3120        LDA DAC.EXPONENT
3130        BEQ DP.LOG.ERR    ...ZERO
3140        STA INTPWR        SAVE POWER OF 10
3150 *---ADJUST RANGE-----------------
3160        LDA #$40          MAKE FRACTION .1 TO .9999
3170        STA DAC.EXPONENT
3180        LDA #CON.SQR10    1/SQR(10) ... SQR(10)
3190        LDY /CON.SQR10
3200        JSR MOVE.YA.ARG.1
3210        JSR DMULT
3220 *---FORM (X-1)/(X+1)-------------
3230        JSR MOVE.DAC.TEMP1
3240        JSR MOVE.DAC.ARG
3250        JSR DP.TRUE       GET 1 IN DAC
3260        JSR DSUB          X-1
3270        JSR MOVE.DAC.TEMP2 SAVE IT
3280        JSR DP.TRUE       GET 1 IN DAC
3290        JSR MOVE.TEMP1.ARG
3300        JSR DADD          X+1
3310        JSR MOVE.TEMP2.ARG
3320        JSR DDIV          X-1/X+1
3330 *---NUMERATOR = Z*P(Z^2)---------
3340        JSR MOVE.DAC.TEMP1 SAVE IT
3350        JSR MOVE.DAC.ARG
3360        JSR DMULT         Z^2
3370        JSR MOVE.DAC.TEMP2 SAVE Z^2
3380        LDA #P.LOG
3390        LDY /P.LOG
3400        LDX #P.LOG.N
3410        JSR POLY.N
3420        JSR MOVE.TEMP1.ARG
3430        JSR DMULT         Z*P(Z^2)
3440        JSR MOVE.DAC.TEMP1
3450 *---DENOMINATOR = Q(Z^2)---------
3460        LDA #Q.LOG
3470        LDY /Q.LOG
3480        LDX #Q.LOG.N
3490        JSR POLY.1
3500        JSR MOVE.TEMP1.ARG
3510        JSR DDIV          Z*P(Z^2)/Q(Z^2)
3520 *---ADD INTEGER POWER------------
3530        SEC
3540        LDA INTPWR        GET POWER OF 10
3550        SBC #$40
3560        BEQ .5            ...0, NO NEED TO ADD ANYTHING
3570        STA ARG.SIGN
3580        BCS .1            ...1 TO 63
3590        EOR #$FF          MAKE IT POSITIVE
3600        ADC #1
3610 .1     LDY #0
3620        STY ARG.HI
3630        LDX #$41
3640        CMP #10
```

```
3650        BCC .3           1...9
3660        INX              10...63
3670 .2     STA ARG.HI       STORE REMAINDER
3680        SBC #10
3690        INY              INC. QUOTIENT
3700        BCS .2           ...TRY ANOTHER SUBTRACTION
3710        DEY              CORRECT QUOTIENT
3720        TYA              GET QUOTIENT
3730 .3     ASL              LEFT JUSTIFY
3740        ASL
3750        ASL
3760        ASL
3770        ORA ARG.HI       MERGE WITH NEXT DIGIT
3780        STA ARG.HI
3790        STX ARG.EXPONENT  $41 OR $42
3800        LDX #9           CLEAR REST OF ARG
3810        LDA #0
3820 .4     STA ARG.HI,X
3830        DEX
3840        BNE .4
3850        JSR DADD
3860 *---SUBTRACT 0.5-----------------
3870 .5     LDA #CON.1HALF
3880        LDY /CON.1HALF
3890        JSR MOVE.YA.ARG.1
3900        LDA #$FF
3910        STA ARG.SIGN
3920        JMP DADD
3930 *--------------------------------
```

```
=============================================================
DOCUMENT :AAL-8411:DOS3.3:S.Macro.Ex.txt
=============================================================
```

```
 1000  *SAVE S.MACRO EXAMPLES
 1010  *--------------------------------
 1020  *    BY SANDY GREENFARB
 1030  *--------------------------------
 1040  *
 1050  *    PARAMETERS CAN SUBSTITUTE ANYWHERE,
 1060  *      EVEN IN OPCODES
 1070  *--------------------------------
 1080         .MA TEST     VALUE,CONDITION,LABEL
 1090         CMP ]1
 1100         B]2 ]3
 1110         .EM
 1120  *
 1130         >TEST #3,CC,SMALLER
 1140         >TEST TYPE,EQ,SAME
 1150  *
 1160  TYPE   .DA #35
 1170  SAME   NOP
 1180  SMALLER NOP
 1190  *--------------------------------
 1200  *
 1210  *   MACROS CAN SIMPLIFY CODE FOR EFFICIENCY
 1220  *--------------------------------
 1230         .MA SETD     VALUE,VARIABLE
 1240         LDA #]1      LO-BYTE
 1250         STA ]2
 1260      .DO ]1/256*257-]1  ARE LOW AND HI EQUAL?
 1270         LDA /]1
 1280      .ELSE
 1290  *                   HI = LO-BYTE
 1300      .FIN
 1310         STA ]2+1
 1320         .EM
 1330  *
 1340         >SETD $1234,VALUE
 1350         >SETD $2323,VALUE
 1360  *
 1370  VALUE  .BS 2
 1380  *--------------------------------
 1400  *
 1410  *   MACROS CAN PREVENT PROGRAMMING MISTAKES
 1420  *      SUCH AS OVER-WRITING WHEN YOU COPY
 1430  *      ONE VARIABLE INTO ANOTHER.
 1440  *--------------------------------
 1450         .MA MOVD     VAR1,VAR2
 1460      .DO ]2-]1-1
 1470         LDA ]1       NO OVERLAP
 1480         STA ]2
 1490         LDA ]1+1
```

```
1500         STA ]2+1
1510      .ELSE
1520         LDA ]1+1     THIS CODE BUILT WHEN THE
1530         STA ]2+1     VARIABLES OVERLAP
1540         LDA ]1
1550         STA ]2
1560      .FIN
1570         .EM
1580  *
1590         >MOVD $11,$22
1600         >MOVD $28,VALUE
1610         >MOVD $11,$12
1620  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8411:DOS3.3:S.MASK.INDEX.txt
==============================================================================

 1000  *SAVE S.MASK --> INDEX
 1010  *--------------------------------
 1020  TEST   LDY #$01
 1030  .1     TYA
 1040         JSR $FDDA
 1050         TYA
 1060         JSR SMALLEST.WAY
 1070         JSR HEX
 1080         TYA
 1090         JSR WAY.WITH.X
 1100         JSR HEX
 1110         TYA
 1120         JSR WAY.WITHOUT.X
 1130         JSR HEX
 1140         TYA
 1150         JSR ANOTHER.WAY.WITHOUT.X
 1160         JSR HEX
 1170         TYA
 1180         JSR STRAIGHT.TESTING.WAY
 1190         JSR HEX
 1200         JSR $FD8E
 1210         TYA
 1220         ASL
 1230         TAY
 1240         BCC .1
 1250         RTS
 1260  *--------------------------------
 1270  HEX    PHA
 1280         LDA #"-"
 1290         JSR $FDED
 1300         PLA
 1310         JMP $FDDA
 1320  *--------------------------------
 1330  *   WAY WITH FEWEST BYTES
 1340  *      8 BYTES
 1350  *      MIN:  16 CYCLES
 1360  *      MAX:  65 CYCLES
 1370  *      AVE:  40.5 CYCLES
 1380  *--------------------------------
 1390  SMALLEST.WAY
 1400         LDX #8
 1410  .1     DEX
 1420         ASL
 1430         BCC .1
 1440         TXA
 1450         RTS
 1460  *--------------------------------
 1470  *   FASTER WAY USING X-REGISTER
 1480  *      26 BYTES
```

```
1490  *      MIN: 25 CYCLES
1500  *      MAX: 42 CYCLES
1510  *      AVE: 33.5 CYCLES
1520  *--------------------------------
1530  WAY.WITH.X
1540         LDX #0       KEEP INDEX IN X
1550         CMP #$10     80-40-20-10 / 08-04-02-01
1560         BCC .1       ...8,4,2,1
1570         LSR          ...80,40,20,10
1580         LSR          SHIFT OVER TO 8,4,2,1
1590         LSR
1600         LSR
1610         LDX #4       AND BUMP INDEX BY 4
1620  .1     CMP #$04     08-04 / 02-01
1630         BCC .2       ...2,1
1640         LSR          ...8,4
1650         LSR          SHIFT OVER TO 2,1
1660         INX          AND BUMP INDEX BY 2
1670         INX
1680  .2     LSR          02 / 01
1690         BEQ .3       ...01
1700         INX          ...02, BUMP INDEX
1710  .3     TXA          GET RESULT
1720         RTS
1730  *--------------------------------
1740  *   WAY WITHOUT USING X-REGISTER
1750  *      23 BYTES
1760  *      MIN: 14 CYCLES
1770  *      MAX: 30 CYCLES
1780  *      AVE: 22 CYCLES
1790  *--------------------------------
1800  WAY.WITHOUT.X
1810         LSR          40-20-10-08-04-02-01-00
1820         CMP #$04
1830         BCC .2       ...2,1,0
1840         BEQ .3       ...4, SHOULD BE 3
1850         LSR          20-10-08-04
1860         LSR          10-08-04-02
1870         LSR          08-04-02-01
1880         LSR          04-02-01-00
1890         CMP #4
1900         BCC .1       2,1,0 INTO 6,5,4
1910         LDA #2       4 INTO 7
1920  .1     ADC #4
1930  .2     RTS
1940  .3     SBC #1       4 INTO 3
1950         RTS
1960  *--------------------------------
1970  *   ANOTHER WAY WITHOUT X-REGISTER
1980  *      32 BYTES
1990  *      MIN: 14 CYCLES
2000  *      MAX: 24 CYCLES
2010  *      AVE: 18.375 CYCLES
2020  *--------------------------------
```

```
2030  ANOTHER.WAY.WITHOUT.X
2040         CMP #$08      80-40-20-10-08-04-02-01
2050         BCC .5        ...4,2,1
2060         BEQ .4        ...8, SHOULD BE 3
2070         CMP #$40
2080         BCC .2        ...20,10
2090         BEQ .1        ...40
2100         LDA #7
2110         RTS
2120  .1     LDA #6
2130         RTS
2140  .2     CMP #$20
2150         BEQ .3
2160         LDA #4
2170         RTS
2180  .3     LDA #5
2190         RTS
2200  .4     SBC #2
2210  .5     LSR
2220         RTS
2230  *---------------------------------
2240  *   STRAIGHTFORWARD TESTING APPROACH
2250  *      33 BYTES
2260  *      MIN:  14 CYCLES
2270  *      MAX:  27 CYCLES
2280  *      AVE:  18.5 CYCLES
2290  *---------------------------------
2300  STRAIGHT.TESTING.WAY
2310         CMP #$08
2320         BCC .5
2330         BEQ .4
2340         CMP #$20
2350         BCC .3
2360         BEQ .2
2370         CMP #$80
2380         BCC .1
2390         LDA #7
2400         RTS
2410  .1     LDA #6
2420         RTS
2430  .2     LDA #5
2440         RTS
2450  .3     LDA #4
2460         RTS
2470  .4     LDA #3
2480         RTS
2490  .5     LSR           CONVERT 4,2,1 TO 2,1,0
2500         RTS
2510  *---------------------------------
```

```
================================================================
DOCUMENT :AAL-8411:DOS3.3:S.New80ColMD.txt
================================================================

 1000  *SAVE S.NEW 80 COL MONITOR DUMP
 1010  *--------------------------------
 1020  *   TO INSTALL,
 1030  *      1.  ASSEMBLE THIS PROGRAM
 1040  *      2.  ENTER THESE MONITOR COMMANDS
 1050  *    $C083 C083 FCC9<CC9.CEFM
 1060  *    $FDBE:C9 FC N FDA6:F N FDB0:F
 1070  *--------------------------------
 1080  *   BY JAN EUGENIDES & BOB S-C
 1090  *--------------------------------
 1100  CH      .EQ $24
 1110  A1      .EQ $3C,3D
 1120  A2      .EQ $3E,3F
 1130  A4      .EQ $42,43
 1140  BUFFER .EQ $2F0
 1150  PRBYTE .EQ $FDDA
 1160  COUT    .EQ $FDED
 1170  PRBLNK .EQ $F948
 1180  *--------------------------------
 1190          .OR $FCC9
 1200          .TA $CC9
 1210  *--------------------------------
 1220  PATCH  PHA          SAVE BYTE
 1230         LDA A1       COMPUTE INDEX
 1240         AND #$0F     0...F
 1250         TAX
 1260         PLA          GET BYTE AGAIN
 1270         STA BUFFER,X SAVE IN BUFFER
 1280         JSR PRBYTE   PRINT ON SCREEN
 1290         INX          GET # BYTES THIS LINE
 1300         STX A4       SAVE IN A4L
 1310         CPX #$10     END OF LINE?
 1320         BEQ .1       ...YES, PRINT ASCII CHARS
 1330         LDA A1       ...NO, SEE IF END OF RANGE
 1340         CMP A2
 1350         LDA A1+1
 1360         SBC A2+1
 1370         BCC .4       ...NO, RETURN
 1380  .1     JSR PRBLNK   PRINT 3 SPACES
 1390         LDX #0       PRINT ASCII CHARS FROM BUFFER
 1400  .2     LDA BUFFER,X GET CHAR
 1410         ORA #$80     MAKE NORMAL VIDEO
 1420         CMP #$A0     TRAP CONTROL CHARS
 1430         BCS .3       ...NOT CONTROL CHAR
 1440         LDA #$AE     ...CTRL, SUBSTITUTE "."
 1450  .3     JSR COUT     PRINT CHAR
 1460         INX          NEXT
 1470         CPX A4       END OF LIST?
 1480         BCC .2       ...NOT YET
```

```
1490  .4     RTS          RETURN
```

```
===========================================================================
DOCUMENT :AAL-8411:DOS3.3:S.NewSQR.Rtn.txt
===========================================================================

 1000  *SAVE S.NEW SQR ROUTINE
 1010  *--------------------------------
 1020  *      SQR (DAC)
 1030  *--------------------------------
 1040  ERR.SQ JMP AS.ILLERR  ILLEGAL QUANTITY
 1050  DP.SQR.0 RTS
 1060  DP.SQR LDA DAC.EXPONENT
 1070         BEQ DP.SQR.0  SQR(0)=0
 1080         LDA DAC.SIGN
 1090         BMI ERR.SQ   MUST BE POSITIVE
 1100         JSR MOVE.DAC.TEMP3 SAVE X
 1110  *---APPROX. ROOT OF .1 - 1-------
 1120         LDA DAC.HI   CONVERT TWO DIGITS TO BINARY
 1130         AND #$0F     SAVE LO DIGIT
 1140         CMP #5       01234 OR 56789
 1150         PHP          SAVE ANSWER
 1160         LDA DAC.HI   GET HI DIGIT
 1170         LSR
 1180         LSR
 1190         LSR
 1200         LSR          $01...$09
 1210         PLP          01234 OR 56789
 1220         ROL          $02...$13
 1230         TAX
 1240         LDA SQR.TBL,X
 1250         STA DAC.HI
 1260  *---TAKE HALF OF EXPONENT--------
 1270         LDA DAC.EXPONENT
 1280         SEC
 1290         SBC #$40     REMOVE OFFSET
 1300         ROR          DIVIDE BY TWO (KEEP SIGN)
 1310         PHP          SAVE ODD/EVEN BIT
 1320         CLC
 1330         ADC #$C0     RE-BIAS EXPONENT
 1340         STA DAC.EXPONENT
 1350         PLP
 1360         BCC .1       EVEN, DON'T MULT BY SQR(10)
 1370  *---ADJUST APPROX FOR ODD EXP----
 1380         LDA #CON.SQR10
 1390         LDY /CON.SQR10
 1400         JSR MOVE.YA.ARG.1
 1410         JSR DMULT
 1420  *---THREE NEWTON ITERATIONS------
 1430  .1     LDA #3
 1440         STA TEMP3
 1450  .2     JSR MOVE.DAC.TEMP2     TEMP2 = Y
 1460         JSR MOVE.TEMP3.ARG     GET X
 1470         JSR DDIV               X/Y
 1480         JSR MOVE.TEMP2.ARG
```

```
        JSR DADD              X/Y+Y
        LDA #CON.HALF
        LDY /CON.HALF
        JSR MOVE.YA.ARG.1
        JSR DMULT             (X/Y+Y)/2
        DEC TEMP3             ANY MORE?
        BNE .2                ...YES
*---ONE MORE NEWTON ITERATION----
        JSR MOVE.DAC.TEMP2    TEMP2 = Y
        JSR MOVE.TEMP3.ARG    GET X
        JSR DDIV              X/Y
        JSR MOVE.TEMP2.ARG
        LDA #$FF
        STA ARG.SIGN
        JSR DADD              X/Y-Y
        LDA #CON.HALF
        LDY /CON.HALF
        JSR MOVE.YA.ARG.1
        JSR DMULT             (X/Y-Y)/2
        JSR MOVE.TEMP2.ARG
        JMP DADD              Y + (X/Y-Y)/2
*--------------------------------
SQR.TBL    .EQ *-2  (NO ENTRIES AT 0...1)
           .HS 35.42.47.52.57.61.65.69.72
           .HS 76.79.82.85.88.91.94.96.99
CON.SQR10  .HS 41316227766016837933 20
CON.HALF   .HS 40500000000000000000 00
*--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8411:DOS3.3:S.QUICK.DEC.HEX.txt
==================================================================

 1000  *SAVE S.QUICK DEC-HEX
 1010  *---------------------------------
 1020  T      LDA #0
 1030         STA 0
 1040  .1     LDA 0
 1050         JSR DEC.HEX.2
 1060         JSR $FDDA
 1070         LDA #" "
 1080         JSR $FDED
 1090         JSR $FDED
 1100         SED
 1110         CLC
 1120         LDA 0
 1130         ADC #1
 1140         STA 0
 1150         CLD
 1160         CMP #0
 1170         BNE .1
 1180         RTS
 1190  *---------------------------------
 1200  DEC.HEX
 1210         PHA          SAVE BYTE
 1220         LSR
 1230         LSR
 1240         LSR
 1250         LSR
 1260         TAX          HI NYBBLE TO X
 1270         PLA          GET ORIG BYTE
 1280         CLC
 1290         ADC TBL,X
 1300         RTS
 1310  *---------------------------------
 1320  TBL    .DA #0-0,#10-$10,#20-$20,#30-$30
 1330         .DA #40-$40,#50-$50,#60-$60
 1340         .DA #70-$70,#80-$80,#90-$90
 1350  *---------------------------------
 1360  LOW    .EQ 1
 1370  HIGH   .EQ 2
 1380  *---------------------------------
 1390  DEC.HEX.2
 1400         PHA
 1410         AND #$0F     SAVE LOW NYBBLE
 1420         STA LOW
 1430         PLA
 1440         AND #$F0     GET HIGH NYBBLE
 1450         STA HIGH
 1460         LSR          /2
 1470         LSR          /4
 1480         ADC HIGH     /4*5
```

```
1490        LSR         /8*5 = *10/16
1500        ADC LOW     + LOW NYBBLE
1510        RTS
1520  *--------------------------------
```

================================================================
DOCUMENT :AAL-8412:Articles:BBasic.Review.txt
================================================================

Blankenship's Basic........................Bob Sander-Cederlof

John Blankenship has put together an Applesoft enhancement package, at a mouth-watering price. (See his ad elsewhere in this issue for his $20 introductory offer.) He sent me a review copy, so I tried it out.

BBASIC is a large chunk of machine language code that sits between HIMEM and the DOS file buffers. It also sits between you and Applesoft, hiding itself behind a facade of new editing and listing features. BBASIC takes control even in direct mode, giving you an EDIT command, structured listings, and the ability to skip out of long catalogs.

In pure BBASIC, line numbers are used only as line numbers, not as destinations for GOTOs or GOSUBs. A built-in RENUM command soon convinces you to live this way and like it. In place of line-number branches, you use alphabetic "names" for subroutines, and WHEN-ELSE-ENDWHEN for logic flow. John has also added WHILE-ENDWHILE, REPEAT-UNTIL, CASE, and other structured looping and branching words.

During execution, a special COMPILE verb creates a table of "names" used in your program. This speeds up execution.

Hires Text generation is built-in, along with some extensions to the hires graphics. Musical tone generation with control over pitch, duration, and timbre is also included. You also get SORT, SEARCH, and PRINT USING.

I am just scratching the surface. I didn't like every feature, but there is plenty left over. Worth a lot more than $20.

By the way, if John's name sounds familiar, it may be because he is the author of "The Apple House", a book on controlling your home published by Prentice-Hall. John also is a Professor at DeVry Institute.

================================================================================
DOCUMENT :AAL-8412:Articles:CorrectnMVNMVP.txt
================================================================================

Correction re MVN and MVP in 65802.........Bob Sander-Cederlof

In the October AAL I presented a general memory mover written in 65802 code. I stated that the MVP and MVN instructions took 3 cycles-per-byte during the move. I was wrong.

In looking through small tiny print in the preliminary documentation for the chip, I came across the number "7". Shocked, I wrote a little test program which moved 10000 bytes 1000 times. That means the MVN in my test would move a total of 10,000,000 bytes. With a stop watch I clocked the running time at just under 70 seconds. If it had been 3 cycles-per-byte, the test would have run in 30 seconds.

I don't know how I got that "3" in my head, but the right number is "7". Still considerably faster than 6502, though.

=================================================================
DOCUMENT :AAL-8412:Articles:DP18.Trig.txt
=================================================================

18-Digit Arithmetic, Part 8................Bob Sander-Cederlof

Someone pointed out last week that this series is getting a little long.  Well, we are nearing the end.  What we are doing is probably unprecedented in the industry:  listing the source code and explaining it for a large commercially valuable software product.  It takes time and space to break precedents.

This month's installment completes the normal set of math functions, with sine, cosine, and arc tangent.  We even slipped in a simple form of the tangent function.  Still to come are the formatted INPUT and PRINT routines.

Some Elementary Info:

Trigonometry is a frightening word.  (If it doesn't scare you, skip ahead several paragraphs.)  The "-ometry" refers to measurement, but what is a "trigon".  Believe it or not, "trigon" is another name for a triangle.  Trigon means three sides, and figures with three sides just happen to also have three angles.  "Trig" (a nice nickname) is a branch of mathematics dealing with triangles, without which we could not fly to the moon, draw a map, or build bridges.  Strangely enough, much of electronics also uses trig funtions ... are electrons triangular?

When I took trig in high school, long before the day of personal calculators, we used trig tables.  (These were not articles of furniture made in the local woodshop, but rather long lists of strange numbers printed and bound into books.)  The tables contained values for various ratios of the sides of a triangle having one 90-degree angle.  Now we use calculators or computers, but obviously the trig tables would not fit in them.  Instead, approximation formulas are used.

In high school, we talked about six different ratios:  sine, cosine, tangent, cotangent, secant, and cosecant.  When it is all boiled down, we really only need the sine; all the rest are derivable from those.  The sine function gives a a number for any angle.  We frequently need to be able to go from a trig value back to an angle, and the most useful function for that is called the inverse tangent, or arctangent.

Even though I have been talking about triangles, trig functions are even more related to circles.  We compute functions of the angle between any two radii, like the hands on an old-fashioned, pre-digital wrist watch.  When we start talking about circles, we get into radians vs. degrees.

Just as scientists like logarithms to the base e (rather than 10), they also like trig functions based on angles expressed in radians, rather than degrees. Degrees were invented back in Babylon, I understand, and are nice and clean: 360 make a complete circle. Radians are not clean: 360 degrees is two-times-pi radians. Nevertheless, many physical and electronic formulas simplify when angles are expressed in radians. Consequently, calculators and computer languages usually expect your angles to be expressed in radians. Some allow both options. Applesoft expects radians, and so do my DP18 programs.

We commonly think of an angle as being somewhere between 0 and 360 degrees, or the equivalent range in radians. However, angles can actually be any number, from -infinity to +infinity. The numbers beyond one complete circle are valid, but they don't buy much. If you stand in one place and spin around 1445 degrees (4*360 + 5) you will end up pointing the same direction as if you merely swiveled 5 degrees. Therefore the first step in a sine function calculation involves subtracting out all the multiples of a full circle from the angle.

The arctangent function could return an infinite number of answers, but that is impractical. We will return only the principal value, which is the one closest to 0. All others are that value plus or minus any number of full circles. In DP18 the ATN function may have one or two arguments. If you only have one argument, the result will be an angle between -pi/2 and +pi/2. If you specify two arguments, a value between -pi and +pi will be returned.

The Nitty-Gritty:

Enough of this preliminary stuff, let's get into the code. In the listing which follows, you will find entries for four functions: SIN, COS, TAN, and ATN.

Perhaps the easiest is the TAN function, at lines 2530-2630. Since tan=sin/cos, that is all this code does. We lose a little speed and possibly some precision with this simplistic solution, but the TAN function is relatively rarely called.

Next in difficulty is the COS function, lines 1630-1710. Since cos(-x)=cos(x), we start by making the sign positive (lines 1690-1700. Since cos(x)=sin(x+pi/2), we add pi/2 and fall into the SIN function. Simple, but effective.

The SIN function gets more interesting. For very very small angles, within the precision of 20 digits, sin(x)=x. Lines 1780-1810 check for exponents below -10; all angles smaller than 10^-10 are small enough that sin(x)=x.

Next we take advantage of the fact that sin(-x)=-sin(x), at lines 1820-1860. We remember the sign by shoving it on the stack, and force the sign of x positive.

Lines 1870-1950 get the principal angle. I divide x by twopi, and throw away the integral part. The fractional part that remains is a fraction of a full circle, a value between 0 and .999999...9 (not radians, and not degrees either). Note that if x was extremely large there will be no fractional part, and the remainder will be zero. Some SIN function calculators give an error message when this happens, but I chose to let it ride.

Lines 1960-2000 multiply the circle-fraction by four. This gives a number between 0 and 3.99999...9, which I will refer to later as the "circle fraction times four", or c-f-t-f. The integer part is effectively a quadrant number, and the fractional part a fraction within the quadrant:

```
                    |
                  1 | 0
                ---------
                  2 | 3
                    |
```

Lines 2010-2030 determine if the angle is in the first (0) quadrant. If so, no folding need be done.

Lines 2040-2070 determine if the angle is in the second (1) quadrant. If so, we skip ahead to apply the fact that sin(pi/2 + x) = sin(pi/2 - x).

Lines 2080-2160 are executed if the angle is in the 3rd or 4th quadrants (integral part is 2 or 3). Here I apply the fact that sin(pi+x)=-sin(x). I pull the saved sign off the stack, complement it, and shove it back on (lines 2090-2110). Then I subtract 2 from the c-f-t-f, yielding a number between 0 and 1.99999...9. We have folded the third and fourth quadrants over the first and second quadrants. Next lines 2170-2190 determine if the result was in the first quadrant or not.

Lines 2200-2240 fold a second quadrant number into the first quadrant, by applying the fact that sin(pi/2+x) = sin(pi/2-x). Subtacting the c-f-t-f from 2 flips us into the first quadrant.

Lines 2260-2270 pull the sign off the stack and make it the sign of the angle. Remember that now the angle is a fraction (between 0 and .99999...9) of a quadrant. After all these folding operations, the angle might again be very very small, so lines 2280-2300 check for that possibility. If so, sin(x)=x, but that is only true when x is in radians. Lines 2490-2520 convert the quadrant-fraction to radians by multiplying by pi/2, and exits.

Lines 2310-2470 handle larger angles by computing x*P/Q, where P and Q are polynomials in x^2. The constants for P and Q are given in lines 1420-1550, and come from the Hart book. [ I should mention here that I wrote those constants with pretty periods separating groups of five digits. This will not assemble in some older versions of the S-C Macro Assembler. If you get a syntax error, just leave out the periods. ]

Turning the Tables:

ATN is hardest to compute. First we have to deal with the two variants of calls, having one or two arguments. While all the previous function programs were called with the argument already in DAC, DP.ATN is called immediately after parsing the ATN-token. Lines 2960-3070 parse and process the following parentheses and whatever is between them.

Lines 2960-2970 require an opening parenthesis. Line 3070 requires the closing parenthesis. In between we expect one expression, or two expressions separated by a comma. If there is only one, we fake a second one (= 1.0).

What are the two arguments? Looking at a cartesian system, with the vector shown below, the arguments are (Y,X). If you call with one argument, it is (Y/X).

```
                y
                |
                |   .
                |
              --------x
                |
```

By using two separate arguments, rather than just the ratio, we can tell which of the four quadrants the vector was in. DP.ATAN will return a value between -pi and +pi, depending on the two signs. If you specify only the ratio, DP.ATAN will return a value between 0 and +pi depending on the sign.

Lines 3120-3160 save the two signs in bits 6 and 7 of UV.SIGN. Way at the end, lines 4100 and following, UV.SIGN determines the final value. If the sign of the denominator (X-vector) was negative, the composite vector is in the 2nd or 3rd quadrant: computing pi - angle gives a result between pi/2 and pi.

If the numerator (Y-vector) was negative, the composite vector is in the 3rd or 4th quadrant. Flipping the sign gives a result between 0 and -pi.

Lines 3180-3220 check for special cases of Y=0 or X=0. If the first argument (Y-vector) is zero, the angle is 0 or pi depending on the sign of the second argument. If the second argument (X-vector) is zero, the angle is either +pi/2 or -pi/2, depending on the sign of the first argument. What if both arguments are zero? That should produce an error message, but I am overlooking it: I will return an angle of 0 in this case.

If neither argument is zero, some special checks are made to see if the value of the ratio is very small or very large. I check before actually dividing, so the divide routine won't kick out on an overflow error. If the ratio would be greater than 10^20, I return a value of pi/2. This is accurate within the precision of DP18. On the other hand, if the ratio is smaller than 10^-63 I return 0. If neither extreme is true, I go ahead an divide to get the actual ratio. Then I check for an extremely small ratio, in which case atan(x)=x.

If we find our way down to line 3390, the ratio is between 10^-10 and 10^20. That is still too large a range for comfort, so we apply the fact that atan(1/x) = atan(pi/2 - x). If the ratio of Y/X is greater than 1.0, then we take the reciprocal and remember that we did so. This in effect folds the range at pi/4. The resulting argument range is between 10^-10 and 1. The variable N holds either 0 or 2 as a flag: 0 if we were already under 1, 2 if we formed the reciprocal.

The shape of the curve of the arctangent function between 0 and 1 (an angle between 0 and pi/4) is deceptive. It looks nice and easy, but a polynomial over that range with 20 digits of precision is much too long. We can easily reduce the range still further by applying another identity. If the reduced argument is now already below tan(pi/12), fine. If not, calculating (x*sqr(3)-1) / (sqr(3)+x) will bring it into that range. If we have to apply that formula, N will be incremented (making it 1 or 3).

The curve between 0 and tan(pi/12) looks almost like a straight line to the naked eye, but it really is far from straight. It takes a ratio of the form P/xQ where P and Q are polynomials in x^2. The coefficients are given in lines 2650-2770, again from Hart. The ratio is computed in lines 3800-3960.

Lines 3970-4080 start the unfolding process. The variable N is either 0, 1, 2, or 3 by this time. If N is 0, no folding was done. If N is 1, only folding above pi/12 was done. If N is 2, only folding above pi/4 was done. If N is 3, both folds were done. These lines convert the angle back to the correct value, using a table of addends and an optional sign flip:

| N | unfolding formula |
|---|---|
| 0 | none |
| 1 | pi/6 + x |
| 2 | pi/2 - x |
| 3 | pi/2 - ( pi/6 + x) = pi/3 - x |

That's it! We already discussed the code beyond line 4100, which figures out which quadrant the angle is in.

Any questions?

==============================================================================
DOCUMENT :AAL-8412:Articles:Front.Page.txt
==============================================================================

Volume 5 -- Issue 3 December, 1984

In This Issue...



New Source for 65802's

I talked to Constantine Geromnimon at Alliance Computers this morning. His company has ordered hundreds of 65802's, and offers them to you at $49.95 each. They expect their next shipment to come in around the middle of January, so now is the time to order. Call them at (718) 672-0684, or write to P. O. Box 408, Corona, NY 11368.

EPROM Programmer

A new EPROM Programmer, called the PROmGRAMER, is out from SCRG (the makers of quikLoader). This one burns anything from 2716's up to 27256's, and retails at $149.50. We'll sell 'em to you for a nice round $140. The software comes on disk, with instructions for loading it into EPROM for the quikLoader card.

Tom Weishaar Writes Again!

If you are among the throng who mourn the passing of Softalk, and particularly of the many informative columns such as DOStalk by Tom Weishaar, you will be as glad as I am that Tom has started publishing his own monthly newsletter.

Called "Open-Apple", you can subscribe for $24. In an unprecedented move toward international goodwill and the wholesome exchange of information, Tom has set the price the same for everyone, everywhere. We promptly sent him a check. If you love your Apple, do likewise. Send to Open-Apple, 10026 Roe, Overland Park, Kansas 66207. If you are cautious, send no money; Tom will bill you with the first issue, and you can cancel if you lose interest.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3

for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8412:Articles:Funny.DivBy7.txt
==============================================================================

Strange Way to Divide by 7.................Bob Sander-Cederlof

Division by seven is a necessary step for hi-res plotting routines. The quotient is the byte index on a given scan line. The remainder gives the bit position within that byte.

The hi-res code inside the Applesoft ROMs uses a subtraction loop to divide by seven, which can loop up to 36 times at 7 cycles per loop. This is a maximum of over 250 cycles, which is why super-fast hi-res usually uses lookup tables for the quotient and remainder.

I stumbled on a faster way of dividing any value up to 255 by seven. This is not directly usable by standard hi-res, because the x-coordinate can be as large as 279. My trick also does not give the remainder, just the quotient.

Here is the program, along with a test routine which tries every value from 0 to $FF, printing the quotient. The output from the test program is also shown, and you can see that the quotient is correct in every case. Can you explain why it works?

[ Hint: 1/7 = 1/8 + 1/64 + 1/512 + 1/4096 + ... ]

<<<<program>>>>

It is possible to divide by 3 or 15 using a program based on the same principle as the divide-by-seven above. Here is the code for those.

>>>>listings>>>>>, side by side>>>>>

Using the divide by 15, you could make a divide by ten. First multiply the original number by three (by shifting one bit left and adding), then divide by 15 using the above program, and then by 2 (by shifting one bit right). Since 3X/30 = X/10, there you have it.

==============================================================================
DOCUMENT :AAL-8412:Articles:Hex.To.Dec.txt
==============================================================================

Sly Hex Conversion.........................Bob Sander-Cederlof

Have you ever wondered what would happen if you added, in the 6502 decimal mode,values that were not decimal? I have. I also wondered if any of the results might be useful.

For example, what happens if I add 0 to $0A, in decimal mode? The following little piece of code will tell me:

```
       CLC
       SED             set decimal mode
       LDA #$0A
       ADC #0
       CLD             clear decimal mode
       JMP $FDDA       monitor print byte routine
```

Lo! The $0A turns into $10! It makes sense, because of course adding zero does not change anything. But the automatic "decimal adjust" that occurs after the add when the 6502 is in decimal mode detects the "A" nybble, generates a carry to the next nybble, and subtracts $0A.

It turns out the same process turns $0B into $11, $0C into $12, and so on up to $0F into $15.

That is a useful result! That means that I can convert a hex nybble to BCD byte by merely adding zero when in decimal mode!

A little further experimentation will lead to another useful trick. If I add first $90 and then $40, both additions in decimal mode, a value between $00 and $0F will be converted to the ASCII code for the digits 0-9 and letter A-F. Believe it or not!

The first addition, of $90, gives us $90-$9F. The automatic "decimal adjust" does nothing to $90-$99, and carry will be clear afterwards. If the intermediate result was $9A-$9F, the decimal adjust will first generate a nybble carry because the A-F nybble is greater than 9, and reduce that nybble by A. The nybble carry will increment the 9 nybble to A, which gets reduced back to 0 and a byte carry is set. This means we end up with $90-$99 with carry clear or $00-$05 with carry set.

Adding $40 in the next step brings the $90-$99 up to $30-$39 (with carry out of the byte, which we will ignore). The $00-$05 will be brought up to $41-$45, ASCII codes for A-F. Voila!

Useful, but maybe not the best. It turns out that a more traditional approach is only one byte longer and saves a few cycles. With the value $00-$0F in the A-register:

```
       CMP #$0A
       BCC .1          0-9
       ADC #6          convert A-F to $11-16
   .1  ADC #$30
```

will convert to ASCII.

```
==============================================================================
DOCUMENT :AAL-8412:Articles:HiresTableMaker.txt
==============================================================================
```

Generating Tables for Faster Hi-Res........Bob Sander-Cederlof

Look on page A23 in the Apple Supplement in the back of the December 1984 issue of Byte for an excellent article for the hi-res graphics buff: "Preshift-Table Graphics on Your Apple", by Bill Budge, Gregg Williams, and Rob Moore.

The article presents another of Bill Budge's secrets for fast animation using block graphics. If you want to move a block a few dots left or right, it is time-consuming to shift the 7-bits-in-8 dot images. Older techniques stored pre-shifted sets for each image that might be moved. The neater method described in this article stores a 14x256 byte table of all possible shifts of all possible bytes, and uses a fast lookup technique. I am not going to repeat all that here ... get the article.

The article also included some sample programs that used two other tables: a 192 entry address table for the addresses of each hi-res line, and a 280 entry table for the quotient and remainder of each horizontal position. Both of these tables were originally generated by Applesoft programs, and BSAVEd. The example program BLOADed them.

It dawned on me that a machine language program to generate those two tables would take less than half a page of code and be considerably faster than BLOADing pre-generated tables. Furthermore, once the tables were generated, the half-page of code could be overlaid with other programs or data. In a commercial product, this could cut down the boot time significantly.

First I wrote a program to generate the 192 addresses. This was almost a hand-compilation of the Applesoft program in the Byte article, but not quite. (I wrote the comments in near- Basic, as you can see.)

Then I merged into that program the stuff to generate the first 192 quotients and remainders. This is the horizontal dot position divided by 7 (7 dots per byte) to give the byte position on the line and the bit position in that byte.

After the 192 trips through that code, I added a loop to generate the rest of the Q/R pairs, from dot position 192 up to 279.

I timed the program by running it 250 times. All 250 took roughly 3 seconds, which means building the tables once takes about 12 milliseconds. Compare that to loading them from disk, which would take at least a half second.

I haven't tried it yet, but I think the preshift tables which were the meat of the Byte article could also be generated by a machine language program much quicker than BLOADing the same.  And since the program only needs to be used once, during initialization, it too could be burned after using.

=============================================================================
DOCUMENT :AAL-8412:Articles:IIe.Auxmem.LC.txt
=============================================================================

Using the //e or //c Auxiliary Memory "Language Card"
...............William M. Reed

From what I have seen in print, I had assumed that the portion of the 80 column "aux memory" that corresponds to the language card is (generally) unavailable for use. This assumption is only partly true.

In order to access this area ($D000-FFFF, banks 1 and 2) you must switch the ZERO PAGE at the same time. There seem to be no examples of this in the Apple manual, and very little advice.

The only "tricky" thing was to save the stack pointer (in the old manual supplement) before switching zero pages and language cards.

=====================================================================
DOCUMENT :AAL-8412:Articles:IIPlus.65C02.txt
=====================================================================

More Detail on Using 65C02's in old Apples......Andrew Jackson

In recent issues of AAL there have been several articles on the 65C02 and how to get it running in the Apple II+. I too was keen to get a 65C02 working in my machine, and had spent some time trying to get first a 1MHz part and then a 2MHz part to work.

William D. O'Ryan's letter in the June 84 AAL prompted me to try again and I am happy to report that the modification he described does work (replacing the LS257's at B6 and B7 with F257's). I wanted to find exactly why I could not simply substitute a 65C02 for a 6502, and so I spent some time looking at the circuit and specifications, using an oscilloscope to check my results.

The reasons that I eventually came up with are as follows. The Apple II circuit relies on various 'features' of the 6502 so that all the various parts of the Apple will work. The circuit diagram shows that the system timing is derived from o/0; the 6502 actually expects system timing to be derived from o/2. There is a slight delay between these two signals: on a 6502 it is about 50ns and on a 65C02 it is about 30ns. This difference in delays is what causes the problems when fitting a 65C02.

To simplify its circuit design the Apple uses a rather dirty trick when reading data from RAM memory. Normally when the 6502 reads data it expects the data on the bus to be valid 100ns before the end of o/2, and it latches the data into its internal registers when o/2 changes. The setup time allows the data bus to settle into a consistent state before being read. The Apple reduces the setup time to about 45 ns (worst case). This setup time would be ample for the 65C02 were it not for the shift between o/0 and o/2; this shift reduces the setup time to 25ns. A 2MHz 65C02 specifies a MINIMUM 40ns setup time; obviously there is a -15ns tolerance on the setup time, and hence the processor works erratically when timings fall into worst case conditions.

The tolerance is regained by substituting 74F257's for the two 74LS257's at board locations B6 and B7. These two chips multiplex the RAM data and the keyboard data; in doing so they add a delay of 30ns worst case to the data. By substituting F257's, the added delay is reduced to 5 ns; this changes the tolerance on the data setup time from -15ns to +10ns.

The Apple //e must use a slightly modified technique when reading data from RAM which explains why a 65C02 works in it without any modifications. I cannot check this as I do not have a //e circuit description. Anyway, it is probably all inside the MMU chip.

[ The 65816 specifications state a minimum read data setup time of 50ns, 10ns longer than the 65C02. One AAL reader has called us to report that the 65802 works wonderfully well in his old II+, even better than the original 6502. Some of you have wondered where to get the F257's: try Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002, phone (415) 592-8097. Their ad in Byte, Dec '84, page 349, says they have 74F257's at $1.79 each. (editor) ]

=============================================================================
DOCUMENT :AAL-8412:Articles:Little.Review.txt
=============================================================================

Gary Little's New Book, "Inside the Apple //e"

This is a useful book. The kind you want to keep, read, and constantly use as a reference. About 400 pages thick, 6x9, published by Brady Communications at $19.95.

Gary, a lawyer in Vancouver, has been serious about Apples since 1978 (almost as long as me). He's a long-time subscriber to AAL, Call APPLE, and other sources of the in-depth knowledge crammed into his book. He's also a programmer, with serious software on the market such as "Modem Magician". He knows what he's writing about, and writes it well.

A walk through the chapters may be the quickest way to get the measure of the book.

1--condensed history of Apple; intro. to binary, hex, and assembly language.

2--inside the 6502 itself: zero page, stack, registers, status, opcodes, address modes, I/O, interrupts, and the memory layout in the //e.

3--the Apple monitor: the commands explained, plus a table of the most useful subroutines in the monitor ROM.

4--Applesoft: memory map, tokenization, variable storage, integer and real numbers, the CHRGET subroutine, linking to assembly language programs, subroutines in ROM, and more.

5--DOS: internal structure, memory map, page 3 vectors, VTOC, catalog, track/sector lists, RWTS, and a read.sector program. ProDOS: memory map, page 3 vectors, volume bit map, directory, MLI, and a read.block program.

6--character input and the keyboard: RDKEY, 80-column firmware, RDCHAR, reading a line, changing input devices, encoding of keys, auto-repeat, type-ahead, all about RESET.

7--character and graphic output: too much to list here, all the way through double hi-res.

8--memory management: bank switching of ROM and RAM, auxiliary RAM, running co-resident programs.

9--speaker and cassette ports: music and voice.

10--game port: experiments, push button inputs, annunciators, strobe.

11--peripheral slots:  I/O memory locations, slot ROM, expansion ROM, scratchpad RAM, auxiliary slot, software protocols.

Many useful and interesting programs are listed in the book.  There is an optional diskette available (coupon bound in the book offers it for $20).  The diskette also includes a few bonus utility programs for use with DOS 3.3, including RAMDISK and DISK MAP.

Each chapter ends with a bibliography of related books, manuals, and articles.  (You'll find lots of references to AAL.)

If you grew along with Apple, as I did, you probably don't really need this book.  On the other hand, you will still enjoy it, and probably want it for you collection.  If you are relatively new, and having difficulty gathering all the information from past publications and scattered sources, you will want Gary's book too.

As you might suspect, we like the book so well we have decided to stock it.  You can get from us for $18 plus shipping (and tax where applicable).

```
=================================================================
DOCUMENT :AAL-8412:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 1.0......................................$80
S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Update...................................................$20
Source Code for Version 1.1 (on two disk sides).....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM....................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks...............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984    QD#17: Oct-Dec 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39

Quick-Trace (Anthro-Digital)...........CLOSEOUT SPECIAL!..(reg. $50)  $45/// $35

Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.......................$60
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)...........$2.25 each, or package of 20 for $40
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                          or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
quikLoader EPROM System (SCRG)...............................($179) $170
D MAnual Controller (SCRG)....................................($90)  $85
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32
PROMgrammer (SCRG)........................................($149.50) $140

Books, Books, Books..........................compare our discount prices!

    "Inside the Apple //e", Little.........................($19.95)  $18
    "Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95)  $18
    "Apple ][ Circuit Description", Gayler.................($22.95)  $21
```

```
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
```

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

```
*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
***                  (214) 324-2050                  ***
*** We accept Master Card, VISA and American Express ***
```

```
==============================================================================
DOCUMENT :AAL-8412:Articles:Overlap.Patches.txt
==============================================================================
```

A Solution to Overlapping DOS Patches.............Paul Lewis
Fairfax, Virginia

I have recently resolved a compatibility problem between two desirable sets of DOS 3.3 patches: the RAMdisk of the 192K Neptune extended memory card, and the DOS Dater that comes with Applied Engineering's Timemaster II. It seems they both want to put patches into the same "unused" spaces inside DOS.

After examining the two patches carefully, I found out which parts of the patches were overlapping. Being unable to find a truly unused area inside DOS, I used the technique on page 7.3 of "Beneath Apple DOS" of placing routines in the "safe" area between DOS and its buffers. This seems to work fine. [ Until you try to run some other program that does the same thing, like PLE... (editor) ]

The file DATER.OBJ0 contains the DOS.DATER patch that I use. I noticed that the patch could be placed anywhere, since there are no internal references. Using an Applesoft program (part of my HELLO), I move the DOS buffers down far enough to fit this code in, and then BLOAD the patches.

```
100 PRINT CHR$(4)"BRUN AUTO NEPTUNE"
110 PRINT "PSEUDO DISK INSTALLED"
120 POKE 40192,128 : REM Lower the buffers
130 PRINT CHR$(4)"MAXFILES 3"
140 PRINT "BUFFERS MOVED"
150 PRINT CHR$(4)"BLOAD DATER.OBJ0,A$9CD0"
160 POKE 45571,15 :REM Patch file name length
170 POKE 42883,14
180 POKE 44085,208 :REM Hook DOS to the DATER code
190 POKE 44086,156
200 PRINT "DOS DATER INSTALLED"
```

```
=============================================================================
DOCUMENT :AAL-8412:Articles:RememberingWhen.txt
=============================================================================
```

Remembering When...........................Bob Sander-Cederlof

There is a lot of grumbling going on, or at least so says the media. Supposedly Mac owners are MAD over Apple's $995 price tag for the 512K upgrade kit. And the fact that new buyers get a lower system price makes them even madder.

If it's true, then I guess the computer "for the rest of us" has found a market with a real-estate or Detroit mentality. Haven't they noticed that prices on virtually all electronic items go down every year? (I always say, "If houses and cars had gone the way electronics has over the last 30 years, we would now be able to buy a 3-bedroom home for two dollars and a nice car for 50 cents. Of course they would both fit on the head of a pin....")

I remember when I bought my Apple, with two rows of 4K RAM chips totalling 8K bytes. Adding another row of 4K chips would have cost me about $50. The price at that time for one set of 8 16K chips was $520. Through a special arrangement at Mostek, members of our local club were able to get them for $150. So to raise my Apple from 8K to 48K cost me $450. Retail price would have been $1560, plus tax.

Looking back even further, I found a letter from a Raymond Hoobler to the editor of the Journal of Dentistry, from October 1976. Ray owned an Apple 1, which was populated with 1K RAM chips. He was VERY happy with Apple's promise of an upgrade kit consisting of 4K RAM chips for ONLY $500!

It will not be too long before the price of 256K RAMs drops. Then we can start grumbling about the price of 4-megabyte upgrade kits. Or, we could rejoice at the blessings of ever improving technology, mass marketing, and understanding wives.

=============================================================================
DOCUMENT :AAL-8412:Articles:XMas.CloseOuts.txt
=============================================================================

Some Christmas Specials...................Bob Sander-Cederlof

We only have a limited quantity of some of the following items, and since they are out of the main stream of our business we probably will not be re-ordering them.

```
                       # in   Regular  Special
Item                   stock   price    price
---------------------------------------------
Quick-Trace             11    $49.95   $35

The DOS Enhancer         1    $69.95   $30

Add-on Libraries for The Routine Machine
  &Screen                1    $29      $15
  &Chart                 1    $29      $15
  &Array                 1    $29      $15

Micro On The Apple (book and disk)
  Volume 1               1    $24.95   $10
  Volume 2               2    $24.95   $10

Think Tank (II+ 40-col)  1    $120     $60

Good books we probably will not re-stock
  Z-80 Subroutines       3    $18.95   $12
 by Lance Leventhal

  68000 Assem Lang Prog  3    $18.95   $12
  by Lance Leventhal

  Data Base Mgmt Systems 3    $16.95   $10.50
  by David Kruglinski

  Programmer's CP/M Hbk  1    $21.95   $14

  User Guide to Unix     1    $17.95   $11

  Graphics Primer/IBM PC 2    $21.95   $14

  Visicalc Home/Office   1    $15.99   $9
```

```
=============================================================================
DOCUMENT :AAL-8412:DOS3.3:S.DP18.TRIG.txt
=============================================================================

 1000  *SAVE S.DP18 TRIG
 1010  *--------------------------------
 1020  AS.CHRGET   .EQ $B1
 1030  AS.CHRGOT   .EQ $B7
 1040  AS.CHKCLS   .EQ $DEBB
 1050  AS.CHKOPN   .EQ $DEB8
 1060  *--------------------------------
 1070  POLY.1      .EQ $FFFF
 1080  POLY.N      .EQ $FFFF
 1090  DADD        .EQ $FFFF
 1100  DSUB        .EQ $FFFF
 1110  DMULT       .EQ $FFFF
 1120  DDIV        .EQ $FFFF
 1130  DP.INT      .EQ $FFFF
 1140  DP.EXP      .EQ $FFFF
 1150  DP.TRUE     .EQ $FFFF
 1160  DP.FALSE    .EQ $FFFF
 1170  MOVE.DAC.ARG    .EQ $FFFF
 1180  MOVE.YA.ARG.1   .EQ $FFFF
 1190  MOVE.YA.DAC.1   .EQ $FFFF
 1200  SWAP.ARG.DAC    .EQ $FFFF
 1210  MOVE.DAC.TEMP1 .EQ $FFFF
 1220  MOVE.DAC.TEMP2 .EQ $FFFF
 1230  MOVE.DAC.TEMP3 .EQ $FFFF
 1240  MOVE.TEMP1.DAC .EQ $FFFF
 1250  MOVE.TEMP1.ARG .EQ $FFFF
 1260  MOVE.TEMP2.ARG .EQ $FFFF
 1270  MOVE.TEMP3.ARG .EQ $FFFF
 1280  PUSH.DAC.STACK .EQ $FFFF
 1290  POP.STACK.ARG  .EQ $FFFF
 1300  *--------------------------------
 1310  DAC.EXPONENT    .BS 1
 1320  DAC.HI          .BS 10
 1330  DAC.SIGN        .BS 1
 1340  *--------------------------------
 1350  ARG.EXPONENT    .BS 1
 1360  ARG.HI          .BS 10
 1370  ARG.SIGN        .BS 1
 1380  *--------------------------------
 1390  N           .BS 1
 1400  UV.SIGN     .BS 1
 1410  *--------------------------------
 1420  P.SIN       .EQ *
 1430  P.SIN.N     .EQ 6    P6*X^6 + P5*X^5 + ... + P1*X + P0
 1440              .HS 3C.50312.63884.64664.12845   P6
 1450              .HS BE.82818.08039.29577.39110   P5
 1460              .HS 40.62919.63490.93113.55230   P4
 1470              .HS C2.25642.44036.60338.57070   P3
 1480              .HS 43.53892.64053.57788.76289   P2
```

```
1490            .HS C4.49326.67470.47152.36677   P1
1500            .HS 45.12596.16380.91365.41816   P0
1510  *--------------------------------
1520  Q.SIN      .EQ *
1530  Q.SIN.N    .EQ 2    X^2 + Q1*X + Q0
1540            .HS 43.15743.43316.33194.13935   Q1
1550            .HS 44.80189.66936.87727.15787   Q0
1560  *--------------------------------
1570  CON.ONE    .HS 41.10000.00000.00000.00000
1580  CON.TWO    .HS 41.20000.00000.00000.00000
1590  CON.2PI    .HS 41.62831.85307.17958.64769
1600  CON.PI.2   .HS 41.15707.96326.79489.66192
1610  CON.PI     .HS 41.31415.92653.58979.32385
1620  CON.1..2PI .HS 40.15915.49430.91895.33577   1/2PI
1630  *--------------------------------
1640  *      COS (DAC)
1650  *--------------------------------
1660  DP.COS LDA #CON.PI.2     PI/2
1670         LDY /CON.PI.2
1680         JSR MOVE.YA.ARG.1 COS(X) = SIN(X+PI/2)
1690         LDA #0            GET ABS(DAC) TO FORCE
1700         STA DAC.SIGN      ...COS(-X)=COS(X)
1710         JSR DADD
1720  *--------------------------------
1730  *      SIN (DAC)
1740  *      #3371
1750  *--------------------------------
1760  DP.SIN
1770  *---IF X VERY SMALL...-----------
1780         LDA DAC.EXPONENT
1790         CMP #$40-10
1800         BCS .1            NOT VERY SMALL
1810         RTS               VERY SMALL, SIN(X)=X
1820  *---ADJUST FOR SIGN OF X---------
1830  .1     LDA DAC.SIGN      SIN(-X) = - SIN(X)
1840         PHA               ...SO SAVE SIGN OF X
1850         LDA #0            ...AND MAKE X POSITIVE
1860         STA DAC.SIGN
1870  *---X*(1/2PI)-------------------
1880         LDA #CON.1..2PI
1890         LDY /CON.1..2PI
1900         JSR MOVE.YA.ARG.1
1910         JSR DMULT
1920  *---GET FRACTIONAL PART----------
1930         JSR MOVE.DAC.ARG
1940         JSR DP.INT
1950         JSR DSUB
1960  *---FOLD QUADRANTS INTO ONE------
1970         JSR MOVE.DAC.ARG  MULTIPLY BY FOUR
1980         JSR DADD          BY DOUBLING TWICE
1990         JSR MOVE.DAC.ARG
2000         JSR DADD          0 <= DAC < 4
2010         LDA DAC.EXPONENT  IS DAC < 1?
2020         CMP #$41
```

```
       BCC .4            ...YES, IT IS IN 1ST QUADRANT
*---2ND, 3RD, OR 4TH------------
       LDA DAC.HI
       CMP #$20          IS DAC < 2.0?
       BCC .3            ...YES, 1ST OR 2ND QUADRANT
*---FOLD 3RD-4TH OVER 1ST-2ND----
       PLA               ...NO, FLIP SIGN FOR
       EOR #$80                 3RD OR 4TH QUADRANTS
       PHA
       LDA #CON.TWO      FOLD 3RD & 4TH OVER 1ST & 2ND
       LDY /CON.TWO
       JSR MOVE.YA.ARG.1
       JSR SWAP.ARG.DAC
       JSR DSUB
       LDA DAC.EXPONENT
       CMP #$41
       BCC .4            ...ALREADY IN 1ST
*---FOLD 2ND OVER 1ST------------
.3     LDA #CON.TWO      LET X=2-X
       LDY /CON.TWO
       JSR MOVE.YA.ARG.1
       JSR DSUB
*---ANGLE NOW IN 1ST QUADRANT----
.4     PLA               PUT FINAL SIGN ON X
       STA DAC.SIGN
       LDA DAC.EXPONENT  CHECK FOR VERY SMALL
       CMP #$40-9
       BCC .5            ...YES, SIN(X)=X*PI/2
       JSR MOVE.DAC.ARG  PREPARE FOR POLYNOMIALS
       JSR MOVE.DAC.TEMP1      X IN TEMP1
       JSR DMULT               X*X IN TEMP2
       JSR MOVE.DAC.TEMP2
       LDA #P.SIN
       LDY /P.SIN
       LDX #P.SIN.N
       JSR POLY.N
       JSR MOVE.DAC.TEMP3
       LDA #Q.SIN
       LDY /Q.SIN
       LDX #Q.SIN.N
       JSR POLY.1
       JSR MOVE.TEMP3.ARG
       JSR DDIV               P/Q
       JSR MOVE.TEMP1.ARG     XP/Q
       JMP DMULT
*-------------------------------
.5     LDA #CON.PI.2     FOR VERY SMALL X
       LDY /CON.PI.2     SIN(2X/PI) = X*PI/2
       JSR MOVE.YA.ARG.1
       JMP DMULT
*-------------------------------
*      TAN (DAC) = SIN(DAC) / COS(DAC)
*-------------------------------
DP.TAN JSR PUSH.DAC.STACK  SAVE ANGLE
```

```
2570        JSR DP.SIN          TAN=SIN/COS
2580        JSR POP.STACK.ARG   GET ANGLE
2590        JSR PUSH.DAC.STACK  SAVE SIN
2600        JSR SWAP.ARG.DAC
2610        JSR DP.COS          GET COSINE
2620        JSR POP.STACK.ARG   GET SIN
2630        JMP DDIV            SIN/COS
2640 *--------------------------------
2650 P.ATN      .EQ *    HART # 5505
2660 P.ATN.N    .EQ 3    P3*X^3 + P2*X^2 + P1*X + P0
2670            .HS 42.12595.80226.30295.47240   P3
2680            .HS 43.12557.91664.37980.65520   P2
2690            .HS 43.29892.80380.69396.22448   P1
2700            .HS 43.19720.30956.84935.02854   P0
2710 *--------------------------------
2720 Q.ATN      .EQ *
2730 Q.ATN.N    .EQ 4    X^4 + Q3X^3 + Q2X^2 + Q1X + Q0
2740            .HS 42.37066.08632.20190.23801   Q3
2750            .HS 43.20769.26817.33604.63361   Q2
2760            .HS 43.36466.24032.97707.76242   Q1
2770            .HS 43.19720.30956.84935.02861   Q0
2780 *--------------------------------
2790 ATN.TBL.H
2800        .DA /CON.PI.6
2810        .DA /CON.PI.2
2820        .DA /CON.PI.3
2830 ATN.TBL.L
2840        .DA #CON.PI.6
2850        .DA #CON.PI.2
2860        .DA #CON.PI.3
2870 *--------------------------------
2880 CON.TAN.PI.12 .HS 40.26794.91924.31122.70647
2890 CON.PI.6   .HS 40.52359.87755.98298.87308
2900 CON.PI.3   .HS 41.10471.97551.19659.77462
2910 CON.SQR.3  .HS 41.17320.50807.56887.72935
2920 *--------------------------------
2930 *      ATN FUNCTION
2940 *      1 OR 2 ARGUMENTS
2950 *--------------------------------
2960 DP.ATN JSR AS.CHRGET
2970        JSR AS.CHKOPN CHECK FOR (
2980        JSR DP.EXP   GET EXPRESSION
2990        JSR PUSH.DAC.STACK
3000        JSR DP.TRUE  IN CASE 1 ARGUMENT
3010        JSR AS.CHRGOT
3020        CMP #',      TWO-ARG?
3030        BNE .1       NO
3040        JSR AS.CHRGET GOBBLE ,
3050        JSR DP.EXP   YES,GET OTHER ONE
3060 .1     JSR POP.STACK.ARG GET 1ST ARG BACK
3070        JSR AS.CHKCLS     REQUIRE ")"
3080 *--------------------------------
3090 *      ATN (ARG,DAC)  ARG/DAC
3100 *--------------------------------
```

```
3110  DP.ATAN
3120         LDA DAC.SIGN      SAVE BOTH SIGNS
3130         ASL               SIGN OF DENOMINATOR
3140         LDA ARG.SIGN      SIGN OF NUMERATOR
3150         ROR               BIT 7 = DENOM SIGN
3160         STA UV.SIGN       BIT 6 = NUMER SIGN
3170  *---CHECK FOR BOUNDARIES---------
3180         LDA DAC.EXPONENT  CHECK DENOMINATOR
3190         BEQ .1            ...V/0, SO RETURN PI/2
3200         SEC
3210         LDA ARG.EXPONENT
3220         BEQ .12           ...0/U, SO RETURN 0
3230         SBC DAC.EXPONENT
3240         BMI .13
3250         CMP #20           IF >10^20, RETURN PI/2
3260         BCC .11           ...NOT >10^20
3270  .1     LDA #CON.PI.2     V/0 OR OVERFLOW
3280         LDY /CON.PI.2     SO RETURN PI/2
3290         JSR MOVE.YA.DAC.1
3300         JMP DP.ATN.C
3310  .13    CMP #-63          IF <10^-63, RETURN 0
3320         BCS .11
3330  .12    JSR DP.FALSE      RETURN 0
3340  .14    JMP DP.ATN.B
3350  .11    JSR DDIV          CALCULATE V/U
3360         LDA DAC.EXPONENT
3370         CMP #$40-10       IF X VERY SMALL, ATAN(X)=X
3380         BCC .14           ...VERY SMALL INDEED!
3390  *---FOLD AT PI/4-----------------
3400         LDA #0            GET ABS(X), BECAUSE
3410         STA DAC.SIGN        SIGNS ALREADY REMEMBERED
3420         STA N
3430         LDA DAC.EXPONENT  IS X<1?
3440         CMP #$41
3450         BCC .3            ...YES, X<1
3460         LDA #CON.ONE      FORM RECIPROCAL
3470         LDY /CON.ONE
3480         JSR MOVE.YA.ARG.1
3490         JSR DDIV          1/X
3500         LDA #2            AND REMEMBER WE DID IT
3510         STA N
3520  *---FOLD AT PI/12----------------
3530  .3     JSR MOVE.DAC.TEMP1 SAVE X
3540         LDA #CON.TAN.PI.12 TAN(PI/12)
3550         LDY /CON.TAN.PI.12
3560         JSR MOVE.YA.ARG.1
3570         JSR DSUB           IS X>TAN(PI/12)?
3580         LDA DAC.SIGN
3590         PHA
3600         JSR MOVE.TEMP1.DAC  RESTORE X
3610         PLA
3620         BPL .4             ...NO, WE DON'T HAVE TO FOLD
3630         INC N              ...YES, SO FORM
3640         LDA #CON.SQR.3     (X*SQR(3)-1) / (SQR(3)+X)
```

```
3650        LDY /CON.SQR.3
3660        JSR MOVE.YA.ARG.1
3670        JSR DMULT          X*SQR(3)
3680        JSR MOVE.DAC.ARG
3690        JSR DP.TRUE
3700        JSR DSUB           X*SQR(3)-1
3710        JSR MOVE.DAC.TEMP2 SAVE IT
3720        JSR MOVE.TEMP1.ARG GET X
3730        LDA #CON.SQR.3
3740        LDY /CON.SQR.3
3750        JSR MOVE.YA.DAC.1
3760        JSR DADD           SQR(3)+X
3770        JSR MOVE.TEMP2.ARG
3780        JSR DDIV           THE ANSWER
3790        JSR MOVE.DAC.TEMP1 SAVE FOLDED-UP X
3800  *---ATAN(0...PI/12)-------------
3810  .4     JSR MOVE.DAC.ARG
3820        JSR DMULT          X^2
3830        JSR MOVE.DAC.TEMP2 SAVE X^2
3840        LDA #P.ATN
3850        LDY /P.ATN
3860        LDX #P.ATN.N
3870        JSR POLY.N
3880        JSR MOVE.DAC.TEMP3
3890        LDA #Q.ATN
3900        LDY /Q.ATN
3910        LDX #Q.ATN.N
3920        JSR POLY.1
3930        JSR MOVE.TEMP3.ARG GET P
3940        JSR DDIV           P/Q
3950        JSR MOVE.TEMP1.ARG GET X
3960        JSR DMULT          P(X^2)/Q(X^2)*X
3970  *---UNFOLD FROM PI/12, PI/4------
3980        LDX N             0, 1, 2, OR 3
3990        BEQ DP.ATN.B      ...NO ADDEND
4000        DEX               0, 1, OR 2
4010        BEQ .5            ...NO COMPLEMENT
4020        LDA DAC.SIGN      ATAN(1/X)=ATAN(PI/2 - X)
4030        EOR #$80
4040        STA DAC.SIGN
4050  .5     LDA ATN.TBL.L,X   GET A(N)
4060        LDY ATN.TBL.H,X
4070        JSR MOVE.YA.ARG.1
4080        JSR DADD          X + A(N)
4090  *---UNFOLD INTO QUADRANTS--------
4100  DP.ATN.B
4110        BIT UV.SIGN       TEST SIGN OF DENOMINATOR
4120        BPL DP.ATN.C      ...POSITIVE, 1ST OR 4TH
4130        LDA #CON.PI       ...NEGATIVE, 2ND OR 3RD
4140        LDY /CON.PI          SO DO PI-X
4150        JSR MOVE.YA.ARG.1
4160        JSR DSUB
4170  *--------------------------------
4180  DP.ATN.C
```

```
4190        BIT UV.SIGN      TEST SIGN OF NUMERATOR
4200        BVC .6           ...POSITIVE, 1ST OR 2ND
4210        LDA DAC.SIGN     ...NEGATIVE, 3RD OR 4TH
4220        EOR #$80         -X
4230        STA DAC.SIGN
4240 .6     RTS
4250 *--------------------------------
```

=============================================================================
DOCUMENT :AAL-8412:DOS3.3:S.Funny.Divby15.txt
=============================================================================

```
 1000  *SAVE S.FUNNY DIVIDE BY FIFTEEN
 1010  *--------------------------------
 1020  BYTE   .EQ 0
 1030  *--------------------------------
 1040  T      LDA #0
 1050         STA BYTE
 1060  .2     LDX #15
 1110  .1     JSR DIVIDE.BY.FIFTEEN
 1120         JSR $FDDA
 1130         INC BYTE
 1140         BEQ .3
 1150         DEX
 1160         BNE .1
 1170         JSR $FD8E
 1180         JMP .2
 1190  .3     RTS
 1200  *--------------------------------
 1210  DIVIDE.BY.FIFTEEN
 1220         LDA BYTE
 1230         LSR
 1240         LSR
 1250         LSR
 1260         LSR
 1270         ADC BYTE
 1280         ROR
 1290         LSR
 1300         LSR
 1310         LSR
 1320         ADC BYTE
 1330         ROR
 1340         LSR
 1350         LSR
 1360         LSR
 1370         RTS
 1380  *--------------------------------
```

```
=========================================================================
DOCUMENT :AAL-8412:DOS3.3:S.FunnyDivby3.txt
=========================================================================

 1000  *SAVE S.FUNNY DIVIDE BY THREE
 1010  *--------------------------------
 1020  BYTE   .EQ 0
 1030  *--------------------------------
 1040  T      LDA #0
 1050         STA BYTE
 1060  .2     LDX #15
 1070  .1     CPX #7
 1080         BNE .4
 1090         LDA #$A0
 1100         JSR $FDED
 1110  .4     JSR DIVIDE.BY.THREE
 1120         JSR $FDDA
 1130         INC BYTE
 1140         BEQ .3
 1150         DEX
 1160         BNE .1
 1170         JSR $FD8E
 1180         JMP .2
 1190  .3     RTS
 1200  *--------------------------------
 1210  DIVIDE.BY.THREE
 1220         LDA BYTE
 1230         LSR
 1240         LSR
 1250         ADC BYTE
 1260         ROR
 1270         LSR
 1280         ADC BYTE
 1290         ROR
 1300         LSR
 1310         ADC BYTE
 1320         ROR
 1330         LSR
 1340         ADC BYTE
 1350         ROR
 1360         LSR
 1370         RTS
```

=============================================================================
DOCUMENT :AAL-8412:DOS3.3:S.FunnyDivby7.txt
=============================================================================

```
 1000  *SAVE S.FUNNY DIVIDE BY SEVEN
 1010  *--------------------------------
 1020  BYTE   .EQ 0
 1030  *--------------------------------
 1040  T      LDA #0
 1050         STA BYTE
 1060  .2     LDX #14
 1070  .1     CPX #7
 1080         BNE .4
 1090         LDA #$A0
 1100         JSR $FDED
 1110  .4     JSR DIVIDE.BY.SEVEN
 1120         JSR $FDDA
 1130         INC BYTE
 1140         BEQ .3
 1150         DEX
 1160         BNE .1
 1170         JSR $FD8E
 1180         JMP .2
 1190  .3     RTS
 1200  *--------------------------------
 1210  DIVIDE.BY.SEVEN
 1220         LDA BYTE
 1230         LSR
 1240         LSR
 1250         LSR
 1260         ADC BYTE
 1270         ROR
 1280         LSR
 1290         LSR
 1300         ADC BYTE
 1310         ROR
 1320         LSR
 1330         LSR
 1340         RTS
 1350  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8412:DOS3.3:S.HEX.TO.DEC.txt
==============================================================================

```
 1000  *SAVE S.HEX TO DEC
 1010  T      LDX #0
 1020  .1     TXA
 1030         JSR $FDDA
 1040         LDA #"-"
 1050         JSR $FDED
 1060         TXA
 1070  *--------------------------------
 1080         SED
 1090         CLC
 1110         ADC #0
 1120         CLD
 1130  *--------------------------------
 1140         JSR $FDDA
 1150         LDA #"-"
 1160         JSR $FDED
 1170         TXA
 1180  *--------------------------------
 1190         SED
 1200         CLC
 1210         ADC #$90
 1220         ADC #$40
 1230         CLD
 1240  *--------------------------------
 1250         JSR $FDDA
 1260         LDA #"-"
 1270         JSR $FDED
 1280         TXA
 1290  *--------------------------------
 1300         CMP #10
 1310         BCC .2
 1320         ADC #6
 1330  .2     ADC #$30
 1340  *--------------------------------
 1350         JSR $FDDA
 1360  *--------------------------------
 1370         JSR $FD8E
 1380         INX
 1390         CPX #16
 1400         BCC .1
 1410         RTS
```

```
=====================================================================
DOCUMENT :AAL-8412:DOS3.3:S.MakeHiresAddr.txt
=====================================================================

 1000  *SAVE S.MAKE HIRES ADDRS
 1010  *--------------------------------
 1020  I       .EQ 0
 1030  JL      .EQ 1
 1040  JH      .EQ 2
 1050  K       .EQ 3
 1060  Q       .EQ 4
 1070  R       .EQ 5
 1080  *--------------------------------
 1090  ADDRL   .EQ $900
 1100  ADDRH   .EQ $9C0
 1110  QUO.1   .EQ $A80
 1120  QUO.2   .EQ QUO.1+192
 1130  REM.1   .EQ QUO.1+280
 1140  REM.2   .EQ REM.1+192
 1150  *--------------------------------
 1160  BUILD   LDX #0        FOR X = 0 TO 191 STEP 1
 1170          STX I         FOR I = 0 TO $50 STEP $28
 1180          STX JL        FOR J = 0 TO $0380 STEP $0080
 1190          STX JH
 1200          STX K         FOR K = 0 TO $1C STEP $04
 1210          STX Q         QUOTIENT = 0
 1220          STX R         REMAINDER = 0
 1230  *---BUILD NEXT HI-RES ADDR-------
 1240  .1      LDA I
 1250          ORA JL
 1260          STA ADDRL,X
 1270          LDA #$20
 1280          ORA JH
 1290          ORA K
 1300          STA ADDRH,X
 1310  *---SAVE NEXT Q/R PAIR-----------
 1320          LDA Q
 1330          STA QUO.1,X
 1340          LDA R
 1350          STA REM.1,X
 1360  *---NEXT K----------------------
 1370          CLC
 1380          LDA K
 1390          ADC #4
 1400          STA K
 1410          EOR #$20
 1420          BNE .2
 1430  *---NEXT J----------------------
 1440          STA K
 1450          LDA JL
 1460          EOR #$80
 1470          STA JL
 1480          BNE .2
```

```
1490        INC JH
1500        LDA JH
1510        EOR #4
1520        BNE .2
1530  *---NEXT I----------------------
1540        STA JH
1550        CLC
1560        LDA I
1570        ADC #$28
1580        STA I
1590  *---BUMP Q/R PAIR----------------
1600  .2    INC R        R COUNTS 0...6
1610        LDA R
1620        EOR #7       IF R=7, MAKE 0 AND BUMP Q
1630        BNE .3       ...NOT 7 YET
1640        STA R        ...R=7, SO MAKE IT 0
1650        INC Q             AND BUMP Q
1660  *---NEXT X----------------------
1670  .3    INX
1680        CPX #192
1690        BCC .1
1700  *---NOW FINISH Q/R PAIRS---------
1710  *---BETWEEN 192 AND 279----------
1720        LDX #0       FOR X = 0 TO 280-192-1
1730  .4    LDA Q
1740        STA QUO.2,X
1750        LDA R
1760        STA REM.2,X
1770  *---BUMP Q/R PAIR AS BEFORE------
1780        INC R
1790        LDA R
1800        EOR #7
1810        BNE .5
1820        STA R
1830        INC Q
1840  *---NEXT X----------------------
1850  .5    INX
1860        CPX #280-192
1870        BCC .4
1880        RTS
1890  *--------------------------------
1900        .LIST OFF
```

```
==============================================================================
DOCUMENT :AAL-8412:DOS3.3:S.Time.MVN.txt
==============================================================================
```

```
 1000  *SAVE S.TIME MVN
 1010         .OP 65816
 1020         .OR $300
 1030  *---------------------------------
 1040  CNTR   .EQ 0 AND 1
 1050  *---------------------------------
 1060  MVN.TIMER
 1070         CLC          65816 MODE
 1080         XCE
 1090         REP #$30     16-BIT MODE
 1100  *---------------------------------
 1110         LDA ##1000
 1120         STA CNTR
 1130  *---------------------------------
 1140  .1     LDX ##$3000  Source start address
 1150         LDY ##$4000  Destination start address
 1160         LDA ##9999   # Bytes - 1
 1170         MVN 0,0
 1180         DEC CNTR
 1190  *      BNE .1
 1200  *---------------------------------
 1210         SEC          RETURN TO 6502 MODE
 1220         XCE
 1230         RTS
```

```
=============================================================================
DOCUMENT :AAL-8501:Articles:DP18.Print.txt
=============================================================================
```

18-Digit Arithmetic, Part 9................Bob Sander-Cederlof

Nearing the home stretch, this month I will cover the DP18 PRINT statement. I believe that only leaves INPUT for next month.

Normal Applesoft PRINT has a wide variety of options. PRINT may appear all by itself to print a carriage return, or with one or more expressions. The expressions may be separated by commas or semicolons: both are used to separate the expressions for syntax purposes, but commas also cause a form of tabbing. A final comma or semicolon may be used to suppress the normal carriage return at the end of the printed line. All numeric values are printed in an unformatted style.

We wanted to have additional formatting capabilities in DP18 PRINT. Many users of Applesoft have tried to write money handling programs, agonizing over the contortions necessary to make pretty reports. BASIC on many other micros comes with PRINT USING, which includes a string describing the exact format to use for print a list of items. Applesoft doesn't have PRINT USING (we have graphics instead, and all in a 10K interpreter). DP18 does.

DP18 doesn't have everything though. Here are some things we left out. Commas may be used to separate items in a DP18 PRINT statement, but no tabbing happens. Instead, commas cause carriage returns. DP18 values are so long that comma tabbing seemed useless. You cannot fit two fully extended unformatted values in one 40-column line. Maybe you could say we do tab, all the way to the next line. Anyway, this gives us a useful NEW feature: the ability for one PRINT statement to print on more than one line.

DP18 PRINT can only print DP18 expressions. Normal Applesoft real or integer expressions can be printed by normal Applesoft PRINT, or by converting them to DP18 values using VAL and STR$. Applesoft string expressions can be printed using a DP18 "picture", but not in the simple manner you are used to in normal Applesoft PRINT.

DP18 in its present form supports three different kinds of items in a PRINT statement: DP18 expressions, #WD items, and $PIC items.

The first kind is the easiest to use, and will remind you a lot of Applesoft. Since all you tell DP18 is the expression, it makes up its own mind about the format to use. We call this "unformatted", because it hard to predict how it will look once it is printed. If the absolute value of the number to be printed is within the range from .01 to 999,999,999,999,999,999 (18 digits) it will print as a normal number, with no leading or trailing blanks and no trailing zeroes. If outside that range, it will be printed with an E exponent. Doesn't this remind you of Applesoft? Here are some examples using numbers (bear in mind they could be long complicated DP18 expressions):

```
]&DP:PRINT 1,2,3;4;5
1
2
345
]&DP:PRINT .009,.01,999999999999999999
9E-3
.01
999999999999999999
]&DP:PRINT 1000000000000000000
1E+18
]
```

If a PRINT list item begins with the character "#", it is a #WD formatted item. Three things follow the "#" character, separated by commas: a field width, the number of fractional digits, and a DP18 expression. (If you have ever used Fortran, this is going to remind you of the "Fw.d" format.)

```
&DP:PRINT #w,d,value
```

The w and d parameters are Applesoft expressions (or simple constants), and the value is a DP18 expression. The value will be printed right-justified in a field w-characters wide, with d decimal places after the decimal point. Leading blanks will be printed if there is room for any. If the number will not fit in w characters, w asterisks will be printed instead to show you there was an overflow problem. Values are rounded to the required number of decimal places, not just truncated. Here are some examples:

```
]&DP:PRINT #8,3,2.04;#8,3,5,#10,5,3.14159,#3,1,99
   2.040   5.000
   3.14159
***
]&DP:PRINT #8,4,3.14159,#7,3,3.14159,#6,2,3.14159
  3.1416
  3.142
  3.14

100 FOR I=0TO5
110 PRINT I;:&DP:PRINT #10-I,5-I,3.1415926
120 NEXT
]RUN
0   3.14159
1   3.1416
2   3.142
3   3.14
4   3.1
5   3.
```

The third type of PRINT item begins with a dollar sign. A string constant, variable, or expression follows the dollar sign. If the picture specifies fields for DP18 or string values to be printed in, then the list of values must follow the picture, all separated by commas.

```
      &DP:PRINT $ picture
      &DP:PRINT $ picture,list
```

The "picture" is any Applesoft string expression; it is used as the template for formatting the expressions in the optional list. The list may have any number of expressions separated by commas as long as they correspond with the picture. You may even have no expressions at all, which is why I say the list is optional.

The picture consists of a string of characters. There are four basic types of characters used in pictures: commands, literals, numbers, and field descriptions. These are described below.

Any number in the picture makes up a repeat count. The repeat count specifies how many times to repeat the following command or field-description character. If a command is not preceded by a repeat count, a 1 is assumed. Repeat counts may range anywhere from 1 to 255.

The commands which may be included in pictures give you control over the screen and cursor. Some of the commands allow a repeat count to be specified. In the following descriptions, "n" refers to the optional repeat count. If no repeat count is used, n=1.

```
/ -- Prints n carriage returns.

X -- Prints n spaces.

> -- Clear to from the cursor to the end of line.
     If the next picture character is also ">",
     clear from the cursor to the end of screen.

V -- Performs VTAB n, where n must be from 1 to 24.

H -- Performs HTAB n.  [As implemented now, this is
     probably not compatible with your printer or
     80-column cards.]
```

Literals are defined in strings using the apostrophe. Any text you want to print from inside the picture may be included between apostrophes. If you want to include an apostrophe inside a literal, put two apostrophes in a row. If you put a repeat count before the literal, it will be printed n times.

Now here are some examples using repeat counts, commands, and literals.

```
&DP:PRINT $ "VH>>"     (moves the cursor to the top left
                        corner, and clears the screen.)
```

```
]&DP:PRINT $ "'SEPARATE'/'LINES'"
SEPARATE
LINES
]&DP:PRINT $ "4V10H3'BANG! '"
starting at line 4 column 10 prints:
          BANG! BANG! BANG!
```

There are two kind of field descriptions: one for telling DP18 how to print numbers and the other for telling how to print strings. Since string descriptors are easier, let's start with them.

A string field descriptor tells DP18 how to print the value of an Applesoft string. There are three different characters used, which tell DP18 whether to left-justify, right-justify, or center the value of the string within the field. Since we are building a "picture", the width of the field is shown by using multiples of the controlling character. The three different controlling characters are:

A -- Print the string left justified in the field.

R -- Print the string right justified in the field.

C -- Print the string centered in the field.

The data to be printed comes from the list of data items which follows the picture. Here are some examples using string descriptors:

```
]A$="ABC"
]P$="AAAAAAA'-'RRRRRRR'-'CCCCCCC'-'"
]&DP:PRINT $ P$,A$,A$,A$
ABC    -    ABC-  ABC  -
aaaaaaa.rrrrrrr.ccccccc.

]PRINT $"RRRRR X 5A 'HI' 6C 'BYE'","AB","ABC","XY"
   AB ABC  HI  XY  BYE
rrrrrxaaaaa..cccccc...
```

If you mix the A, C, and R control letters in one string field descriptor, the controlling letter will be the last one in the field. If you want to have two fields adjacent to each other, you can separate the descriptors with a space. The space will not become part of the printed output. If a string value is too long to fit in a field, the field will be filled with asterisks instead of the actual data. When you see asterisks where you expected data, the data was too long.

```
]&DP:PRINT$"AAA AAA AAA","AN","EGG","ROLLS"
AN EGG***
```

Numeric field descriptors are made up of the characters listed below. The number to be printed is taken from the expression list. The expression corresponding to a numeric field descriptor MUST be a DP18 expression. If it is not a DP18 numeric expression, an error will result. If the number is too large for the field, asterisks will be printed. The number is rounded to the number of decimal places you specify in the descriptor before printing. Trailing zeroes after the decimal point are printed if necessary to fill up the field.

+ -- Reserves a place for the sign of the number. the sign will be printed in this position even if the number is positive. The sign may be placed anywhere within or at either end of the number.

- -- Also reserves a place for the sign, but the sign will only be printed if it is negative. If the number is positive, the fill character is printed instead.

(If neither + nor - is present in the field descriptor, the sign is printed only if the number is negative. It is printed just to the left of the first significant digit of the number. If you used zero or star fill, this looks ridiculous; therefore be sure to specify the sign position when you use zero or star fill.)

# -- Reserves a place for a digit, and selects space fill. Unused digit positions to the left of the most significant digit will be filled with spaces.

* -- Reserves a place for a digit, and selects star fill. Unused digit positions to the left of the most significant digit will be filled with stars.

Z -- Reserves a place for a digit, and selects zero fill. Unused digit positions to the left of the most significant digit will be filled with zeroes.

. -- Reserves a position for the decimal point. The number will be lined up with the decimal point. If no decimal point is present in the picture, none is printed. Don't try to put more than one decimal point in one descriptor.

, -- Puts a comma in the number. If the comma would precede all the non-blank characters printed in the field, the comma will not be printed.

If a mixture of #, *, and Z characters are used in field descriptor, the field will be controlled by the last one.

```
]PRINT$ "'THE ANSWER IS '###,###.##",53156.6378
THE ANSWER IS  53,156.64
```

```
]PRINT$ "####.##+/####.##-/####.##+/####.##-",
12,12.3,-12.34,-12.345
  12.00+
  12.30
  12.34-
  12.35-

]PRINT$ "5Z.3Z",125.65
00125.650
```

The listing of the DP18 PRINT code follows. There are references to five subroutines printed in previous issues of AAL in lines 1220-1260. The subroutines INPUT.NUM and INPUT.STR which are also referenced will not be printed until next month. Ah, anticipation...!

When the &DP processor encounters a PRINT token, it jumps to DP.PRINT at line 1690. I like simple code, so you can see for yourself that DP.PRINT is only three lines long. All the work is done by PRINT.END (lines 2100-2420) and the routines it calls.

PRINT.END checks for ";" and "," separators between PRINT groups, and branches to the processors for each of the three types of PRINT groups. Lines 2110-2130 check whether we are at the end of the PRINT statement. If so, AS.CROUT prints a carriage return and we leave. If not at the end, a semicolon takes us down to line 2400. There we again check for the end, because a semicolon on the end of the PRINT statement means to omit the final carriage return. A comma takes us to line 2380 where we force-print a carriage return (DP18's kind of tabbing, remember).

Lines 2180-2210 check for the three possible types of PRINT groups: "$" means print with a picture, "#" means print with a w.d format, and anything else means unformatted printing. The #w.d type is handled right here in lines 2230-2360.

Lines 2230-2250, with the help of some code in the Applesoft ROMs, read the next characters from the PRINT statement, calculate whatever expression they represent, and save the result for the field width "w". Lines 2260-2300 do the same for "d". Line 2310 evaluates the DP18 expression for the data value to be printed. Lines 2320-2350 call on the FORMAT.PRINT subroutine discussed some months ago in AAL. After printing, we go back to the top of PRINT.END to allow another PRINT group.

Unformatted printing is handled in lines 1740-2080. Line 1750 evaluates the DP18 expression to be printed. Lines 1760-1800 decide whether to use normal or exponential format, depending on position of the decimal point. The exponential format is handled by QUICK.PRINT and the normal format by FOUT, both printed in an earlier installment. We call FOUT with a format of 40 characters wide and 19 places after the decimal point. Then we print only the significant digits of the resulting string. All leading blanks and trailing zeroes are omitted. If the last character is a trailing decimal point, it too is omitted.

Printing with a picture starts at line 2440. The picture processing code is also used by DP18's INPUT$ statement, and a simple flag is used to tell who called. PRINT sets the INPUT.FLAG = 1, INPUT sets it = 0. INPUT$ and PRINT$ join at line 2470.

The first step in picture processing is to make a working copy of the picture in DP18's PICTURE.BUF. Lines 2490-2510 evaluate the string expression which is the picture. Lines 2520-2650 copy the result into PICTURE.BUF, and place a terminating $00 at the end. Line 2680 initializes a bunch of variables so we can begin to process a field within the picture. (PICTURE.BUF is 256 bytes long. If you want a good project, figure out how to avoid using PICTURE.BUF. We could with more difficulty use the picture right where it is after AS.FRMEVL finishes.)

Lines 2700-2840 control the picture parsing. The basic idea is to scan through the picture executing command characters as we go, converting numbers to repeat counts, and printing literals. When a field descriptor is encountered, it is built up in WBUF to form a template for the conversion. If any of the characters of the descriptor were preceded by a repeat count, those characters will be reduplicated the specified number of times in the WBUF template. After the template is complete, an expression will be evaluated from the PRINT list, and converted into character form. Then those characters will be merged into the template, and the result printed. I got ahead of myself a little, but I wanted to give the overall view first.

PRUS.NEXT calls LOOKUP to process each character of the picture. Lookup searches the table shown in lines 3620-4010. Each entry in the table is three bytes long: the first byte is the character to be matched, and the next two are the address of a subroutine for processing that character. Actually this address is one less than the subroutine address, because it will be pushed onto the stack and branched to with an RTS instruction (see lines 3160-3190 and 3260). The order of the entries in the table is also somewhat significant. There are three groups of entries: the first group includes characters which may be part of a numberic field descriptor; the second, characters for string field descriptors; and the third, command characters. The labels L.EITHER and L.BOTH mark the edges of these three groups.

If LOOKUP matches a character, it checks to see if the character is in the third group (line 2980). If so, we know any field descriptor which may have been building is ended, so lines 3000-3010 clear the FLD.FLAG. If not, lines 3030-3070 start a new field unless we were already in one.

Lines 3080-3140 check if we have finished a field descriptor. We may have, if the matched character was a command character or a field-descriptor character of the opposite type field. So, if the mathced character was a numeric-field character, we call PRT.STR.IF.NEEDED; if it was a string-field character, we call PRT.NUM.IF.NEEDED; and if a command character, we call both of the IF.NEEDED's. The IF.NEEDED routines check if we were building up the corresponding field descriptor. If so, we need to get a value from the PRINT list and print it now, before continuing to process the latest picture character.

Next, LOOKUP branches to the processor for the particular character matched. It sets up the repeat count, if any has been accumulated, in the Y-register. If no repeat count has been accumulated, y is set to 1. The routines are all in lines 4020-5250.

If LOOKUP does not find the picture character in the table, it may be a digit of a repeat count. If so, lines 3280-3450 multiply the existing repeat count by ten and add in the new digit. No check for overflow is done here, so if you write a repeat count of more than 255, it will be taken modulo 256. If you want to check for overflow, insert the check after line 3330:

```
                CMP #25
                BCS RP.OVERFLOW
```

and put a line after line 3610:

```
    RP.OVERFLOW JMP AS.OVRFLW
```

If the character is not even a digit, it is good for nothing but separating field descriptors. Lines 3470-3480 call the two IF.NEEDED routines, in case a field descriptor preceded the non-matching character, and then fall into PRUS.CLEAR to get ready for the next picture character.

If the picture character is Z, #, or * the code at lines 4070-4240 goes to work. There are three different entry points here. A "Z" enters at IP.ZERO, where the A-register is cleared and a $2C opcode is used to skip over the following two bytes of code. You may recall that $2C is the opcode for BIT with a two-byte address. The 6502 acts like the "LDA #' '" is an address for the BIT instruction, and in effect that "hops over" line 4110. (This is a common coding trick in the 6502 world, and is safe except when the second of the two skipped bytes is in the range from $C0 through $C7. In that range you run the risk of flipping some soft switches in the I/O space.)

Lines 4070-4140 store zero, blank, or asterisk in FILL.CHAR and in the template being created in WBUF. These positions in the template will later be replaced with the actual digits of the converted number, unless they precede the most significant digit. The "w" and "d" parameters are also incremented as appropriate, so that we can later call FOUT to create the initial image of the converted number. Lines 4220-4230 loop on the repeat count, storing multiple copies of the fill character if you used a repeat count. We also set the FOUND.NUM flag non-zero, so that the PRT.NUM.IF.NEEDED subroutine will realize the need to print.

The RTS on the end of all the IP... processors takes control back to the middle of PRUS.NEXT, because they are actually just extensions to LOOKUP.

Lines 4290-4310 handle both the + and - picture characters. The character is stored in the template, and also in SIGN.CHAR1 as a flag. We need later to know whether any + or - appeared in the template at all, so the flag will be useful then.

If a decimal point appears in the picture, we store it in the template and also note the fact by setting DECFLG non-zero. A comma is merely stored in the template. See lines 4340-4440 for these two.

Lines 4450-4560 build templates for string field descriptors. The characters A, C, and R and just counted, while saving the lates one in FOUND.CHAR. When the PRT.STR.IF.NEEDED subroutine is called later, all we will need to know is which mode to use (A, C, or R) and how wide the field is.

Lines 4570-4760 print literal strings from the picture. The only tricky part of this is the handling of the closing apostrophe. A single apostrophe signals the end of the literal string, while two apostrophe's in a row mean an apostrophe should be printed within the literal.

Slash or "X" in a picture are handled by lines 4770-4880. Note the use again of the $2C to skip over two bytes of code.

Lines 4900-4960 handle the HTAB command. This is the bare minimum handling, and I can suggest some enhancements you might like to add here. You might want to check and be sure the value is between 1 and 40, giving an error message if out of range. You might want to adapt it to work with your particular printer and 80-column card combinations. Or 132-column Ultra-Term. It's up to you.

Lines 4970-5040 process the VTAB command, and here I do check for a valid line number. Of course, if you have an Ultra-Term set up for more than 24 lines you would want to change the limit in line 5000.

Lines 5050-5180 handle the screen clearing commands. A single ">" character calls MON.CLREOL to clear from the cursor to the end of the current line. If the following character in the picture is also a ">", MON.CLREOS is called instead.

PRT.NUM.IF.NEEDED (lines 5190-5330)is one of the two IF.NEEDED twins. If FOUND.NUM is non-zero, indicating that we have been building a numeric field template, then now is the time to print a number. Unless, of course, we are doing INPUT$ rather than PRINT$. More on that subject next month. PRT.STR.IF.NEEDED (lines 6320-6460) does the same for strings.

When a number needs to be printed, lines 5340-5420 get it ready for conversion. Line 5390 evaluates the next expression from the PRINT list, and it all falls into PRT.NUM.1 at line 5440. INPUT$ has an entry at this same point.

Lines 5450-5490 make room for the sign character if the expression value is negative and a sign reservation character was used in the template. Then W and D are correct for calling FOUT in lines 5500-5530. The remainder of the PRINT.NUM subroutine copies characters from the FOUT.BUF string into the template, and then prints the fleshed-out template. Sounds easier than it really is....

Lines 5540-5690 control the scan through the template in WBUF. Commas in the template are handled right there: if any previous digits have been displayed, or if the fill character is "0" or "*", the comma is left in the template. If no digits have been stored yet and the fill character is blank, the comma is blanked out. It would look kind of silly hanging out in front of a number.

Lines 5700-5720 process a + or - character from the template. The actual code for PRUS.SGN at lines 6110-6310 does the work. If the template character is "+", it gets changed to "-" if the sign of the numeric value is negative. If the template character is "-", it gets changed to blank if the numeric value is positive.

If the template character is a digit place-holder, the next character from FOUT.BUF is examined. If the FOUT.BUF character is a digit, it is stored into the template. If not a digit, it might be a decimal point, a minus sign, or a leading blank. A leading blank gets changed to whatever the fill character is for the current template and stored in the template. A minus sign will be stored if there was no sign-position character in the template. A decimal point will be in the same position in both template and FOUT.BUF, so nothing needs to be done with it.

Since a sign-position character could come at the end of the template, lines 6000-6020 check for that condition.

Finally, lines 6030-6100 print out the composite string from WBUF.

String fields are printed by PRINT.STR, starting at line 6470. Lines 6470-6550 evaluate a string expression from the PRINT list, and set up a pointer to the resulting string value. The entry PRINT.STR.1 is shared with INPUT$. Lines 6570-6620 determine how much longer the field is than the string value. If it is too short, lines 6630-6700 fill the field with stars for an overflow indication.

If the string will fit, lines 6710-6750 store the number of left-over spaces in the field. If we are left-justifying, these will all come at the end; if right-justifying, at the beginning; if centering, half on each end. Lines 6760-6800 branch according to which type of string field we have (A, C, or R). Lines 6810-6840 print leading spaces for type-R fields.

Lines 6850-6910 divide the number of extra spaces in half, so half can be printed before the string and half after. If there were an odd number of extra spaces, the extra extra space will be printed after the string. For example, a four-character string in a nine-character field would be preceded by two blanks and followed by three.

That about winds up the discussion of the DP18 PRINT support. You can add or subtract features from this base, to create the exact configuration you need.

I should give credit to Bobby Deen for the original coding of the PRINT statement routines published this month, and the INPUT stuff next month. I revised them considerably since he wrote them two years ago, but you can still see his marks. Bobby is still pulling in a 4.0 average (highest possible) at Texas A & M, and programming for pay at the same time.

```
==============================================================================
DOCUMENT :AAL-8501:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 5 -- Issue 4                    January, 1985

In This Issue...



Note about Apple Manuals

We have mentioned before how hard it is to find the Apple technical manuals, but it looks like there is now hope.  We read somewhere this week that Apple has arranged for Addison- Wesley to distribute the manuals.  If this really comes to pass, we will probably be able to get them for you like any bookstore.  Here's hoping!

New Version of 6800/6801/6301 Cross Assembler

We have started the long process of upgrading the various S-C Cross Assemblers, and the first one is now available.  Owners of Version 1.0 of the 6800/6801/6301 Cross Assembler and of the Version 2.0 of the S-C Macro Assembler can upgrade to Version 2.0 of the Cross Assembler for $20.

If you have not already upgraded to Version 2.0 of the S-C Macro Assembler (for the 6502 et al), you need to do that first or at the same time.  If you already have 6502 Version 2.0, but don't have the older version of the 6800 product, you can go directly there for only $50.

6800 XASM Version 2.0 adds 80-column support (for //e, //c, Videx, and STB-80 users), five new directives, and all the other bells and whistles of our 2.0 products.

New disk price!!

Due to incredible competition, floppy disk prices are falling almost as fast as if they were semiconductors!  Check our ad on page three for the current low price.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

=================================================================
DOCUMENT :AAL-8501:Articles:My.Ad.txt
=================================================================

```
S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 2.0....................................$100
Version 2.0 Update..................................................$20
Source Code for Version 1.1 (on two disk sides)....................$100
Full Screen Editor for S-C Macro (with complete source code)........$49
S-C Cross Reference Utility (without source code)...................$20
S-C Cross Reference Utility (with complete source code).............$50
DISASM Dis-Assembler (RAK-Ware).....................................$30
Source Code for DISASM.....................................additional $30

S-C Word Processor (with complete source code)......................$50
Double Precision Floating Point for Applesoft (with source code)....$50
S-C Documentor (complete commented source code of Applesoft ROMs)...$50
Source Code of //e CX & F8 ROMs on disk.............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.............................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984    QD#17: Oct-Dec 1984

AWIIe Toolkit (Don Lancaster, Synergetics)..........................$39
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor.....................$60
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)......................... package of 20 for $35
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                          or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................  6 cents each
quikLoader EPROM System (SCRG)...............................($179) $170
D MAnual Controller (SCRG).....................................($90)  $85
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)...........................................($35)  $32
PROmGRAMER (SCRG).........................................($149.50) $140

Books, Books, Books..........................compare our discount prices!

    "Inside the Apple //e", Little..........................($19.95)  $18
    "Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95)  $18
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
```

```
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8501:Articles:Short.on.Mans.txt
==============================================================================

Note about Apple Manuals

We have mentioned before how hard it is to find the Apple technical manuals, but it looks like there is now hope.  We read somewhere this week that Apple has arranged for Addison-Wesley to distribute the manuals.  If this really comes to pass, we will probably be able to get them for you like any bookstore.  Here's hoping!

==========================================================================
DOCUMENT :AAL-8501:Articles:ShortPrint255.txt
==========================================================================

Short Single-Byte Hex-to-decimal Printer...Bob Sander-Cederlof

Inside DOS there exists a subroutine whose purpose is to convert a single byte into a three digit decimal number, and print it out. It is called twice from within the CATALOG processor: to print the volume number, and to print the number of sectors in a file. It isn't very space or speed efficient, and has been picked apart in various articles in Nibble and elsewhere. The DOS routine is located at $AE42.

In any case, here is a shorter routine that does the same job. I also added a little test routine which exercises the subroutine by calling it for every possible value of a byte.

Lines 1200-1290 are the test routine. It is essentially equivalent to: FOR A = 0 TO 255 : PRINT X" "; : NEXT X.

Lines 1020-1160 are the conversion and print subroutine. It is written as a loop that runs the Y-register from 2 down to 0. Line 1030 starts Y=2, and lines 1140-1150 decrement and test Y, like BASIC's NEXT Y.

Another loop keeps subtracting a table entry from the value being converted until the remainder is smaller than the table entry. The table contains powers of ten. The first time through, 100 is subtracted as many times as possible. Each time, the X-register is incremented. Since line 1040 started X out as the ASCII code for zero, when the inner loop finishes X will have the ASCII code for the next decimal digit of the original value. Line 1120 calls the monitor COUT routine to print the digit.

The next time through the table value that gets subtracted is 10, and the third and last time through 1 gets subtracted. So you see that we first print the hundreds digit, then the tens digit, and finally the units digit.

The DOS version takes 40 bytes plus a three byte table, and mine takes 25 bytes plus a three byte table. It's probably not fair to compare 40 to 25 too unfavorably, because mine does use the X-register while the DOS version does not. The part of the CATALOG code that prints the number of sectors in a file requires that the X-register not be changed, so mine is not quite compatible as is. On the other hand, DOS goes to the trouble of saving the value to be printed in location $44, which is unnecessary, and also saves a value in $45 which is otherwise totally ignored. This foolishness takes place at $ADB9-$ADBF and $AE04-$AE0A.

Anyway, here is my code:

---

DOCUMENT :AAL-8501:Articles:Sym.Sourceror.txt

---

Symbol Table Source Maker......Peter McInerney and Bruce Love

When developing a very large program in separately assembled stages, it is nice to be able to carry forward the information in the symbol table of one section into the equates section to later section. You might do this as a normal part of development or as response to a bug detected in an earlier stage which forces some re-assembly. We designed this utility program to take all the hard work out of the process of building an equate file from a symbol table.

After an assembly, BRUNning the following utility will cause whatever source is in memory to be replaced by a series of .EQ lines constructed from the current symbol table. All global labels are included, in numerical order. The generated source lines can be saved or merged in the usual fashion.

The plan of the program falls into three steps. First the existing symbol table is sorted into numeric order by the value of each symbol. Next a line corresponding to each symbol is constructed and merged into the source code. Finally the source lines are renumbered starting with 1000 using an increment of 10, and control is passed back to the S-C Macro Assembler.

We originally wrote our program based on Version 1.1 of the S-C Macro Assembler. Version 2.0 differs in that each symbol value uses four bytes rather than two, and the RENUMBER routine is in a different location. Bob Sander-Cederlof added some code to handle Version 2.0, and that version is listed here. All the changes that need to be made to use our utility with Version 1.1 are controlled by .DO-.ELSE-.FIN sets, so that you only have to change line 1030 to assemble the other version. Since the following listing was made with the CON listing option, the code between .ELSE and .FIN is shown as non-assembled lines; this allows you to type in both versions of the program.

After an assembly, the symbol table consists of 26 chains of symbols. A hash table of 26 pointers contains the beginning of each of the 26 chains. There is one chain for each letter of the alphabet, and symbols are assigned to a chain based on the first letter of the symbol name. Within each chain, the symbols are linked together in alphabetical order. The first two bytes of each symbol entry are a forward pointer to the next symbol in the chain, or $0000 if it is the end of the chain. If there is no chain for a particular letter, that pointer in the hash table will be $0000.

The value of the symbol is in the next two or four bytes (Version 1.1 or 2.0, respectively). The high byte of the value is first, the low byte last. The byte following the value contains the length of the symbol name in the lower six bits. The length will be a number between 1 and 32, or $01 and $20. Following the length byte are the characters of the name itself. Some other information is stored in the table, including various flags, local labels, and any macro definitions which were in your program; however, we are not concerned with these in our program.

The program begins by setting the output hook to point to our routine named MYCOUT. Any characters that are "printed" through the monitor's COUT routine will be routed to MYCOUT, at lines 2980-3070. MYCOUT merely stores the characters in successive positions of a buffer we put at $280. Lines 1350-1380 zap any source program still in memory, in preparation for adding the new .EQ lines.

Since every symbol carries a pointer, we decided to simply re-string them on a new chain in numeric order by value. Lines 1390-2040 build this new chain. Lines 1390-1490 and 1990-2040 step through each of the 26 alphabetical-order (A-O) chains. The numerical-order (N-O) chain is built with the pointer in ROOT pointing at the largest value, each symbol's pointer pointing at the next smallest value. When we find an A-O chain which is not empty, lines 1500-1980 chomp through the chain finding the right place in the N-O chain for each symbol.

Once the symbols are all strung on the N-O chain, lines 2050-2940 use the N-O chain to generate source lines for each symbol. Lines 2090-2100 check for the possibility of no symbols, just in case you are testing us.

Lines 2110-2210 pick up the value of the symbol (two or four bytes worth) and push it on the stack, low byte first. The loop actually pushes the byte following the value as well, because it saved a few program bytes to include it in the loop. Line 2220 pulls that byte back off.

Lines 2220-2280 pick up the characters of the symbol name and "print" them. Remember that the print hook points to MYCOUT, so that the characters are really placed in WBUF starting at WBUF+3. (The locations WBUF through WBUF+2 are reserved for the line length and line number.)

Lines 2290-2360 generate enough blanks to tab over to column 25. If the symbol is longer than 25 characters, only one blank is generated. All of the blanks are squeezed into a single compressed blank token ($80 + # of blanks). We put this into WBUF by calling MYCOUT1 to avoid the AND #$7F at the beginning of MYCOUT.

Lines 2370-2420 "print" the string of characters " .EQ $", which are stored in backwards order in line 3090.

Lines 2430-2610 "print" the value of the symbol in hexadecimal. Since the value may have up to three bytes of leading zeros, there is code here to suppress those bytes.

Lines 2620-2720 terminate the source line in WBUF with a $00 code, and store the line length in the first byte position. Now the line is ready to be added to the source code being built up.

Lines 2730-2790 make room for the new source line by lowering the pointer PRG.BEG, which points at the start of the source code. We are adding the source lines starting with the highest value, which will be at the end of the source program, and working down to the lowest value at the beginning of the source program.

Lines 2800-2850 copy the line into the hole we just made. Note that we have not filled in a valid line number yet.

LInes 2860-2940 promote the ROOT pointer to the next symbol in the N-O chain. If there are no more symbols, line 2950 calls on the RENUMBER subroutine inside the S-C Macro Assembler to put real line numbers in each line. The point at which RENUMBER is entered is just after a series of three JSR's, all to the same address. The instruction we branch to is a "CPX #$06". We are pointing this out here just in case you have a version of the S-C Macro Assembler with a slightly different position for the RENUMBER subroutine. Of course, you could omit line 2950 and just remember to type "REN" after running our program.

Finally, line 2960 restores the output hook to the 40-column screen output. This will not be what you want if you are using an 80-column card. If you are doing that, we suggest saving the output hook way back at the beginning before stuffing MYCOUT into it, and then restoring the original value here. We didn't do it that way because we were trying every possible way to make this whole program fit in only one page.

One caveat remains. We did not include any test to see whether the source code being generated starts to overlap the end of the symbol table. If you have a gigantic symbol table, say over half of the available memory for source+symbols, you may run into this problem.

When you are using this program, be sure you save the source of whatever you assembled first. Our program replaces the source in memory with the .EQ source lines. Also, realize that the symbol table is essentially wiped out by running our program, because all the chain links are restructured for numerical order. You will have to re-assemble the original program to re-create the original symbol table. Of course, if you assemble the source lines we generate, you will re-create all the global labels of the original program.

We think you will find many uses for our program, beyond the ones which prompted us to write it. We are very proud that we managed to fit everything into a single page, but don't let that stop you from adding features to fit your own needs.
1

==============================================================================
DOCUMENT :AAL-8501:Articles:XASM.6800.2.0.txt
==============================================================================

New Version of 6800/6801/6301 Cross Assembler

We have started the long process of upgrading the various S-C Cross Assemblers, and the first one is now available.  Owners of Version 1.0 of the 6800/6801/6301 Cross Assembler and of the Version 2.0 of the S-C Macro Assembler can upgrade to Version 2.0 of the Cross Assembler for $20.

If you have not already upgraded to Version 2.0 of the S-C Macro Assembler (for the 6502 et al), you need to do that first or at the same time.  If you already have 6502 Version 2.0, but don't have the older version of the 6800 product, you can go directly there for only $50.

6800 XASM Version 2.0 adds 80-column support (for //e, //c, Videx, and STB-80 users), five new directives, and all the other bells and whistles of our 2.0 products.

```
=============================================================================
DOCUMENT :AAL-8501:DOS3.3:S.DP18.Print.txt
=============================================================================

 1000  *SAVE S.DP18 PRINT
 1010  *-------------------------------
 1020  *    APPLESOFT SUBROUTINES
 1030  *-------------------------------
 1040  AS.CROUT      .EQ $DAFB    PRINT CARRIAGE RETURN
 1050  AS.COUT       .EQ $DB5C    PRINT A CHARACTER
 1060  AS.FRMEVL     .EQ $DD7B    EVAL FP FORM. OR STRING
 1070  AS.CHKCOM     .EQ $DEBE    CHECK FOR COMMA
 1080  AS.SYNERR     .EQ $DEC9    SYNTAX ERROR
 1090  AS.ILLERR     .EQ $E199    ILLEGAL QUANTITY ERROR
 1100  AS.FRESTR     .EQ $E5FD   ERR IF NOT STRING,  FREE UP A TEMP
STRING
 1110  AS.GTBYTC     .EQ $E6F5    CHRGET, THEN GETBYT
 1120  AS.GETBYT     .EQ $E6F8    GET EXPR AS BYTE IN X
 1130  *--------------------------------
 1140  *   MONITOR SUBROUTINES
 1150  *--------------------------------
 1160  MON.VTABZ     .EQ $FC24
 1170  MON.CLREOS    .EQ $FC42
 1180  MON.CLREOL    .EQ $FC9C
 1190  *--------------------------------
 1200  *      DP SUBROUTINES PRINTED ELSEWHERE
 1210  *--------------------------------
 1220  DP.NEXT.CMD          .EQ $FFFF
 1230  DP.EVALUATE          .EQ $FFFF
 1240  FOUT                 .EQ $FFFF
 1250  QUICK.PRINT          .EQ $FFFF
 1260  FORMAT.PRINT         .EQ $FFFF
 1270  INPUT.NUM            .EQ $FFFF
 1280  INPUT.STR            .EQ $FFFF
 1290  *-------------------------------
 1300  *      PAGE ZERO USAGE
 1310  *-------------------------------
 1320  MON.CH        .EQ $24
 1330  MON.CV        .EQ $25
 1340  AS.CHRGET     .EQ $B1
 1350  AS.CHRGOT     .EQ $B7
 1360  P2            .EQ $F9
 1370  P1            .EQ $FD      GP POINTER
 1380  TEMP2         .EQ $FB
 1390  *--------------------------------
 1400  WBUF          .EQ $0200
 1410  *-------------------------------
 1420  *      WORK AREAS FOR DPFP
 1430  *-------------------------------
 1440  DECFLG               .BS 1
 1450  DAC.EXPONENT         .BS 1
 1460  DAC.SIGN             .BS 1
 1470  FOUT.BUF             .BS 41
```

```
1480  STACK.PNTR          .BS 1
1490  W                   .BS 1
1500  D                   .BS 1
1510  SIGN.CHAR1          .BS 1
1520  INPUT.TYPE          .BS 1
1530  FOUND.NUM           .BS 1
1540  FOUND.STR           .BS 1
1550  STR.LEN             .BS 1
1560  REPEAT.CNT          .BS 1
1570  FOUND.LEN           .BS 1
1580  FOUND.CHAR          .BS 1
1590  FILL.CHAR           .BS 1
1600  CHAR                .BS 1
1610  INPUT.FLAG          .BS 1
1620  ZERO.CHAR           .BS 1
1630  FLD.FLAG            .BS 1
1640  FLD.START           .BS 1
1650  TEMP3               .BS 2
1660  INDEX               .BS 1
1670  PICTURE.BUF         .BS 256
1680  *-----------------------------
1690  DP.PRINT
1700         JSR AS.CHRGET
1710         JSR PRINT.END
1720         JMP DP.NEXT.CMD
1730  *--------------------------------
1740  DP.UNFORMAT
1750         JSR DP.EVALUATE   GET EXPRESSION
1760         LDA DAC.EXPONENT  GET EXPONENT
1770         CMP #$40+19       MORE THAN 18 DIGITS BEFORE DECPT?
1780         BCS .5            YES, USE SCIENTIFIC
1790         CMP #$40-1        LESS THAN .01?
1800         BCC .5            YES, USE SCIENTIFIC
1810         LDA #'0
1820         STA ZERO.CHAR
1830         LDA #40           ALLOW PLENTY OF WIDTH
1840         LDY #19           AND DECIMAL PLACES
1850         JSR FOUT
1860  *---TRIM TRAILING ZEROES---------
1870         LDY INDEX         FIND END OF BUFFER
1880  .1     DEY
1890         LDA FOUT.BUF-1,Y  TRUNCATE TRAILING ZEROES
1900         CMP #'0           IS THIS ONE ZERO?
1910         BEQ .1            ...YES, KEEP TRIMMING
1920         CMP #'.           OMIT DECIMAL POINT ON INTEGERS
1930         BEQ .2            ...GOT A DECPT
1940         INY               TRIM NO MORE...
1950  .2     LDA #0            MARK END OF MEANINGFUL CHARS
1960         STA FOUT.BUF-1,Y
1970         STY INDEX
1980  *---PRINT WITHOUT LEADING BLANKS-
1990         TAY               Y=0
2000  .3     LDA FOUT.BUF,Y
2010         BEQ PRINT.END
```

```
2020        CMP #$20         BLANK?
2030        BEQ .4           ...YES, DON'T PRINT
2040        JSR AS.COUT      ...NO, PRINT IT
2050 .4     INY
2060        BNE .3           ...ALWAYS
2070 *---PRINT WITH EXPONENT----------
2080 .5     JSR QUICK.PRINT
2090 *--------------------------------
2100 PRINT.END
2110        JSR AS.CHRGOT
2120        BNE .1           NOT ":" OR EOL
2130        JMP AS.CROUT
2140 .1     CMP #';'
2150        BEQ .3
2160        CMP #','
2170        BEQ .2
2180        CMP #'$      PRINT USING?
2190        BEQ DP.PRINT.USING
2200        CMP #'#      PRINT W,D?
2210        BNE DP.UNFORMAT NO,UNFORMATTED PRINT
2220 *---PRINT #W,D,VALUE-------------
2230        JSR AS.GTBYTC     GET W IN X-REG
2240        TXA
2250        PHA
2260        JSR AS.CHKCOM     MUST HAVE COMMA
2270        JSR AS.GETBYT     GET D IN X-REG
2280        TXA
2290        PHA
2300        JSR AS.CHKCOM     ANOTHER COMMA
2310        JSR DP.EVALUATE   GET EXPR
2320        PLA               GET D
2330        TAY
2340        PLA               GET W
2350        JSR FORMAT.PRINT
2360        JMP PRINT.END
2370 *---COMMA AFTER ITEM-------------
2380 .2     JSR AS.CROUT      DP18'S KIND OF TABBING
2390 *---"," OR ";" AFTER ITEM--------
2400 .3     JSR AS.CHRGET     NEXT CHAR
2410        BNE .1            NEXT PRINT ITEM
2420        RTS
2430 *--------------------------------
2440 DP.PRINT.USING
2450        LDA #1       PRINT,NOT INPUT
2460 *--------------------------------
2470 PRINT.INPUT
2480        STA INPUT.FLAG    0=INPUT, 1=PRINT
2490        JSR AS.CHRGET     EAT THE $
2500        JSR AS.FRMEVL     GET PICTURE
2510        JSR AS.FRESTR     ERR IF NOT STRING, FREE TEMP
2520        STX P1              ADDR IN Y,X, LEN IN A
2530        STY P1+1
2540        STA STR.LEN
2550        INC STR.LEN       WE'RE GOING TO ADD ONE
```

```
       TAY                LENGTH TO Y
       LDA #0             PUT 0 AT END OF PICTURE
       STA PICTURE.BUF,Y
       STA STACK.PNTR
       STA FLD.FLAG
.1     DEY
       LDA (P1),Y         MOVE PICTURE TO BUFFER
       STA PICTURE.BUF,Y
       TYA                TEST FOR END
       BNE .1             ...MORE
       STY REPEAT.CNT     Y IS 0
       DEY                Y = $FF
       JSR PRUS.CLEAR     CLEAR VARIABLES
*---------------------------------
*   PARSE THE PICTURE
*---------------------------------
PRUS.NEXT
       INY                NEXT CHAR
       CPY STR.LEN      DONE?
       BEQ .1             ...YES
       LDA PICTURE.BUF,Y GET A CHAR
       STY TEMP2          SAVE PICTURE PNTR
       JSR LOOKUP
       LDY TEMP2          RESTORE PICTURE PNTR
       JMP PRUS.NEXT
.1     LDA INPUT.FLAG
       BNE .2
       JMP AS.CROUT
.2     JMP PRINT.END      HANDLE ; AT END OF STATEMENT
*---------------------------------
* LOOKUP LOOKS UP THE ENTRY CORRESPONDING TO (A)
*---------------------------------
LOOKUP STA CHAR      SAVE KEY
       LDY #-3
.1     INY
       INY
       INY           NEXT ENTRY
       LDA TBL.BASE,Y
       BEQ .7        END OF TABLE
       CMP CHAR      ONE WE WANT?
       BNE .1        NO,NEXT ENTRY
*---FOUND CHAR IN TABLE----------
       CPY #L.BOTH        NEW FIELD?
       BCC .2             ...MAYBE NOT
       LDA #0             START A NEW FIELD
       STA FLD.FLAG
       BEQ .3             ...ALWAYS
.2     LDA FLD.FLAG       BEGINNING OF FIELD?
       BNE .3             ...NO, NOT A NEW FIELD
       JSR PRUS.CLEAR     ...YES, NEW FIELD
       LDA TEMP2
       STA FLD.START
       INC FLD.FLAG
*---PRINT WHATEVER'S NEEDED------
```

```
3090 .3     CPY #L.EITHER
3100        BCC .4       ...ONLY TRY PRT.STR.IF.NEEDED
3110        JSR PRT.NUM.IF.NEEDED
3120        CPY #L.BOTH
3130        BCC .5       ...ONLY TRY PRT.NUM.IF.NEEDED
3140 .4     JSR PRT.STR.IF.NEEDED
3150 *---GET ROUTINE ADDRESS----------
3160 .5     LDA TBL.BASE+2,Y
3170        PHA          PUT ADDRESS ON STACK
3180        LDA TBL.BASE+1,Y
3190        PHA
3200        LDY REPEAT.CNT    GET THE COUNT
3210        BNE .6       COUNT IS NON-0
3220        INY          COUNT IS 0, SO MAKE IT 1
3230 .6     LDA #0       CLEAR REPEAT.CNT
3240        STA REPEAT.CNT
3250        LDA CHAR     GET THE ORIGINAL CHARACTER
3260        RTS          JUMP TO ROUTINE
3270 *---CHAR NOT IN TABLE------------
3280 .7     LDA CHAR     GET CHAR AGAIN
3290        EOR #'0      CHECK FOR DIGIT 0-9
3300        CMP #10
3310        BCS .9       ...NOT A NUMBER
3320        STA TEMP3
3330        LDA REPEAT.CNT    PREVIOUS * 10
3340        ASL               *2
3350        ASL               *4
3360        ADC REPEAT.CNT    *5
3370        ASL               *10
3380        ADC TEMP3         + DIGIT
3390        STA REPEAT.CNT
3400        LDA FLD.FLAG      BEGINNING OF FIELD?
3410        BNE .8            ...NO
3420        LDA TEMP2         YES, SAVE STARTING POSN
3430        STA FLD.START
3440        INC FLD.FLAG
3450 .8     RTS
3460 *---NOT IN TABLE, NOT A DIGIT----
3470 .9     JSR PRT.STR.IF.NEEDED
3480        JSR PRT.NUM.IF.NEEDED
3490 *--------------------------------
3500 PRUS.CLEAR
3510        LDX #1
3520        STX W        W = 1
3530        DEX          REST = 0
3540        STX D
3550        STX DECFLG   NO DECIMAL
3560        STX SIGN.CHAR1
3570        STX FOUND.NUM    FLAG IF # HAS BEEN FOUND
3580        STX FOUND.STR
3590        STX FOUND.LEN
3600        STX FOUND.CHAR
3610        RTS
3620 *--------------------------------
```

```
*   TABLE IS IN THREE SECTIONS:
*      1ST SECTION (BEFORE L.EITHER) ARE FOR
*      FOR DESCRIBING NUMERIC FIELDS, AND CAN
*      TERMINATE A STRING FIELD.
*
*      2ND SECTION (BTWN L.EITHER & L.BOTH) IS
*      FOR DESCRIBING STRING FIELDS, AND CAN
*      TERMINATE A NUMERIC FIELD
*
*      3RD SECTION (AFTER L.BOTH) CAN TERMINATE
*       BOTH KINDS OF FIELDS.
*
*      TABLE FORMAT = #CHAR,ADDRESS-1
*      END OF TABLE MARKED WITH $00
*---------------------------------
       .MA TBL
       .DA #']1',]2-1
       .EM
*---------------------------------
TBL.BASE
       >TBL "+",IP.PLUS.MINUS   -#-
       >TBL "-",IP.PLUS.MINUS   -#-
       >TBL "#",IP.NUMBER       -#-
       >TBL "*",IP.ASTERISK     -#-
       >TBL "Z",IP.ZERO         -#-
       >TBL ".",IP.POINT        -#-
       >TBL ",",IP.COMMA        -#-
L.EITHER .EQ *-TBL.BASE
       >TBL "A",IP.ACR          -$-
       >TBL "C",IP.ACR          -$-
       >TBL "R",IP.ACR          -$-
L.BOTH   .EQ *-TBL.BASE
       >TBL "'",IP.QT           -#$-
       >TBL "/",IP.SLASH        -#$-
       >TBL "X",IP.X            -#$-
       >TBL "H",IP.HTAB         -#$-
       >TBL "V",IP.VTAB         -#$-
       >TBL ">",IP.GREATER      -#$-
       .HS 00       END OF TABLE
*---------------------------------
*   Z -- Digit position marker, zero fill
*   # -- Digit position marker, blank fill
*   * -- Digit position marker, star fill
*---------------------------------
IP.ZERO
       LDA #'0      USE 0 FOR FILL CHAR
       .HS 2C
IP.NUMBER
       LDA #' '     USE BLANK FOR FILL CHAR
IP.ASTERISK
       STA FILL.CHAR  SAVE AS FILL CHAR
.1     JSR STA.WBUFX.INX
       INC FOUND.NUM  FOUND A DIGIT
       INC W        LENGTH
```

```
       PHA
       LDA DECFLG   HAD DECIMAL PT?
       BEQ .2       NO
       INC D        YES
.2     PLA
       DEY
       BNE .1       NEXT ONE
       RTS
*--------------------------------
*   + -- Sign position marker (prints + or -)
*   - -- Sign position marker (prints space or -)
*--------------------------------
IP.PLUS.MINUS
       STA SIGN.CHAR1 SAVE SIGN CHAR
       JMP STA.WBUFX.INX
*--------------------------------
*   . -- Decimal position marker
*--------------------------------
IP.POINT
       INC DECFLG   FOUND A DECIMAL POINT
*--------------------------------
*   , -- Puts a comma in a number
*--------------------------------
IP.COMMA
STA.WBUFX.INX
       STA WBUF,X   SAVE CHAR
       INX
       RTS
*--------------------------------
*   A -- String field, left justified
*   C -- String field, centered
*   R -- String field, right justified
*--------------------------------
IP.ACR INC FOUND.STR     FOUND A STRING
       STA FOUND.CHAR    SAVE THE CHAR
       TYA
       CLC
       ADC FOUND.LEN  ADD LENGTH TO REPEAT COUNT
       STA FOUND.LEN
       RTS
*--------------------------------
*   ' -- Start of embedded string
*--------------------------------
IP.QT
.1     LDX TEMP2         X = PICTURE PNTR
.2     INX
       LDA PICTURE.BUF,X GET CHAR
       CMP #''           APOSTROPHE?
       BNE .3            ...NO, PRINT IT
       LDA PICTURE.BUF+1,X
       CMP #''           TWO APOSTROPHE'S IN A ROW?
       BNE .4            ...NO, MEANS END OF LITERAL
       INX               ...YES, PRINT APOSTROPHE
.3     JSR AS.COUT
```

```
       JMP .2
.4     DEY          REPEAT COUNT
       BNE .1       ...REPEAT THE STRING
       STX TEMP2    NEW PICTURE PNTR
       RTS
.5     JMP AS.SYNERR
*--------------------------------
*   / -- Print n carriage returns
*   X -- print n spaces
*--------------------------------
IP.SLASH
       LDA #$0D     CR'S
       .HS 2C       (SKIP NEXT 2 BYTES)
IP.X   LDA #$20     BLANKS'
.1     JSR AS.COUT  PRINT THE CHAR
       DEY
       BNE .1
       RTS
*--------------------------------
*   H -- HTAB to column n
*   V -- VTAB to line n
*--------------------------------
IP.HTAB
       DEY
       STY MON.CH   HTAB
       RTS
*--------------------------------
IP.VTAB
       DEY
       CPY #24
       BCS .1       OUT OF RANGE
       TYA
       JMP DP.VTAB
.1     JMP AS.ILLERR  ILLEGAL QUANTITY ERROR
*--------------------------------
*   > -- CLEAR TO END OF LINE
*  >> -- CLEAR TO END OF SCREEN
*--------------------------------
IP.GREATER
       LDY TEMP2
       LDA PICTURE.BUF+1,Y
       CMP #'>'
       BEQ .1       ...CLEAR TO END OF SCREEN
*---CLEAR TO END OF LINE---------
       JMP MON.CLREOL
*---CLEAR TO END OF SCREEN-------
.1     INC TEMP2
       JMP MON.CLREOS
*--------------------------------
PRT.NUM.IF.NEEDED
       LDA FOUND.NUM  HAS # BEEN FOUND?
       BEQ .1       NO
       TYA
       PHA          SAVE Y
```

```
5250        LDA INPUT.FLAG
5260        BEQ .2       INPUT
5270        JSR PRINT.NUM     PRINT
5280        JMP .3
5290 .2     JSR INPUT.NUM
5300 .3     PLA          RESTORE Y
5310        TAY
5320        JSR PRUS.CLEAR
5330 .1     RTS
5340 *---------------------------------
5350 PRINT.NUM
5360        LDA #0       PUT $00
5370        STA WBUF,X   AT END OF STRING
5380        JSR AS.CHKCOM  MUST HAVE COMMA
5390        JSR DP.EVALUATE   GET EXPRESSION
5400        LDA #'0
5410        STA ZERO.CHAR
5420 *
5430 *---------------------------------
5440 PRT.NUM.1
5450        LDA DAC.SIGN
5460        BPL .1
5470        LDA SIGN.CHAR1  SIGN IS -
5480        BEQ .1       NO SIGN CHAR
5490        INC W        RESERVE PLACE FOR SIGN
5500 *---CONVERT VALUE INTO FOUT.BUF--
5510 .1     LDA W
5520        LDY D
5530        JSR FOUT
5540 *---FILL IN THE PICTURE----------
5550        LDX #0       INDEX INTO WBUF
5560        LDY #0       INDEX INTO FBUF
5570        STY DECFLG   USE FOR DIGITS FLAG
5580 .2     LDA WBUF,X   GET CHAR FROM PICTURE
5590        BEQ .10      END OF PICTURE
5600        CMP #',      COMMA?
5610        BNE .3
5620        INX
5630        LDA DECFLG   ANY DIGITS BEFORE THIS?
5640        BNE .2            ...YES, LEAVE COMMA
5650        LDA FILL.CHAR     ...NO, BUT LEAVE IF FILL
5660        CMP #' '                 IS NON-BLANK.
5670        BNE .2       ...NOT BLANK, SO LEAVE IN THE COMMA
5680        STA WBUF-1,X ...COVER COMMA WITH BLANK
5690        BNE .2       ...ALWAYS
5700 *---CHECK FOR PICTURE SIGN-------
5710 .3     JSR PRUS.SGN      IF + OR -, PROCESS
5720        BCC .2            ...WAS + OR -
5730 *---PICTURE IS DIGIT OR DECPT----
5740        LDA FOUT.BUF,Y    GET CHAR FROM VALUE STRING
5750        CMP #$20          SPACE?
5760        BNE .5            ...NO
5770        LDA FILL.CHAR     ...YES, USE FILL CHAR
5780 .5     PHA               SAVE FOUT OR FILL CHAR
```

```
5790        CMP #'-           IS IT A SIGN CHAR?
5800        BNE .7            ...NO
5810        LDA SIGN.CHAR1    IS THERE A SIGN IN FORMAT?
5820        BNE .8            ...YES, SKIP THE SIGN
5830        LDA WBUF+1,X      ...NO, INSTALL SIGN HERE
5840        CMP #',           (UNLESS NEXT PIC.CHAR IS COMMA)
5850        BNE .6            ...NOT COMMA
5860        LDA FILL.CHAR     ...COMMA, SO COVER WITH FILLER
5870        JSR STA.WBUFX.INX
5880 .6     LDA FOUT.BUF,Y    GET SIGN CHAR AGAIN
5890 .7     JSR STA.WBUFX.INX
5900 .8     PLA               GET FOUT OR FILL CHAR BACK
5910        INY               ADVANCE FOUT PNTR
5920        CPY INDEX         END OF FOUTBUF?
5930        BCS .9            ...YES
5940        CMP FILL.CHAR     IF WE INSTALLED A DIGIT
5950        BEQ .2            WE MUST SET THE DIGITS FLAG
5960        CMP #'-           SIGN CHAR?
5970        BEQ .2            ...YES
5980        INC DECFLG        FOUND A DIGIT
5990        BNE .2            ...ALWAYS
6000 *---END OF FOUT.BUF-------------
6010 .9     LDA WBUF,X
6020        JSR PRUS.SGN
6030 *---END OF FOUT OR PICTURE-------
6040 .10    LDY #0
6050 .11    LDA WBUF,Y
6060        BEQ .12
6070        JSR AS.COUT  PRINT IT
6080        INY
6090        BNE .11      ALWAYS
6100 .12    RTS
6110 *--------------------------------
6120 PRUS.SGN
6130        CMP #'+      SIGN?
6140        BNE .1       NO
6150        INX
6160        LDA DAC.SIGN
6170        BPL .2      SIGN ALREADY +
6180        LDA #'-
6190        STA WBUF-1,X
6200        BNE .2       ALWAYS
6210 .1     CMP #'-      -?
6220        BNE .3       NO
6230        INX
6240        LDA DAC.SIGN
6250        BMI .2      SIGN ALREADY -
6260        LDA FILL.CHAR
6270        STA WBUF-1,X   BLANK OUT SIGN
6280 .2     CLC
6290        RTS
6300 .3     SEC
6310        RTS
6320 *--------------------------------
```

```
6330  PRT.STR.IF.NEEDED
6340         LDA FOUND.STR HAS STRING BEEN FOUND?
6350         BEQ .3       NO
6360         TYA
6370         PHA          SAVE Y
6380         LDA INPUT.FLAG
6390         BEQ .1
6400         JSR PRINT.STR
6410         JMP .2
6420  .1     JSR INPUT.STR
6430  .2     PLA
6440         TAY          RESTORE Y
6450         JSR PRUS.CLEAR
6460  .3     RTS
6470  *---------------------------------
6480  PRINT.STR
6490         LDA #$20
6500         STA FILL.CHAR
6510         JSR AS.CHKCOM  MUST HAVE COMMA
6520         JSR AS.FRMEVL     GET EXPRESSION
6530         JSR AS.FRESTR     GET ADR AND LEN
6540         STX P2
6550         STY P2+1
6560  *---------------------------------
6570  PRINT.STR.1
6580         PHA          SAVE LENGTH
6590         SEC               LENGTH IS IN A
6600         SBC FOUND.LEN     SUBTRACT FIELD LENGTH
6610         BEQ .2            ...SAME, SO OKAY
6620         BCC .2            ...EXP IS SHORTER THAN FIELD
6630  *---FIELD OVERFLOW---------------
6640         PLA               DISCARD LENGTH
6650         LDY FOUND.LEN     GET FIELD LEN
6660         LDA #'*           OVERFLOW CHAR
6670  .1     JSR AS.COUT
6680         DEY
6690         BNE .1
6700         RTS
6710  *---JUSTIFY IN FIELD-------------
6720  .2     EOR #$FF     GET POSITIVE #
6730         TAY
6740         INY
6750         STY FOUND.LEN
6760         LDA FOUND.CHAR
6770         CMP #'A      LJ FIELD
6780         BEQ .5
6790         CMP #'C      CJ FIELD
6800         BEQ .4
6810  *---RIGHT JUSTIFY----------------
6820         JSR PRINT.Y.SPACES
6830         PLA          RESTORE STRING LEN
6840         JMP PRT.STR  PRINT STRING
6850  *---CENTER JUSTIFY---------------
6860  .4     TYA          # OF SPACES
```

```
6870        LSR          DIVIDE BY 2
6880        TAY          # LEADING BLANKS
6890        ADC #0       +1 IF IT WAS ODD
6900        STA FOUND.LEN   # TRAILING BLANKS
6910        JSR PRINT.Y.SPACES
6920 *---LEFT JUSTIFY-----------------
6930 .5     PLA          GET STRING LEN
6940        JSR PRT.STR  PRINT IT
6950        LDY FOUND.LEN  TRAILING SPACES
6960        JMP PRINT.Y.SPACES
6970 *--------------------------------
6980 PRT.STR
6990        STA FOUND.CHAR    LEN OF STRING
7000        LDY #$FF
7010 .1     INY
7020        CPY FOUND.CHAR
7030        BCS .2       DONE
7040        LDA (P2),Y   GET CHAR
7050        JSR AS.COUT  PRINT IT
7060        JMP .1
7070 .2     RTS
7080 *--------------------------------
7090 PRINT.Y.SPACES
7100        TYA          TEST COUNT
7110        BEQ .2       ...ZERO, EXIT NOW
7120        LDA FILL.CHAR
7130 .1     JSR AS.COUT
7140        DEY
7150        BNE .1
7160 .2     RTS
7170 *--------------------------------
7180 DP.VTAB
7190        STA MON.CV
7200        JMP MON.VTABZ
7210 *--------------------------------
```

=========================================================================
DOCUMENT :AAL-8501:DOS3.3:S.PRINT.000.255.txt
=========================================================================

```
 1000  *SAVE S.PRINT 000-255
 1010  *--------------------------------
 1020  PRINT.000.255
 1030         LDY #2
 1040  .1     LDX #"0"
 1050  .2     CMP DECTBL,Y
 1060         BCC .3       DIGIT FINISHED
 1070         SBC DECTBL,Y
 1080         INX
 1090         BNE .2       ...ALWAYS
 1100  .3     PHA          SAVE REMAINDER
 1110         TXA
 1120         JSR $FDED
 1130         PLA          GET REMAINDER
 1140         DEY
 1150         BPL .1
 1160         RTS
 1170  *--------------------------------
 1180  DECTBL .DA #1,#10,#100
 1190  *--------------------------------
 1200  T      LDA #0
 1210  .1     PHA          SAVE VALUE
 1220         JSR PRINT.000.255
 1230         LDA #" "
 1240         JSR $FDED
 1250         PLA          GET PREVIOUS VALUE
 1260         CLC
 1270         ADC #1       INCREMENT
 1280         BNE .1
 1290         RTS
```

```
==============================================================================
DOCUMENT :AAL-8501:DOS3.3:S.SymSourceror.txt
==============================================================================
```

```
 1000         .LIST CON
 1010  *SAVE S.SYMBOL SOURCEROR
 1020  *---------------------------------
 1030  VERSION    .EQ 1    0=1.1, 1=2.0
 1040  *---------------------------------
 1050  *   THE FOLLOWING ADDRESS SHOULD POINT
 1060  *   TO A "CPX #$06" INSTRUCTION.  IF IT
 1070  *   DOESN'T IN YOUR PARTICULAR COPY, FIND
 1080  *   THAT INSTRUCTION AND PLACE THE CORRECT
 1090  *   ADDRESS HERE.
 1100  *---------------------------------
 1110      .DO VERSION     ...V 2.0
 1120  RENUMBER .EQ $D65B       V 2.0
 1130      .ELSE           ...V 1.1
 1140  RENUMBER .EQ $D7DA       V 1.1
 1150      .FIN
 1160  *---------------------------------
 1170  PTR        .EQ $00,01
 1180  A1         .EQ $02,03
 1190  A2         .EQ $04,05
 1200  ROOT       .EQ $06,07
 1210  XSAVE      .EQ $8
 1220  CSW        .EQ $36,37
 1230  *---------------------------------
 1240  HASH.TAB   .EQ $132
 1250  WBUF       .EQ $280
 1260  *---------------------------------
 1270  PRBYTE     .EQ $FDDA
 1280  COUT       .EQ $FDED
 1290  SETVID     .EQ $FE93
 1300  *---------------------------------
 1310  *      PROGRAM POINTERS
 1320  *---------------------------------
 1330  PRG.BEG    .EQ $CA,CB
 1340  PRG.END    .EQ $4C,4D
 1350  *---------------------------------
 1360  MAKE.SOURCE.FROM.SYMBOL.TABLE
 1370         LDA #MYCOUT       GRAB THE OUTPUT HOOK
 1380         STA CSW
 1390         LDA /MYCOUT
 1400         STA CSW+1
 1410         LDA PRG.END       EMPTY THE PROGRAM AREA
 1420         STA PRG.BEG
 1430         LDA PRG.END+1
 1440         STA PRG.BEG+1
 1450  *---SCAN THROUGH HASH TABLE------
 1460         LDX #0
 1470         STX ROOT          EMPTY NUMERIC-ORDER CHAIN
 1480         STX ROOT+1
```

```
1490  *---GET START OF NEXT CHAIN------
1500  .1     LDA HASH.TAB+1,X
1510         BEQ .6            ...THIS CHAIN IS EMPTY
1520         STA PTR+1
1530         LDA HASH.TAB,X
1540         STA PTR
1550         STX XSAVE
1560  *---SEARCH FOR POSITION IN N-O CHAIN---
1570  .2     LDA #ROOT    START SEARCH FROM BEGINNING
1580         STA A1            OF NUMERIC-ORDER CHAIN
1590         LDA /ROOT
1600         STA A1+1
1610  .3     LDA A1       PROMOTE BOTH POINTERS
1620         STA A2            TO THE NUMERIC-ORDER CHAIN
1630         LDA A1+1
1640         STA A2+1
1650         LDY #0
1660         LDA (A1),Y
1670         TAX
1680         INY
1690         LDA (A1),Y
1700         STA A1+1
1710         STX A1
1720         BEQ .5
1730  *---COMPARE A-O WITH N-O VALUE---
1740      .DO VERSION     ...V 2.0
1750         LDX #3       4-BYTE VALUES
1760      .ELSE           ...V 1.1
1770         LDX #1       2-BYTE VALUES
1780      .FIN
1790         SEC
1800  .4     INY
1810         LDA (A1),Y
1820         SBC (PTR),Y
1830         DEX
1840         BPL .4
1850         BCS .3       ...A-O VALUE < N-O VALUE
1860  *---INSERT A-O VALUE INTO N-O CHAIN---
1870  .5     LDY #0
1880         LDA (PTR),Y
1890         TAX
1900         LDA A1
1910         STA (PTR),Y
1920         LDA PTR
1930         STA (A2),Y
1940         INY
1950         LDA (PTR),Y
1960         PHA
1970         LDA A1+1
1980         STA (PTR),Y
1990         LDA PTR+1
2000         STA (A2),Y
2010         STX PTR
2020         PLA
```

```
2030        STA PTR+1
2040        BNE .2       ...NOT END OF CHAIN YET
2050  *---NEXT HASH CHAIN--------------
2060        LDX XSAVE
2070  .6    INX
2080        INX
2090        CPX #2*26    26 HASH CHAINS
2100        BCC .1       ...STILL ANOTHER CHAIN
2110  *--------------------------------
2120  *   RUN THROUGH NUMERIC-ORDER CHAIN
2130  *   AND CREATE A SOURCE LINE FOR EACH SYMBOL.
2140  *--------------------------------
2150        LDA ROOT+1   CHECK FOR NO CHAIN AT ALL
2160        BEQ .17
2170     .DO VERSION     ...V 2.0
2180  .8    LDX #4
2190     .ELSE           ...V 1.1
2200  .8    LDX #2
2210     .FIN
2220        LDY #2
2230  .9    LDA (ROOT),Y
2240        PHA
2250        INY
2260        DEX
2270        BPL .9
2280        PLA
2290        AND #$3F
2300        TAX
2310  .10   LDA (ROOT),Y
2320        JSR COUT
2330        DEX
2340        BNE .10
2350  *---TAB TO .EQ COLUMN------------
2360        LDA #$81
2370        CPY #25
2380        BCS .11
2390        TYA
2400        EOR #$FF
2410        ADC #$9A
2420  .11   JSR MYCOUT1
2430  *---OUTPUT ".EQ $"---------------
2440        LDX #4
2450  .12   LDA STRING,X
2460        JSR COUT
2470        DEX
2480        BPL .12
2490  *---OUTPUT VALUE OF SYMBOL-------
2500     .DO VERSION     ...V 2.0
2510        LDX #4
2520        PLA
2530        BNE .16      ...PRINT 32-BITS
2540        DEX
2550        PLA
2560        BNE .16      ...PRINT 24-BITS
```

```
2570       .ELSE           ...V 1.1
2580          LDX #2
2590       .FIN
2600          DEX
2610          PLA
2620          BNE .16      ...PRINT 24-BITS
2630          DEX
2640  .13     PLA
2650  .16     JSR PRBYTE
2660          DEX
2670          BNE .13
2680  *---APPEND $00 BYTE--------------
2690          TXA          APPEND $00 BYTE
2700       .DO VERSION     ...V 2.0
2710          STA WBUF-4,Y
2720          DEY
2730          DEY
2740       .ELSE           ...V 1.1
2750          STA WBUF-2,Y
2760       .FIN
2770          DEY
2780          STY WBUF     # BYTES IN LINE
2790  *---MAKE ROOM IN SOURCE AREA-----
2800          LDA PRG.BEG
2810          SEC
2820          SBC WBUF
2830          STA PRG.BEG
2840          BCS .14
2850          DEC PRG.BEG+1
2860  *---COPY LINE INTO SOURCE AREA---
2870  .14     DEY
2880  .15     LDA WBUF,Y
2890          STA (PRG.BEG),Y
2900          DEY
2910          BPL .15
2920  *---NEXT SYMBOL FROM CHAIN-------
2930          INY          Y=0
2940          LDA (ROOT),Y      FROM THE NUMERIC-ORDER CHAIN
2950          TAX
2960          INY
2970          LDA (ROOT),Y
2980          STA ROOT+1
2990          STX ROOT
3000          BNE .8       ...NOT END OF CHAIN YET
3010          JSR RENUMBER ...END, SO RENUMBER THE LINES
3020  .17     JMP SETVID   RESTORE HOOK AND RETURN
3030  *--------------------------------
3040  MYCOUT
3050          AND #$7F
3060  MYCOUT1
3070          INY
3080       .DO VERSION     ...V 2.0
3090          STA WBUF-5,Y
3100       .ELSE           ...V 1.1
```

```
3110        STA WBUF-3,Y
3120     .FIN
3130        RTS
3140 *--------------------------------
3150 STRING .AS "$ QE."
3160 *--------------------------------
3170 END
```

==============================================================================
DOCUMENT :AAL-8502:Articles:Book.review.txt
==============================================================================

Review of "Assembly Language for the Applesoft Programmer"
......reviewed by Bob Sander-Cederlof

Roy E. Myers (author of Microcomputer Graphics) and C.W. Finley, Jr., are the authors of the new book named above, and published by Addison-Wesley. We like it.

Until August of last year we consistently recommended Roger Wagner's "Assembly Lines: the Book" when you asked us which book would best help you learn Apple assembly language. It was especially well-suited to beginners at assembly language who were nevertheless somewhat familiar with the Apple and Applesoft. But it went out of print with the demise of Softalk Publishing, and we can't get them now.

Finley and Myers have not only filled the void, they have improved on our previous favorite. Physically, the book is larger (7x9, paper, 361 + vi pages). It is set in large clear type. And it only costs $16.95 (Wagner's book was $19.95). I especially like the fact that they use the S-C assembler for all of the examples. However, if you don't use our assembler, the book loses no value; all the examples are written so as to be as compatible as possible with other possible assemblers.

Take another look at that title: "Assembly Language for the Applesoft Programmer." There is a double meaning there. This is not only a text for the Applesoft programmer who wants to learn beginning assembly language. It also for the person who wants to USE assembly language along with Applesoft programs. Combining both languages gives the best of both worlds, but doing so involves a lot of work. This book will help.

The book divides into five main sections:

* Introduction

* Fundamentals of 6502 Programming: 6502 architecture, instruction set; addressing; branches, loops, nesting; logical operations and bit manipulation.

* Linkage: fitting a program into the Apple; accessing machine language programs via BLOAD, POKE, USR, ctrl-Y, and "&"; soft switches; using Applesoft ROM subroutines, esp. floating point math; development of a working example.

* Graphics: the Screen, its organization and addressing with text, lo-res, and hi-res; ROM routines for lo- and hi-res graphics; bit-pattern images and animation; bit-masking techniques and complementary drawing; development of a working shoot-em-up video game (GREMLIN).

*  Searching and Sorting:  &-routine to sort array elements; another to search strings.

There are five useful appendices and an index.

We think enough of this book to add it to our stock.  Check our list of books on page 3 for price.

=============================================================================
DOCUMENT :AAL-8502:Articles:DOSless.Disks.txt
=============================================================================

Making Dos-less Disks......................Bob Sander-Cederlof

Last night I re-invented the wheel, and I think I made a pretty good one. I learned a little at the same time.

When you use the DOS "INIT" command, a copy of DOS is written on tracks 0 through 2. If the disk is meant to be a data disk, that wastes three perfectly good tracks. Because of the way DOS checks for the end of track-sector lists and various other things, a standard DOS cannot allow files to be written into track 0. But it is perfectly all right to leave the DOS image off of tracks 1 and 2 and use them for files. Of course it is a good idea to change the image on track 0 so that it will not begin to boot DOS and get lost (when you forget it is DOS-less and try to boot it anyway).

There are some more wasted sectors in track 17, the catalog track. INIT sets up 15 sectors for the catalog, which is enough for 105 files. I have never needed that many, but some of you might have even needed more. Last night I needed only about 30 files, and I needed every sector I could get to store them all. My "wheel" sets up only seven catalog sectors, enough for only 49 files. This frees up eight more sectors for data.

With the help of "Beneath Apple DOS" I examined the code in the DOS File Manager which handles the INIT command ($AE8E-AF07). This routine calls RWTS to initialize 35 empty tracks on a diskette, writes a VTOC in track 17 sector 0 and writes 15 empty catalog sectors on the rest of track 17. Then it scoots back to track 0 and writes the DOS image on the first three tracks.

I used Rak-Ware's DISASM to make a source file out of the INIT code, and then loaded it into the S-C Macro Assembler. Then step-by-step I proceeded to add meaningful labels and comments, and modify the code to do what I wanted.

The File Manager INIT code expects various parameters to have been set up by the DOS command parser, and those will not be set up when my program runs. I decided I would let my program assume that the last disk drive you accessed is the one where you have placed the blank disk you want to initialize.

I also decided to make the volume number always 001. I always do this anyway, and generally consider the volume number to be a nuisance (since I don't have a Corvus which uses the volume numbers for something useful). If you want to be able to choose the volume number, you could add the code for that purpose. Lines 1240-1270 set the volume number into the VTOC image and into the RWTS parameter block (IOB).

Lines 1290-1300 call RWTS to format the blank diskette. Beware! It is entirely too easy to forget to remove your heavily loaded program diskette before running this program! Be absolutely SURE you have the diskette in the drive which you WANT to initialize. After this program runs, the disk will have no remnant of any data which may have been on it before.

Lines 1310-1570 set up a VTOC image. The program assumes that part of the VTOC image at $B3BB is already set up, because you could not run this program without having read at least one VTOC somewhere along the way. The VTOC bitmap is set up first to $FFFF0000 at each sector position, and then the entry for track 0 is cleared. Finally the bits for sector 0 and sectors 9 through 15 of track 17 are cleared. Then lines 1580-1640 call on RWTS to write out the VTOC on track 17, sector 0.

The catalog sectors are chained together with a series of pointers. A pointer in the VTOC points to the first catalog sector, which is almost always track 17 sector 15. A pointer in the first catalog sector points to the second one, and so on. The last catalog sector points at track 0, which is a flag indicating the end of the catalog. (Too bad, because if DOS tested for a final pointer to 0,0 instead of just 0,x we could put the catalog for this data disk all in track 0 and free up even more sectors.)

Lines 1650-1700 clear the catalog buffer, and then lines 1710-1900 insert the forward pointers and call on RWTS to write each sector on the disk.

Finally, lines 1910-2000 write out a bootup program on track 0 sector 0. BOOTER is the code that will be executed if you accidentally try to boot our DOS-less disk.

Lines 2010-2090 finish setting up a call to RWTS, and check for an I/O error. I didn't bother to write any error handler into this program, as you can see by the BRK in line 2090. If you want you can printout the DOS error code at this point, or at least get it in the A-register before the BRK.

The BOOTER program is tricker than it looks. Anyway it tricked me a lot. First notice the .PH and .EP directives in lines 2120 and 2280. These tell the assembler to continue assembling bytes following the preceding code, but to assemble it with the assumption that at execution time it will be origined at $0800. The boot ROM on the disk controller reads track 0 sector 0 into $800-$8FF, so BOOTER has to be set up to run there.

Notice line 2140, which is ".HS 01" The boot ROM reads the first sector into $800-8FF, then checks location $800 to see how many sectors you want the boot ROM to read. About the only disk I have heard of which has anything other than 01 in this byte is the BASICS disk. If you put, for example, 03 in that byte sectors 1 and 2 would be read into $900 and $A00. You can read up to 16 sectors this way, but remember that the sector numbers will not be the same as the ones you use when you write them with RWTS. (RWTS uses a table to convert logical sector numbers into physical sector numbers.)

Line 2150 turns off the disk motor. I forgot the first time, and of course the drive just kept spinning.

Lines 2160-2210 print out the message from lines 2240-2270. My first attempt I called the standard COUT subroutine at $FDED to print each character, and I lost an hour finding out why I never saw my message. Instead, the drive just kept grinding the head to track 0, over and over and over.... But it worked if I first copied the boot ROM code from $C600 down to $8600, and typed 8600G to boot. I finally figured out that PR#6 sets the output hook to slot 6 and leaves it there. Then the next character that is printed (usually the prompt character for whatever language you are in) through COUT goes to the disk interface and proceeds to boot. My message sent another character to COUT and restarted the boot, ad infinitum. Changing line 2190 to "JSR $FDF0" fixed it all.

After printing the message line 2220 jumps to the initial entry point of the monitor, so you get a "*" prompt. If you previously had DOS in memory, you will probably be able to use 3D0G to get back to BASIC or the assembler or whatever. Otherwise, stick in a disk that DOES have DOS and try booting again.

Line 2300 is just window dressing. It assures that the rest of track 0 sector 0 will have nothing but zeroes in it. No particular value, but I like it that way.

```
===========================================================================
DOCUMENT :AAL-8502:Articles:DP18.Input.txt
===========================================================================
```

18-Digit Arithmetic, Part 10...............Bob Sander-Cederlof

At least one error crept into the PRINT USING program we printed last month. A line should be inserted to correct the problem:

```
      3045     JSR PRUS.CLEAR    YES, NEW FIELD
```

This is what I expect to be the final installment of the DP18 series. Some of you have been typing in and trying out the various installments, and others buying the source code on the various quarterly disks. We plan to make the composite DP18 source available at a reasonable price: all parts will be properly integrated as a set of 12 source files, ready to assemble with the S-C Macro Assembler. The disk will also include example programs illustrating the various features, the object file of DP18, and a loader program for installing DP18. The price for all of it, on one diskette, will be $50.

Normal Applesoft INPUT statements can be written in several ways. An optional quotation can be used for a prompting message; if one is used a semicolon must follow the quotation. A list of one or more variables follows.

```
       INPUT variable
       INPUT "quote";variable
```

In DP18 we implemented the two forms of the INPUT statement shown above, except that only a single variable may be used in each statement. We also implemented two additional kinds of INPUT statements. INPUT# statements allow expressions to be entered during execution. INPUT$ statements allow picture- controlled input.

```
       INPUT # variable
       INPUT # "quote";variable
       INPUT $ string,variable-list
```

The INPUT# statement allows you to read expressions and evaluate them during an INPUT operation. This can greatly simplify entering some numbers. For example, one-third can be entered as either ".333333333333333333333" or simply as "1/3". You can enter values such as SQR(2), 2*PI, and so on. You can even refer to variables used in the program. After you have entered the expression and typed RETURN, DP18 calls on Applesoft to tokenize the line, evaluates the expression to a numeric value, and stores the value in the INPUT variable your program specified.

We call the INPUT$ statement "INPUT using". It is analogous to "PRINT using", or the PRINT$ statement discussed last month. All characters in the INPUT$ picture are proccessed the same as for PRINT$ until characters defining a numeric or string field are encountered. Then the magic begins....

For a numeric field, underlines are printed to indicate digit positions. The cursor is placed after the last underline. If there is a decimal point in the picture it will be printed. A plus sign in the picture will also be printed. All other positions of the field will be printed as underlines. Once the field has been displayed in this fashion, DP18 will check the current value in the variable corresponding with the field. If the current value is zero, DP18 merely waits for you to enter digits. If the current value is non-zero, that value is displayed in the field on the screen, to be used as a default value.

When INPUT$ is waiting for you to enter a numeric value, you can type the RETURN key to accept the default value. If no default value is displayed and you type the RETURN key, you will be entering a value of zero. If you begin to type digits, they will enter the field from the right end in "calculator style". Using backspace will cause the displayed value to be popped to the right, deleting the last digit you typed. One digit will be deleted each time you type backspace.

If you type a period, enough zeroes will be automatically entered to reach the displayed decimal point. This makes the digits you typed before the period into an integer. Then as you continue to type digits they will be appended after the decimal point. If you type more fractional digits than can be seen in the displayed field, they do become part of the input value; you just cannot see them on the screen. The value on the screen is rounded up if necessary.

A control-X will erase everything you have typed in the current field and allow you to start over. A control-C will immediately BREAK, stopping the program.

If you type a backspace when there are no digits remaining in a field, DP18 will attempt to go back to the previous field in the same picture. This will only work if the screen has not scrolled during the development of the picture, and requires a little bit of planning. (Isn't that what programming is all about?)

Probably it is time for an example.

```
      100 &DP: INPUT $ "HV>>'ENTER X:  '###.#/
          'ENTER Y:  '###.#",X(0),Y(0)
```

Remember how to read pictures from last month's article? The "H" all by itself sets the horizontal cursor position to 0 (beginning of the line). Likewise, "V" sets us to the top line. The ">>" clears from cursor to end of screen. Therefore the "HV>>" does the same thing as a normal HOME command, but from within a picture. The string between apostrophes is printed on the screen. Then "###.#" defines a numeric field, corresponding to the variable X(0). The "/" causes a carriage return to be displayed, and then "ENTER Y:" and the second field.

During execution you will first see the screen clear and the top line become "ENTER X:  ___._" followed by a flashing cursor.  You can type digits, a sign, a decimal point, backspace, and so on.  When you finally type the RETURN a second line will appear:  "ENTER Y:  ___._". If you then type a backspace, the cursor will move back to the first line, displaying as a default value whatever you left in that line.

And what about string fields in the INPUT$ command?  Again, underlines will be displayed for each position of the string field.  If the string already is non-null, its current value will be displayed as a default.

The code that follows is, as has been our practice throughout the DP18 series, preceded by some .EQ lines to define routines previously published, or part of the Apple ROMs.  Variable storage is also defined.  In the integrated source all these definitions are only done once, and the whole program is assembled together.

When the main execution loop of DP18 encounters the INPUT token, we land at line 1840.  Lines 1850-1860 get the character following INPUT, and abort with SYNTAX ERROR if that character is a colon or end-of-line token.  Lines 1870-1910 handle INPUT$, by merely loading up zero in the A-register and jumping to PRINT.INPUT (which was listed last month as part of the PRINT USING code).  The zero value will be stored in a flag, indicating to PRINT.INPUT later on that it was called from INPUT$ rather than PRINT$.  When the picture processor encounters a numeric or string field description in the picture either INPUT.NUM or INPUT.STR will be called, rather than PRINT.NUM or PRINT.STR.

Lines 1930-2510 handle the normal INPUT and INPUT# modes.  The character which follows INPUT is stored at INPUT.TYPE, to be checked later.  If that character was "#", line 1960 gets the next character to position properly for scanning optional quote or the variable name. Lines 1970-2120 process the optional quote.  If it is not there, a "?" prompt is used; it it is there, the string itself is printed.  Lines 2090-2110 make a ";" optional after the quote.  Normal Applesoft INPUT requires a semicolon after the quote, but DP18's INPUT makes it optional.  In fact, you could even get by with a whole bunch of semicolons, if you feel like it....

Lines 2140-2190 read a line of text.  If the first character of the line is a control-C, we abort just like Applesoft.  An empty line returns a zero value, using line 2500-2510.

Lines 2210-2270 set up the input line, which begins at $0200 (WBUF), so that it can be scanned using CHRGET, after pushing current TXTPTR value on the stack.  If the INPUT.TYPE was "#", AS.PARSE and DP.EVALUATE convert the expression down to a value.  If not, FIN converts the number string to a value.  I could have used PARSE and EVALUATE regardless, but it would take a lot more time to convert plain numbers that way.  Lines 2400-2430 restore the old value of TXTPTR, so that we can continue scanning the program.

Lines 2440-2480 scan the input variable name, and store the converted value in that variable. Then back to DP18's main loop to get the next command!

If we are processing an INPUT$ statement, chances are good that we will input a number. If so, the picture processor will call on INPUT.NUM at line 2530. WBUF at this time holds the image of the numeric field description, as amplified from the picture. Lines 2540-2600 copy it into IBUF, because we are going to clobber the WBUF version everytime we re-display the value being entered. IBUF is currently assembled as a 256 byte buffer, which is quite extravagant. Probably this is an area where things could be tightened up, if you need the memory space.

The code beyond line 2530 is hard to follow. I am reminded of the original Adventure game, and its twisty little passages, little twisty passages, and so on. I am going to give it a broad brush, and those of you with an intense interest can explore in more detail on your own.

As each digit is typed, it is appended to the numeric value by ACCUMULATE.DIGIT. Then, after refreshing the picture of the field from IBUF, the value is reconverted to display format and shown on the screen. It may sound inefficient, but it all works nicely. Trimming off digits when backspace is typed is done by truncating the DP18 value and then redisplaying.

LAST.FLD is the routine that tries to back up input to a previous field when you type backspace beyond the first digit. At the beginning of each field, all the necessary parameters are pushed on DP18's stack. LAST.FLD pops these back to move to a previous field. Guess what ... I forgot to check for stack overflow in the STACK.IT subroutine. Should be no problem, however, because only five bytes are stacked for each field, there is room for 24 fields. Since a picture must necessarily be less than 256 characters (maximum length of an Applesoft string) thereby limiting the number of fields, it is unlikely that you will have more than 24 fields stack up. If you think it important to have more, you had better increase the size of STACK.

String input is handled in an analogous fashion by INPUT.STR, starting at line 4970.

As I mentioned before, this is my final article on DP18. But maybe not, if you want more. Some of you might send improvements, corrections, or whatever, and I might pass them along in these pages.

DP18 works, and works well; we're proud of it. You can use DP18 in your programs, even those you plan to sell. Just give us credit where appropriate in your documentation. Remember, you can get all the source code already typed in and integrated together from us for only $50.
1

===================================================================
DOCUMENT :AAL-8502:Articles:Front.Page.txt
===================================================================

$1.80

Volume 5 -- Issue 5                  February, 1985

In This Issue...



65816 News -- Talked with Bill Mensch a few days ago, and he expects full production in just a few weeks.  There should be a lot of sources soon.  Bill has a few more great chips in mind, upgrading the 6502 family even further.

David Eyes is writing a detailed programmer's reference manual for the 65816, to be published about July by Brady.  Bill says it should answer all our questions.  I'll be reviewing it as soon as possible.

We hear of a 6MHz 65816 board with 256K RAM for plugging into Apples. Let you know when we learn more details.

Woz News -- We hear Steve, Wendell Sander (/// designer), and Joe Ennis (//c designer have teamed up to form a new enterprise, outside Apple, with plans to produce a device for the home video market.

Apple II Forever College -- If you would like in-depth training in Cupertino, $500 buys 3 days under the masters.  One session starts March 6th, another May 8th.  Call Marian Djurovich at (408) 973-6411 for details.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

```
=================================================================
DOCUMENT :AAL-8502:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0.....................................$100
   Version 2.0 Upgrade Kit (for 1.0/1.1/1.2 owners)...................$20
   Source Code for Version 1.1 (on two disk sides)...................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility for S-C Macro (without source code).......$20
   Source Code for S-C Cross Reference Utility.............additional $50
DISASM Dis-Assembler (RAK-Ware).......................................$30
   Source Code for DISASM.................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                   Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains        1980       -         -         -         1
 the source code from      1981       2         3         4         5
 three issues of AAL,      1982       6         7         8         9
 saving you lots of        1983      10        11        12        13
 typing and testing.       1984      14        15        16        17

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)..$40
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)........................ package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                           or $25 per 100
   (These are cardboard folders designed to fit into 6"X9" Envelopes.)
Envelopes for Diskette Mailers............................  6 cents each
quikLoader EPROM System (SCRG)..............................($179) $170
D MAnual Controller (SCRG)...................................($90)  $85
Switch-a-Slot (SCRG)........................................($190) $175
Extend-a-Slot (SCRG).........................................($35)  $32
PROmGRAMER (SCRG).........................................($149.50) $140
Write Guard Disk Mod Kit (Mark IV)....................................$40

Books, Books, Books..........................compare our discount prices!
   "Inside the Apple //e", Little...........................($19.95)  $18
   "Apple II+/IIe Troubleshooting & Repair Guide", Brenner..($19.95)  $18
   "Apple ][ Circuit Description", Gayler...................($22.95)  $21
   "Understanding the Apple II", Sather.....................($22.95)  $21
   "Understanding the Apple //e", Sather  (avail. April, price about $21)
   "Enhancing Your Apple II, vol. 1", Lancaster.............($15.95)  $15
                  Second edition, with //e information.
   "Enhancing Your Apple II/IIe, vol 2", Lancaster ....(May, $19.95)  $19
   "Assembly Cookbook for the Apple II/IIe", Lancaster......($21.95)  $20
```

```
"Incredible Secret Money Machine", Lancaster..............($7.95)   $7
"Beneath Apple DOS", Worth & Lechner.....................($19.95)  $18
"Beneath Apple ProDOS", Worth & Lechner..................($19.95)  $18
"What's Where in the Apple", Second Edition..............($19.95)  $19
"6502 Assembly Language Programming", Leventhal..........($18.95)  $18
"6502 Subroutines", Leventhal............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster.......($9.95)   $9
"Microcomputer Graphics", Myers..........................($12.95)  $12
"Apple II Assembly Language", DeJong.....................($15.95)  $15
"Apple II Applications" (Interfacing & I/O experiments)..($13.95)  $13
"Apple Programmer's Handbook", Paul Irwin................($22.95)  $21
"Electronically Hearing: Computer Speech Recognition.....($13.95)  $13
"Electron. Speaking: Computer Speech Generation (Cater)...($14.95) $14
"Assem. Lang. for Applesoft Programmers", Finley & Myers..($16.95) $16
```

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

=============================================================================
DOCUMENT :AAL-8502:Articles:Preshift.Tables.txt
=============================================================================

Building Hi-Res Pre-Shift Tables.............Gianluca Pomponi
Pisa, Italy

Given my interest in everything related to graphics, I read eagerly Bob's article "Generating Tables..." in the Dec 94 issue of AAL. I haven't yet had the chance to read the Apple Supplement of Byte (my local newsstand receives it discontinuously); however, I had already heard about the use of preshift tables in animation. I experimented with this technique some time ago, getting excellent results in moving colored shapes against some very complex backgrounds with relatively simple code.

Maybe one of the most challenging steps is typing in the preshift tables. Writing a program to generate the tables is not difficult, and is probably better. The code that follows only takes $68 bytes as a subroutine, using two page zero variables. And it only takes 24 milliseconds to generate the tables, which is many times faster than reading them from a disk.

The Byte article used 14 tables of 256 bytes each. They correspond to left and right portions of each possible 8-bit value shifted any amount from 1 to 7 bits. No columns are kept in memory for shifting 0 bits, as the result is entirely too predictable.

Since, in hi-res graphics, the high bit does not get shifted, you can deal with it separately. Before looking up the preshifted values you can split off the high bit and rejoin it later. The extra code for this is very minor, and it results in a vast memory saving. By doing it this way we get by with 12 tables of 128 bytes each (six pages instead of 14!). Six tables for the left side results and six for the right, for every possible shift of from 1 to 6 bits, for every possible value from $00 to $7F.

I sometimes find it worthwhile to limit the quotient-remainder tables such as Bob generated in the December article to only 256 bytes each (instead of 280), using code like the following to read them when the X-coordinate is larger than 255:

```
       LDX XCOORD      low byte of xcoord
       LDA QUO+4,X
       CLC
       ADC #$24
       STA XBYTE
       LDA REM+4,X
       STA XBIT
```

Here now is my program to generate the preshift tables, as modified by Bob. LInes 1080-1210 allocate space for the 12 tables, each 128 bytes long.  I put them at $0900 for this example, but of course you can put them wherever you wish.

Lines 1230-1310 are a macro definition.  The macro is called out six times in the main loop, once for each shift of a value.  For the benefit of those without a macro assembler, I have shown the expansion in the listing of lines 1430-1480.  Some of the code in the macro could have been handled by a subroutine, but it would save a negligible amount of space at a cost of an non-negligible amount of time.

The shifting algorithm is familiar to those of you who have been fiddling with hi-res for a while.  Remember that the picture bits are stored backwards in each byte, so that shifting the picture on the screen right one bit requires shifting the bits in memory left within each byte, stepping over bit 7, and from byte to byte in a left-to-right direction.

The little program called TIME, lines 1530-1660, calls the BUILD program 1000 times.  I ran it and clocked it at a little less than 24 seconds, which means building once took less than 24 milliseconds. The tables would take up six disk sectors if they were stored part of the program on disk.  The disk spins at 300 rpm, or 200 milliseconds per revolution.  The absolute minimum time to read six sectors would be 67.5 milliseconds, but in actual practice it takes closer to a half second.  It depends on whether it is part of a larger file or stored as a separate file, the latter taking longer.  Since the program only needs to be executed once, even the memory it occupies it available to the program for other purposes.

```
==============================================================================
DOCUMENT :AAL-8502:Articles:Q.n.A.txt
==============================================================================
```

Questions and Answers

I have just finished installing Version 2.0 of your assembler, and I have a few questions.

a. First, how is the line length of the escape-L changed? The short line looks ridiculous on an 80-column screen. I would also like to change the first character from "*" to ";".

b. How can I get the assembler to initialize things with DOS's MON CI modes set?

c. In working with big programs, it is easy to exceed line number 9999. It happens all the time. As new lines get added, the formatting of lines around 9999 goes haywire, as the spacing is done according to the line number at the time of entry. Thus when a line number changes from 4 to 5 digits or vice versa due to renumbering the opcode and operand columns no longer line up properly. What can be done about the erratic column alignment?

d. I noticed that the symbol table generated by an assembly takes more memory with version 2.0 than it did with 1.1. Why?

e. There appear to be two errors in the sample program S.INLINE on the Macro 2.0 disk. The comment on how to use it shows a comma between the &INPUT and the string variable, when the program in fact requires that there be NO comma. Then, the first line of the main routine does a CMP, which should be an LDA. With these corrections, the program is great. &INPUT will accept input from keyboard or disk, and reads the complete record including commas, quotes, and colons. This I find rather useful.

Mike Lawrie, South Africa

a. The routine which generates the star-dash line starts at $DB21, with the following:

```
       TXA
       BEQ ...
       LDA #$AA        change to $BB for ";"
       JSR ...
       LDA $D01E   ("-" CHAR)
       CPX #$26        increase as you like
```

For example, I changed mine just now like this:

```
       $C083 C083 DB25:BB N DB2D:46
```

b.  Whatever selections you have turned on with the MON command are turned off by the DOS "INT" or "FP" commands.  I guess if you want the MONCI modes all the time you could add code to the assembler to set the proper bits inside DOS.  The flags are in $AA5E:  C=$40, I=$20, O=$10.  Store $60 into $AA5E to effect MONCI.

c.  I agree with you that it is annoying the way the columns stagger when the line numbers are near 9999.  There are several possible solutions.  One solution, is to start line numbers at 10000.  You can do this by changing the code at $D32B:

```
       LDA #990       change to #9990
       STA ...
       LDA /990       change to /9990
```

A better way is to make a the line numbers always print with five digits.  To effect this, change the code at $DE63:

```
       LDX #3  change to LDX #4

       $C083 C083 DE64:4
```

d.  The symbol table does indeed take more space in version 2.0 than it did in previous versions.  This is due to the fact that symbols can have values up to 32-bits long.  Every symbol has two more bytes in the table now.

e.  Right on both counts.  Disks with serial numbers 1186 and larger have the corrections you give.

Is there any way of loading a program from the monitor (without going back to Basic) or reload DOS or reboot without losing what is in memory?

Munson Compton, Shreveport, LA

If you entered the monitor via CALL-151 from Basic, or MNTR or MGO-151 from the S-C Macro Assembler, DOS is still alive and will still respond to commands.  You can BLOAD or LOAD a program, but of course using LOAD will flip you into either Applesoft, Integer BASIC, or the Macro Assembler depending on file type and what languages are around. If you want to stay in the monitor after the LOAD file has been read into memory, you could temporarily patch the DOS LOAD code which starts at $A413.  The book "Beneath Apple DOS" would be helpful here. It looks to me like you could so subvert type A files by patching the JMP ($9D60) at $A44D to RTS (by putting 60 at $A44D).  Type I files might be tricked by putting an RTS (60) at $A5AF.  I don't know what other ramifications these patches might have.  Beware!!!

You can reboot a slave disk without losing the actual text of an assembler source file from memory.  However, the pointer which tells the assembler where the program starts will be reset.  Before rebooting, record the value stored in $CA and $CB, and after getting back into the assembler restore those two bytes.  Of course, if the assembler is in the language card rebooting DOS marks it as not being there. From the monitor you can put it all back by typing:

```
       ]CALL-151
       *C081 C081 E000:20
       *INT
       :$CA:...(whatever values you recorded earlier)
       :LIST   (Voila!)
```

I have the Apple ToolKit and the Big Mac assemblers, and use them primarily to key in source files from articles such as yours. I've figured out how to transpose most of the different labels and opcodes, but would like some enlightenment on the use of the .1, .2, .3 etc. labels that are repeated in the code. I assume this is a capability of your assembler that others don't have.

David Roberson

For help in converting our listings to other assemblers and vice versa, you should refer to my "Directory of Assembler Directives" article in the September 1982 AAL. You are correct in assuming that most other assemblers do not have the kind of local labels as the S-C assemblers, but some do. These numeric labels are one or two digits after a period, and are very convenient for branch points within a sub- routine. They are defined below a normal label, and are only accessible within that area. The local labels are defined internally relative to the preceding normal label, and must be within a 255-byte range after the normal label. Once a new normal label is defined, a whole new set of local labels is available. The use of local labels simplifies programming, because there is no need to think up dozens of unique names like LOOP1, LOP2, LUPA, LUPB, and so on. Local labels also encourage writing good modular code, with only one entry point per module, since the local labels are not accessible outside the routine in which they are defined.

The LISA assembler uses a different type of numeric label, which I call a near-by label. These are redefinable at will, and when they are referenced a pointer must be included to tell the assembler which direction to search for the definition. You can refer to the nearest definition in either a forward or backward direction. I get thoroughly confused trying to read and/or modify programs using these.

.
.
.
.
.
.
.
.
.
1

```
================================================================================
DOCUMENT :AAL-8502:Articles:Symbol.Pgm.Crx.txt
================================================================================
```

Correction for Symbol Table Source Maker...Bob Sander-Cederlof

I went to great lengths to verify the address of the entry into RENUMBER used by Peter's and Bruce's program, and the day after picking up the printed newsletters Bill discovered that I had used a pre-release copy of Version 2.0.  The correct address in the actual release is different.  The correct line 1060 for the version we are sending out is:

```
       1060 RENUMBER .EQ $D658   for the D000 version
    OR 1060 RENUMBER .EQ $1658   for the 1000 version
```

In any case, just be sure the address is the location of the CPX #$06 instruction.

==============================================================================
DOCUMENT :AAL-8502:Articles:WriteGuard.txt
==============================================================================

Write Guard Disk Modification Kit

Mark IV Designs (Mark Hansen) has come up with a neat way to override the write protect switch in a disk drive. Sometimes you want to write on the back side of a disk, in spite of all good breeding. Yet it is a nuisance to have to cut a notch in the other edge of the disk. We finally bought a hole punch, but it is still a nuisance. Other times you want to write protect a disk, but not put one of those little sticky things over the existing notch. What to do?

Instructions for adding an external toggle switch in series or in parallel with the internal sensor are easy to come by, but who wants to drill holes and solder? The Write Guard kit from Mark IV Desings accomplishes all you could wish for without any drilling, cutting, or soldering.

You get a small (1x2x3 inches) box with three-position toggle switch and LED. A short flat cable runs out the back, and you plug that into a socket inside the disk drive (after removing the 74125 from that socket). A piece of velcro attaches the plastic box to either side of your drive. The switch selects normal, always protected, or always unprotected. The LED lights whenever the disk is not protected. One chip on the disk controller card also is replaced with a chip from the kit.

If this kit sounds like something you have been waiting for, you can order one from us at $40.

==============================================================================
DOCUMENT :AAL-8502:Articles:YostsFreeOffer.txt
==============================================================================

Patches Available for Time/Date in Titles...........R. M. Yost

I have implemented a patch to include a Thunderclock (or compatible) time string in the .TItle for version 2.0 of the S-C Macro Assembler. The patch program automatically loads the assembler and my favorite I/O driver, installs the time patch and several others I like, and writes the assembler back on the disk.  The new file includes both assembler and driver, with the patches, as well as a loader which allows the whole thing to be executed with a single BRUN.

I will gladly send a listing of the source code to any Assembly Line reader who is interested.  Just send a stamped self- addressed envelope to R.M.Yost, 7436 Pointe, Canton, MI 48187.

```
==========================================================================
DOCUMENT :AAL-8502:DOS3.3:S.Bld.PreShft.txt
==========================================================================

 1000  *SAVE S.BUILD.PRESHIFT.TABLES
 1010  *--------------------------------
 1020  *   WRITTEN BY G. L. POMPONI, PISA, ITALY
 1030  *   MODIFIED BY BOB SANDER-CEDERLOF
 1040  *--------------------------------
 1050  L.BYTE .EQ 0
 1060  R.BYTE .EQ 1
 1070  *--------------------------------
 1080              .OR $900
 1090  SHIFT.1     .BS 128
 1100  SHIFT.2     .BS 128
 1110  SHIFT.3     .BS 128
 1120  SHIFT.4     .BS 128
 1130  SHIFT.5     .BS 128
 1140  SHIFT.6     .BS 128
 1150  *--------------------------------
 1160  REMND.1     .BS 128
 1170  REMND.2     .BS 128
 1180  REMND.3     .BS 128
 1190  REMND.4     .BS 128
 1200  REMND.5     .BS 128
 1210  REMND.6     .BS 128
 1220  *--------------------------------
 1230         .MA SHIFT
 1240         ASL L.BYTE
 1250         ROL R.BYTE
 1260         LDA L.BYTE
 1270         LSR
 1280         STA SHIFT.]1,X
 1290         LDA R.BYTE
 1300         STA REMND.]1,X
 1310         .EM
 1320  *--------------------------------
 1330         .OR $800
 1340  *--------------------------------
 1350  BUILD.PRESHIFT.TABLES
 1360         LDX #0       FOR X = 0 TO $7F
 1370  *--------------------------------
 1380  .1     STX L.BYTE
 1390         LDA #0
 1400         STA R.BYTE
 1410         ASL L.BYTE
 1420  *--------------------------------
 1430         >SHIFT 1
 1440         >SHIFT 2
 1450         >SHIFT 3
 1460         >SHIFT 4
 1470         >SHIFT 5
 1480         >SHIFT 6
```

```
1490  *--------------------------------
1500         INX          NEXT X
1510         BPL .1       (...UNTIL $80)
1520         RTS
1530  *--------------------------------
1540  *   BUILDS 1000 TIMES IN LESS THAN 24 SECONDS,
1550  *      SO LESS THAN 24 MILLISECONDS TO BUILD ONCE
1560  *--------------------------------
1570  TIME   LDA #4       4*250 = 1000
1580         STA $500
1590  .1     LDY #250
1600  .2     JSR BUILD.PRESHIFT.TABLES
1610         DEY
1620         BNE .2
1630         DEC $500
1640         BNE .1
1650         RTS
1660  *--------------------------------
```

```
===========================================================================
DOCUMENT :AAL-8502:DOS3.3:S.DOSLESS.INIT.txt
===========================================================================
```

```
 1000  *SAVE S.DOSLESS INIT
 1010  *--------------------------------
 1020  RWTS       .EQ $03D9
 1030  GETIOB     .EQ $03E3
 1040  *--------------------------------
 1050  VTOC       .EQ $B3BB
 1060  V.VOLUME   .EQ $B3C1
 1070  V.NXTTRK   .EQ $B3EB
 1080  V.DIRECT   .EQ $B3EC
 1090  V.BITMAP   .EQ $B3F3
 1100  *--------------------------------
 1110  CATALOG.BUFFER .EQ $B4BB
 1120  C.TRACK        .EQ $B4BC
 1130  C.SECTOR       .EQ $B4BD
 1140  *--------------------------------
 1150  R.PARMS    .EQ $B7E8
 1160  R.VOLUME   .EQ $B7EB
 1170  R.TRACK    .EQ $B7EC
 1180  R.SECTOR   .EQ $B7ED
 1190  R.BUFFER   .EQ $B7F0,B7F1
 1200  R.OPCODE   .EQ $B7F4
 1210  *--------------------------------
 1220         .OR $800
 1230  *--------------------------------
 1240  DOSLESS.INIT
 1250         LDA #1       INIT AS VOLUME 001
 1260         STA R.VOLUME
 1270         STA V.VOLUME
 1280  *--------------------------------
 1290         LDA #$04     INIT OPCODE FOR RWTS
 1300         JSR CALL.RWTS.OP.IN.A
 1310  *---MAKE A GENERIC VTOC----------
 1320         LDA #$11
 1330         STA V.NXTTRK
 1340         STA R.TRACK
 1350         LDY #1
 1360         STY V.DIRECT      FORWARD DIRECTION
 1370         DEY          Y=0
 1380         STY R.SECTOR
 1390  *---PREPARE BITMAP---------------
 1400         LDY #4*35
 1410  .1     LDA #0
 1420         DEY
 1430         STA V.BITMAP,Y
 1440         DEY
 1450         STA V.BITMAP,Y
 1460         DEY
 1470         LDA #$FF
 1480         STA V.BITMAP,Y
```

```
1490        DEY
1500        STA V.BITMAP,Y
1510        BNE .1
1520        STY V.BITMAP      CANNOT ALLOCATE TRACK 0
1530        STY V.BITMAP+1
1540        INY                  Y=1, RESERVE F...9
1550        STY 4*17+V.BITMAP         FREE SECTOR 8
1560        LDA #$FE                  RESERVE 0
1570        STA 4*17+V.BITMAP+1       FREE 7...1
1580 *---WRITE VTOC ON NEW DISK-------
1590        LDA #VTOC
1600        STA R.BUFFER
1610        LDA /VTOC
1620        STA R.BUFFER+1
1630        LDA #2            RWTS WRITE OPCODE
1640        JSR CALL.RWTS.OP.IN.A
1650 *---PREPARE CATALOG SECTOR-------
1660        LDX #$00
1670        TXA
1680 .2     STA CATALOG.BUFFER,X
1690        INX
1700        BNE .2
1710 *---WRITE CATALOG CHAIN----------
1720        LDA #CATALOG.BUFFER
1730        STA R.BUFFER
1740        LDA /CATALOG.BUFFER
1750        STA R.BUFFER+1
1760        LDA #17      TRACK 17
1770        LDY #15      START IN SECTOR 15
1780 .3     STA C.TRACK
1790 .4     STY R.SECTOR
1800        DEY
1810        STY C.SECTOR
1820        JSR CALL.RWTS
1830        LDY C.SECTOR
1840        CPY #9
1850        BNE .4
1860        STY R.SECTOR
1870        LDY #0
1880        STY C.TRACK
1890        STY C.SECTOR
1900        JSR CALL.RWTS
1910 *---WRITE BOOT SECTOR------------
1920 .5     LDA #BOOTER
1930        STA R.BUFFER
1940        LDA /BOOTER
1950        STA R.BUFFER+1
1960        LDA #0
1970        STA R.TRACK
1980        STA R.SECTOR
1990        JSR CALL.RWTS
2000        RTS
2010 *--------------------------------
2020 CALL.RWTS.OP.IN.A
```

```
2030        STA R.OPCODE
2040 CALL.RWTS
2050        JSR GETIOB
2060        JSR RWTS
2070        BCS .1       ERROR
2080        RTS
2090 .1     BRK
2100 *--------------------------------
2110 BOOTER
2120        .PH $800
2130 BOOTER.PHASE
2140        .HS 01
2150        LDA $C088,X  MOTOR OFF
2160        LDY #0
2170 .1     LDA MESSAGE,Y
2180        BEQ .2
2190        JSR $FDF0
2200        INY
2210        BNE .1
2220 .2     JMP $FF59
2230 *--------------------------------
2240 MESSAGE
2250        .HS 8D8D8787
2260        .AS -/NO DOS IMAGE ON THIS DISK/
2270        .HS 8D8D00
2280        .EP
2290 *--------------------------------
2300        .BS 256,0
2310 *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8502:DOS3.3:S.DP18.INPUT.txt
==========================================================================

```
*SAVE S.DP18 INPUT
*--------------------------------
*    APPLESOFT SUBROUTINES
*--------------------------------
AS.INLIN     .EQ $D52E   READ A LINE
AS.PARSE     .EQ $D559   PARSE INPUT BUFFER
AS.BREAK     .EQ $D863   CTRL-C BREAK
AS.ADDON     .EQ $D998   ADD (Y) TO TXTPTR
AS.COUT      .EQ $DB5C    PRINT A CHARACTER
AS.CHKCOM    .EQ $DEBE    CHECK FOR COMMA
AS.SYNERR    .EQ $DEC9    SYNTAX ERROR
AS.GETSPA    .EQ $E452
AS.MOVSTR    .EQ $E5E2
*---------------------------------
*   MONITOR SUBROUTINES
*---------------------------------
MON.RDKEY    .EQ $FD0C
MON.LF       .EQ $FC66
*---------------------------------
*      DP SUBROUTINES PRINTED ELSEWHERE
*---------------------------------
DP.NEXT.CMD         .EQ $FFFF
DP.EVALUATE         .EQ $FFFF
MOVE.DAC.YA         .EQ $FFFF
DP.VTAB             .EQ $FFFF
DP.INT              .EQ $FFFF
DP.FALSE            .EQ $FFFF
MOVE.DAC.TEMP1      .EQ $FFFF
MOVE.TEMP1.DAC      .EQ $FFFF
PRINT.INPUT         .EQ $FFFF
FIN                 .EQ $FFFF
GET.A.VAR           .EQ $FFFF
CHECK.DP.VAR        .EQ $FFFF
MOVE.YA.DAC         .EQ $FFFF
PRUS.CLEAR          .EQ $FFFF
PRUS.NEXT           .EQ $FFFF
ACCUMULATE.DIGIT    .EQ $FFFF
PRT.NUM.1           .EQ $FFFF
PRINT.STR.1         .EQ $FFFF
*--------------------------------
*      PAGE ZERO USAGE
*--------------------------------
AS.VALTYP    .EQ $11
MON.WNDWIDTH .EQ $21
MON.CH       .EQ $24
MON.CV       .EQ $25
AS.FRESPA    .EQ $71,72
AS.CHRGET    .EQ $B1
AS.CHRGOT    .EQ $B7
```

```
TXTPTR      .EQ $B8,B9
P2          .EQ $F9
P1          .EQ $FD      GP POINTER
*--------------------------------
WBUF        .EQ $0200
*-------------------------------
*      WORK AREAS FOR DPFP
*-------------------------------
DECFLG             .BS 1
DAC.EXPONENT       .BS 1
DAC.SIGN           .BS 1
IBUF               .BS 256
STACK.PNTR         .BS 1
STACK              .BS 12*10
W                  .BS 1
D                  .BS 1
OLD.W              .BS 1
OLD.D              .BS 1
DGTCNT             .BS 1
INPUT.TYPE         .BS 1
FOUND.NUM          .BS 1
FOUND.STR          .BS 1
FOUND.LEN          .BS 1
FOUND.CHAR         .BS 1
FILL.CHAR          .BS 1
ZERO.CHAR          .BS 1
FLD.FLAG           .BS 1
FLD.START          .BS 1
TEMP               .BS 2
RESULT             .BS 2
DEFAULT.FLAG       .BS 1
LEN                .BS 1
*--------------------------------
DP.SYN3 JMP AS.SYNERR
*--------------------------------
DP.INPUT
       JSR AS.CHRGET
       BEQ DP.SYN3  ...COLON OR EOL
*---INPUT USING------------------
       CMP #'$'     INPUT USING PICTURE?
       BNE .1       ...NO
       LDA #0       ...YES, SIGNAL "INPUT" AND JOIN
       JMP PRINT.INPUT       "PRINT $"
*---INPUT AN EXPRESSION----------
.1     STA INPUT.TYPE    ="#" IF EXP, ELSE <>"#"
       CMP #'#'          INPUT AN EXPRESSION?
       BNE .2            ...NO
       JSR AS.CHRGET     ...YES, GET NEXT CHAR
.2     LDX #"?"     PROMPT CHAR FOR NO QUOTE
       CMP #'"'     QUOTE?
       BNE .6       ...NO, SIMPLE INPUT
       LDY #0       ...YES, PRINT IT NOW
.3     INY
       LDA (TXTPTR),Y    NEXT QUOTED CHARACTER
```

```
       BEQ DP.SYN3       ...NO CLOSING QUOTE
       CMP #'"'          CLOSING QUOTE YET?
       BEQ .4            ...YES
       JSR AS.COUT       ...NO, PRINT CHARACTER
       BNE .3            ...ALWAYS
.4     JSR AS.ADDON      ADD (Y) TO TXTPTR
.5     JSR AS.CHRGET     SCAN NEXT CHAR
       CMP #';'          ALLOW OPTIONAL SEMICOLON
       BEQ .5            ...KEEP LOOKING TILL NOT ';'
       LDX #$80          NULL PROMPT CHARACTER
*---READ A LINE OF TEXT----------
.6     JSR AS.INLIN      '?' OR NULL PROMPT
       LDA WBUF          CHECK FOR EMPTY LINE
       BEQ .11           ...EMPTY LINE
       CMP #$03          CTRL-C?
       BNE .7            ...NO
       JMP AS.BREAK      ABORT INPUT
*---PARSE THE INPUT LINE---------
.7     LDA TXTPTR        SAVE TXTPTR, WHICH POINTS
       PHA               AT THE PROGRAM
       LDA TXTPTR+1
       PHA
       STX TXTPTR        MAKE TXTPTR POINT AT INPUT BUFFER
       STY TXTPTR+1
       JSR AS.CHRGET     GET FIRST CHAR FROM LINE
       LDY INPUT.TYPE    SEE IF SIMPLE OR EXPRESSIONS
       CPY #'#'
       BNE .8            SIMPLE NUMERIC INPUT
       JSR AS.PARSE      EXPRESSION INPUT, SO PARSE
       LDA #WBUF-1       POINT AT INPUT BUFFER AGAIN
       STA TXTPTR        SO EVALUATE CAN PROCESS THE
       LDA /WBUF-1       PARSED LINE
       STA TXTPTR+1
       JSR AS.CHRGET     SCAN FIRST CHAR
       JSR DP.EVALUATE   EVALUATE THE EXPRESSION
       JMP .9
.8     JSR FIN           SIMPLE NUMERIC INPUT
.9     PLA               RESTORE TXTPTR TO PROGRAM
       STA TXTPTR+1
       PLA
       STA TXTPTR
.10    JSR AS.CHRGOT     GET CURRENT PROGRAM CHAR
       JSR GET.A.VAR     GET INPUT VARIABLE
       JSR CHECK.DP.VAR  MUST BE DP18 VARIABLE
       JSR MOVE.DAC.YA   STORE INPUT VALUE
       JMP DP.NEXT.CMD   ...FINISHED?
*---EMPTY INPUT LINE-------------
.11    JSR DP.FALSE      RETURN VALUE = 0
       JMP .10
*-------------------------------
INPUT.NUM
       LDA #0       TERMINATE STRING IN BUFFERS
       STA IBUF,X
       STA WBUF,X
```

```
2570 .1     LDA WBUF-1,X      COPY STRING TO IBUF
2580        STA IBUF-1,X
2590        DEX
2600        BNE .1
2610        LDA FILL.CHAR
2620        STA TEMP
2630        JSR STACK.IT
2640        JSR AS.CHKCOM     MUST HAVE COMMA
2650        JSR GET.A.VAR
2660        JSR CHECK.DP.VAR
2670        STA RESULT   SAVE ADR OF VARIABLE
2680        STY RESULT+1
2690        JSR MOVE.YA.DAC  MOVE DEFAULT INTO DAC
2700        LDA W
2710        STA OLD.W
2720        LDA #1
2730        STA DEFAULT.FLAG
2740        LDA DAC.EXPONENT  IS DAC 0?
2750        BNE INP.X1   NO
2760 INP.X  JSR INP.ZERO.DAC DEFAULT IS 0 OR CTRL-X
2770 INP.X1 LDA #0
2780        STA FLD.FLAG
2790        STA DGTCNT
2800        STA DECFLG
2810        LDA D
2820        STA OLD.D
2830        LDA #$5F     UNDERLINE
2840        STA FILL.CHAR
2850 INP.NEXT.ZERO.CHAR
2860        STA ZERO.CHAR
2870 *--------------------------------
2880 INP.NEXT
2890        JSR INP.PRINT.NUM PRINT THE NUMBER
2900        JSR MOVE.TEMP1.DAC
2910        JSR MON.RDKEY
2920        AND #$7F
2930        CMP #$0D     RETURN?
2940        BEQ .2       ...YES
2950        LDX DEFAULT.FLAG
2960        BEQ .1       NO DEFAULT
2970        JSR INP.ZERO.DAC IGNORE DEFAULT
2980        CMP #8       BACKSPACE?
2990        BEQ INP.NEXT YES,IGNORE
3000 *---DIGIT------------------------
3010 .1     CMP #'0      SEE IF NUMBER
3020        BCC .4       NO
3030        CMP #'9+1
3040        BCS .4       NO
3050        JSR ACCUMULATE.DIGIT
3060        JMP INP.NEXT
3070 *---CARRIAGE RETURN--------------
3080 .2     LDA DGTCNT   IS NUMBER 0?
3090        ORA DEFAULT.FLAG
3100        BNE .3       NO
```

```
3110        STA DAC.EXPONENT YES,SO ZERO THE EXPONENT
3120 .3     LDA RESULT   GET ADR OF VAR
3130        LDY RESULT+1
3140        JSR MOVE.DAC.YA  PUT IT IN VAR
3150        LDA TEMP     RESTORE ORIGINAL FILL CHAR
3160        STA FILL.CHAR
3170        LDA #'0
3180        STA ZERO.CHAR
3190        JMP INP.PRINT.NUM PRINT THE NUMBER
3200 *                        AND RETURN
3210 *---DECIMAL POINT----------------
3220 .4     CMP #'.      DEC POINT?
3230        BNE .5       ...NO
3240 *      SEC          'CMP' LEFT CARRY SET
3250        ROR DECFLG   FOUND DEC PT
3260        BIT DECFLG
3270        BVS INP.NEXT       TWO DEC PTS.
3280        LDA #$40
3290        CLC
3300        ADC DGTCNT
3310        STA DAC.EXPONENT
3320        LDA #'0
3330        BEQ INP.NEXT.ZERO.CHAR  ALWAYS
3340 *---MINUS SIGN-------------------
3350 .5     CMP #'-      MINUS?
3360        BNE .6
3370 *      SEC          'CMP' LEFT CARRY SET
3380        ROR DAC.SIGN MAKE DAC NEGATIVE
3390        BNE INP.NEXT ...ALWAYS
3400 *---PLUS SIGN--------------------
3410 .6     CMP #'+      PLUS?
3420        BNE .7       ...NO
3430        STA DAC.SIGN PUT POSITIVE VALUE IN SIGN
3440        BEQ INP.NEXT ...ALWAYS
3450 *---CTRL-X-----------------------
3460 .7     CMP #$18     CTRL-X?
3470        BNE .8
3480        LDA OLD.D
3490        STA D
3500        JMP INP.X
3510 *---CTRL-C-----------------------
3520 .8     CMP #$3      CTRL-C?
3530        BNE .9       ...NO, TRY BACKSPACE
3540        JMP AS.BREAK
3550 *---BACKSPACE--------------------
3560 .9     CMP #$08     BACKSPACE?
3570        BNE .17      ...NO, TAKE PATH TO INP.NEXT
3580        LDA DECFLG
3590        BPL .10
3600        LDA DAC.EXPONENT
3610        SEC
3620        SBC #$40
3630        CMP DGTCNT
3640        BEQ .15      REMOVE DEC PT ONLY
```

```
*--------------------------------
.10    LDA DAC.EXPONENT
       PHA          SAVE EXPONENT
       LDA DGTCNT
       CLC
       ADC #$3F
       STA DAC.EXPONENT
       JSR DP.INT   CHOP OFF LAST DIGIT
       LDA DAC.EXPONENT
       BEQ .14      THE NUMBER IS 0, SO RESET EVERYTHING
.11    PLA
       STA DAC.EXPONENT
       LDA DGTCNT
       BNE .12
       JSR LAST.FLD
       JMP INP.NEXT
.12    DEC DGTCNT
       BNE .13
       DEC DAC.EXPONENT
.13    LDA DECFLG
       BPL .16      DELETE BY SHIFT
       BMI .17      ALWAYS
*--------------------------------
.14    LDA DECFLG
       BPL .11
       PLA
.15    LDA #$3F
       SEC
       SBC OLD.D
       ADC DGTCNT
       STA DAC.EXPONENT
       LDA #0
       STA DECFLG
       LDA #$5F
       JMP INP.NEXT.ZERO.CHAR
*--------------------------------
.16    LDA DGTCNT
       BEQ .17
       DEC DAC.EXPONENT
.17    JMP INP.NEXT
*--------------------------------
INP.PRINT.NUM
       LDX #-1      COPY IBUF TO WBUF
.1     INX
       LDA IBUF,X
       STA WBUF,X
       BNE .1
       JSR RESTORE.HV.FROM.STACK
       LDA OLD.W
       STA W
       LDA OLD.D
       STA D
       JSR MOVE.DAC.TEMP1
       LDA DECFLG
```

```
       PHA
       JSR PRT.NUM.1
       PLA
       STA DECFLG
       RTS
*---------------------------------
INP.ZERO.DAC
       PHA
       JSR DP.FALSE PUT 0 IN DAC
       LDA #$40
       SEC
       SBC D        CALCULATE EXPONENT
       STA DAC.EXPONENT
       LDA #0
       STA DEFAULT.FLAG
       PLA
       RTS
*---------------------------------
LAST.FLD
       LDY STACK.PNTR
       DEY
       DEY
       DEY
       DEY
       DEY
       BNE .1
       RTS          FIRST FIELD
.1     PLA          DISCARD JSR LAST.FLD
       PLA               "
       PLA          DISCARD JSR INPUT.NUM
       PLA               "
       PLA          DISCARD Y-REG
       PLA          DISCARD JSR PRT.NUM.IF.NEEDED
       PLA               "
       PLA          DISCARD JSR LOOKUP
       PLA               "
       DEY
       LDA STACK,Y
       STA TXTPTR+1
       DEY
       LDA STACK,Y
       STA TXTPTR
       DEY
       LDA STACK,Y
       PHA          SAVE INDEX INTO PICTURE
       DEY
       LDA STACK,Y
       JSR DP.VTAB
       DEY
       LDA STACK,Y
       STA MON.CH
       STY STACK.PNTR
       PLA          RESTORE INDEX INTO PICTURE
       TAY
```

```
       JSR PRUS.CLEAR
       JMP PRUS.NEXT
*--------------------------------
STACK.IT
       LDY STACK.PNTR
       LDA MON.CH   SAVE WHERE THE FIELD IS
       STA STACK,Y
       INY
       LDA MON.CV
       STA STACK,Y
       INY
       DEC FLD.START
       LDA FLD.START
       STA STACK,Y
       INY
       LDA TXTPTR
       STA STACK,Y  SAVE TXTPTR
       INY
       LDA TXTPTR+1
       STA STACK,Y
       INY
       STY STACK.PNTR
       RTS
*--------------------------------
INPUT.STR
       JSR STACK.IT
       JSR AS.CHKCOM   MUST HAVE COMMA
       JSR GET.A.VAR   GET ADR OF VAR
       LDX AS.VALTYP   STR OR NUM
       BMI .1       OK
       JMP AS.SYNERR   MUST BE STRING
.1     STA P1
       STY P1+1
       LDY #0       GET STRING
       STY DEFAULT.FLAG
       STY FLD.FLAG
       STY LEN
       LDA (P1),Y   LENGTH
       BEQ .3       NULL STRING, SO DO NOTHING
       STA LEN
       INY
       LDA (P1),Y   ADR OF STRING
       STA P2       LO ADR
       INY
       LDA (P1),Y
       STA P2+1     HI ADR
       LDY LEN      GET LENGTH
       DEY
.2     LDA (P2),Y
       STA WBUF,Y
       DEY
       BNE .2
       LDA (P2),Y   MOVE LAST BYTE
       STA WBUF
```

```
5270         INY          Y = 1
5280         STA DEFAULT.FLAG  YES THERE IS A DEFAULT
5290  .3     LDA #WBUF
5300         STA P2
5310         LDA /WBUF
5320         STA P2+1
5330         BNE IS.X1    ALWAYS
5340  *-------------------------------
5350  IS.X   LDA #0
5360         STA LEN
5370  IS.X1  LDA FOUND.LEN
5380         PHA
5390         LDA FOUND.CHAR
5400         PHA
5410         JSR RESTORE.HV.FROM.STACK
5420         LDA #$5F     UNDERLINE
5430         STA FILL.CHAR
5440         LDA LEN
5450         JSR PRINT.STR.1
5460         PLA
5470         STA FOUND.CHAR
5480         PLA
5490         STA FOUND.LEN
5500         CMP LEN
5510         BCC .3       OVERFLOW
5520  *---FIND END OF STRING & PUT CURSOR THERE---
5530         JSR RESTORE.HV.FROM.STACK
5540         CLC
5550         ADC LEN      ADD LENGTH OF STRING
5560  .1     CMP MON.WNDWIDTH LONGER THAN WINDOW?
5570         BCC .2
5580         SBC MON.WNDWIDTH WRAP AROUND
5590         PHA
5600         JSR MON.LF   JUMP DOWN TO NEXT LINE
5610         PLA
5620         JMP .1
5630  .2     STA MON.CH   PUT COLUMN BACK IN CH
5640  *---INPUT A CHAR NOW-------------
5650  .3     JSR MON.RDKEY
5660         AND #$7F
5670  *---CARRIAGE RETURN--------------
5680         CMP #$0D     RETURN?
5690         BNE .5       ...NO
5700         LDA DEFAULT.FLAG
5710         BNE .4       DEFAULT, SO LEAVE IT ALONE
5720         LDA LEN      GET LENGTH
5730         JSR AS.GETSPA MAKE ROOM FOR STRING
5740         LDY #0       MOVE DATA INTO VARIABLE
5750         STA (P1),Y   LENGTH
5760         LDA AS.FRESPA
5770         INY
5780         STA (P1),Y   LO ADDRESS
5790         LDA AS.FRESPA+1
5800         INY
```

```
5810        STA (P1),Y   HI ADDRESS
5820        LDX #WBUF
5830        LDY /WBUF
5840        LDA LEN
5850        JSR AS.MOVSTR
5860 .4     JSR RESTORE.HV.FROM.STACK
5870        LDA #$20     SPACE
5880        STA FILL.CHAR
5890        LDA LEN
5900        JMP PRINT.STR.1   PRINT IT ONE MORE TIME
5910 *--------------------------------
5920 .5     LDX DEFAULT.FLAG
5930        BEQ .6       ...NO DEFAULT
5940        LDX #0
5950        STX DEFAULT.FLAG GET RID OF DEFAULT
5960        STX LEN      NULL STRING
5970        CMP #8       BACKSPACE AND DEFAULT?
5980        BNE .8
5990        JMP IS.X1
6000 *---BACKSPACE--------------------
6010 .6     CMP #8       BACKSPACE?
6020        BNE .8
6030        LDA LEN
6040        BNE .7
6050        JSR LAST.FLD BACKUP A FIELD
6060        JMP IS.X1
6070 .7     DEC LEN
6080        JMP IS.X1
6090 *---CTRL-X-----------------------
6100 .8     CMP #$18     CTRL-X?
6110        BNE .9
6120        JMP IS.X
6130 *---CTRL-C-----------------------
6140 .9     CMP #3       CTRL-C?
6150        BNE .10      ...NO
6160        JMP AS.BREAK
6170 *---CHAR FOR STRING--------------
6180 .10    LDY LEN      NORMAL CHAR,
6190        STA WBUF,Y   SAVE IT
6200        INC LEN
6210        JMP IS.X1
6220 *--------------------------------
6230 RESTORE.HV.FROM.STACK
6240        LDY STACK.PNTR
6250        LDA STACK-4,Y
6260        JSR DP.VTAB
6270        LDA STACK-5,Y
6280        STA MON.CH
6290        RTS
6300 *--------------------------------
```

===============================================================================
DOCUMENT :AAL-8503:Articles:BAP.Correction.txt
===============================================================================

Finding Memory Size in ProDOS..............Bob Sander-Cederlof

On page 6-63 of Beneath Apple ProDOS there is a small piece of code designed to determine how much memory there is:

```
       LDA $BF98
       ASL
       ASL
       BIT 0
       BPL SMLMEM      48K
       BVS MEM128      128K
       ...             otherwise 64K
```

The code will not work.  The BIT 0 will test bits 7 and 6 of memory location $0000, which have nothing whatsoever to do with how much memory is in your machine.  What was intended was to test bits 7 and 6 of the A-register, or in other words bits 5 and 4 of $BF98.  Here is one way you can do that:

```
       LDA $BF98
       ASL
       ASL
       ASL
       BCC SMLMEM      48K
       BMI MEM128      128K
       ...             OTHERWISE 64K
```

Notice that not only does this perform the test correctly, it is also one byte shorter!

If you insist on using the same number of bytes, here is another way to test those bits:

```
       LDA $BF98
       AND #%00110000  ISOLATE BITS 5 AND 4
       CMP #%00100000
       BCC SMLMEM      48K
       BNE MEM128      128K
       ...             OTHERWISE 64K
```

If any of you have discovered any other problems with the sample code in this book, pass them along.

==============================================================================
DOCUMENT :AAL-8503:Articles:Disasm.65816.txt
==============================================================================

A Disassembler for the 65816...............Bob Sander-Cederlof

When I first got my Apple, there were no books around for learning 6502 assembly language.  It took me about 3 months to locate and buy a copy of the 6502 programmer's manual from MOS Technology.  About the same time I found a book by William Barden that briefly covered the 8080, 6800, and 6502.  But the way I really learned the 6502 was by using Woz's L command in the Apple monitor.

Of course there were no printers or printer interfaces around in those days either, so I spent hours upon hours copying 20 lines at a time off the screen.  I wrote down a lot of the monitor, and all of the floating point package and Sweet-16 from the tail end of the Integer BASIC ROM.  Fortunately, Apple has never gotten around to eliminating the fabulous L-command from the monitor.

In fact, they have even augmented it.  The //c version includes patches to allow disassembly of the additional opcodes and address modes of the 65C02.  Since Rak-Ware's DISASM calls on the ROM disassembler to decipher each line of code, the //c version automatically grows to accomodate the 65C02.

Now, what about the 65802 and 65816?  It's about time someone wrote a disassembler for that.  Someone?  Why not me?

It's not easy.  On the one hand there is the pressure of competition. Woz's code is SO compact!  On the other hand, the new chip is SO complex!  It is even ambiguous.  There is absolutely no way for a 65816 disassembler to know whether an immediate-mode instruction is two or three bytes long.  Only by executing the programming, and tracing it line-by-line, can we tell.  And even then, it is possible that a tricky programmer might set up code so that it can be interpreted both ways, depending on other conditions.

To make a long story a little shorter, I did it.  You guessed that of course.  My solution to the ambiguity problem was to put the burden on the person using it.  My solution to the complexity problem was to use extensive tables.  My solution to the competition with Woz was to do my best and let him keep his well-deserved glory.

In fact, I started by carefully analyzing Woz's code.  The trail starts at $FE9E in the monitor ROM.  That short piece of code calls INSTDSP at $F8D0 twenty times to disassemble 20 lines of code.  If you take a peek ahead to my listing, lines 1390-1400 patch the language card copy of the monitor inside the L-command loop, so that instead of calling $F8D0 twenty times it calls my disassembler at $0B67 twenty times.  (If you are using the language card version of the S-C Macro Assembler, there is a copy of the monitor in the language card too.)

BRUNning the 65816 disassembler will install this little patch and toggle the immediate-mode size flag. Thereafter each 800G command will toggle the state of the immediate-mode size flag. In one state this flag causes immediate mode instructions to be disassembled as 2-byte instructions; in the other, 3-byte instructions.

The tables are quite complicated, and difficult to type in accurately. Therefore I used macros and let the S-C Macro Assembler do the dirty work. The first table starts at line 1500, and consists of the packed names of the single byte opcodes. The macro at lines 1210-1290 defines how the packing is done. The calling line is of the form ">ON A,B,C,D" where the A, B, and C parameters are the three letters of the opcode name. The D parameter is the letter "A" on those opcodes which might also be multiple-byte: ASL, DEC, INC, LSR, ROR, and ROL.

The packing algorithm is almost the same as the one Woz used in the monitor. Each character is represented by five bits, so that three letters take only 15 bits. The macro sets L1, L2, and L3 to the ASCII value (less 64) of the letters of the opcode name. The .SE directive is used for this so that each invocation of the macro can redefine these variables. This compresses the letters from the range $41...5A to $01...1A. Then the .DA line uses multiplication and addition to pack up the compressed letters. Since arithmetic expressions are parsed by the S-C Macro Assembler in a strict left-to-right fashion, "L1*32+L2*32+L3*2" packs them together.

The "ON" macro also generates a label for the opcode name value by using the opcode name, together with the 4th parameter when present. These names are referred to by another table later on.

The second table is just like the first, but with the names of the longer opcodes instead. Notice that ASL, DEC, etc are in this table too, but without the 4th parameter.

The third and fourth tables have 256 entries, one for every possible opcode byte. Each entry is only one byte long, so each table is 256 bytes. Woz used several smaller tables, because the 6502 didn't use every possible opcode value. The 65816 does define an opcode name for every possible value.

The OPINDEX table uses two macros: "OXA" for single byte opcodes, and "OXB" for longer opcodes. Each entry is a pointer to the name in the OPNAMES.A or OPNAMES.B tables. The pointer is divided by two, leaving room for a flag bit which tells which of the two tables the name is in.

The entries in the OPFORMAT table are offsets into the FMTBL. These are all multiples of 2, because the FMTBL entries are two bytes each.

FMTBL contains coded information indicating how many bytes comprise the instruction and operand, and what the address mode looks like in assembly language. The length can be from two to four bytes, and is coded as 1...3 in the last two bits. The rest of the bits tell which special characters to print and where to print the value of the operand bytes. Single byte opcodes don't have any entries in this table.

One more table, the last one: FMTSTR. This defines the meaning of the bits in FMTBL. Note that the characters are the same as the ones in the various comment lines within FMTBL, only in reverse order.

Finally, we get to the code. The 20-line disassembler calls INSTDSP at line 6180. This starts by calling INSDS1 at line 5760. INSDS1 and INSDS2 are kept as defined points because other software sometimes calls these two points. If you wanted to modify Rak-Ware's DISASM, for example, you would probably need these.

Lines 5760-5840 print the address of the next opcode, and "- ". Lines 5850-5860 pick up that opcode byte. If you enter at INSDS2, have the opcode byte already in the A-register. Lines 5870-5980 dig into the tables to get the opcode name, format, and length for single-byte opcodes. Lines 6000-6160 do the same for longer opcodes. The differences for longer opcodes are several: the second opname table is used, the format is gotten from the tables, and the immediate-mode size flag is used to determine the length of immediate mode opcodes.

Lines 6200-6300 print out the 1-4 bytes of the opcode in hex. If there are less than four bytes, enough blanks are printed so that we always end up in the same position. Lines 6310-6400 unpack the opcode name and print it out. If the opcode is single byte, lines 6410-6420 find out and send us back home (we are finished with this line).

Lines 6430-6450 test the format to detect MVP, MVN, and relative address mode instructions. These special cases are handled by lines 6690-7050. All other operand formats are handled by lines 6470-6680. I see now that I could have put lines 6470-6480 back before line 6430, so that the blank separating the opname from the operand was printed before splitting on the mode. Then lines 6700-6710 could be deleted, saving five bytes. Of course line 6720 would then receive the ".9" label.

Lines 6500-6520 shift out one bit at a time of the format bit string. The corresponding index counts down in the X-register from 10 to 0, and picks a format character from FMTSTR to print. After the character is printed, two special cases are looked for. If the character was "#", meaning immediate mode, and if the immediate-mode size flag indicates long immediates, another "#" is printed. If the character was "$", it is time to print the operand in hex, as two, four, or six digits (lines 6620-6650).

Relative addresses may be either 8-bit or 16-bit. Lines 6780-6820 start the computation for 8-bit values, and call on a monitor routine to finish the printing. Lines 6840-6950 do the same for 16-bit relatives. (There are no two-bit relatives here, no matter what the family tree has borne.)

Finally, lines 6970-7050 print out the two bank bytes for the MVP and MVN instructions. This is different from the way you write MVP and

MVN for assembly by the S-C Macro Assembler.  In the assembler you write "MVP addr1,addr2", where both addresses are 24-bit values.  The bank bytes come from the high byte of each 24-bit address.  To be compatible with the assembler I should change lines 6970-7050 to print out "0000" after each bank byte.

It seems like a worthy project for someone to incorporate my program into Rak-Ware's DISASM, or perhaps a new similar product.  If so, that someone should figure out a neat interactive way to control the immediate-mode size flag.  How about it, Bob?

That's enough of that.  The assembly listing of that table expands to about 4 pages, so here's a hex dump of OPNAMES.A and OPNAMES.B (By the way, OPNAMES.B .EQ $881):

And the OPINDEX table runs about 7 pages, so another hex dump:

The assembly listing of OPFORMAT is around a page and a half, so we'll just LIST this one:

For a complete source listing of this program send a legal-size self-addressed envelope with 2 ounces postage.  Or, order Quarterly Disk #18 for $15 to get S.65816.DISASM along with all the rest of the source code from the last three issues on disk.

```
===========================================================================
DOCUMENT :AAL-8503:Articles:DOS.Buffer.Bldr.txt
===========================================================================
```

Shortening the DOS File Buffer Builder.....Bob Sander-Cederlof

Lately I have been looking through DOS for subroutines that can be shrunk. There seem to be a lot of them, or at least I have been lucky in finding some easy ones with little trouble. Elsewhere this month I show how to shrink the numeric input conversion routine, saving enough bytes to make room for a useful new feature.

Yesterday I happened across the file buffer initializer, which starts at $A7D4 and goes up to $A850. Scanning quickly through the code it looked a likely candidate for the shrinking process. If you take a quick peek, you'll see that it starts out with an SEC instruction that is totally unnecessary. Already we have shaved off one byte!

The DOS file buffers are each 595 bytes, linked together with a chain of pointers. There are normally three buffers, starting at $9600, $9853, and $9AA6. (If you have "Beneath Apple DOS", look on page 6-13 for some explanation.) Each buffer contains a 256 byte area for data, another 256 byte area for a track/ sector list, a 30-character filename, a 45-byte working area for the DOS File Manager, and 4 2-byte pointers. There is a two-byte pointer kept at $9D00,9D01 which points at the first character of the filename in the highest buffer. This is normally $9CD3. Here is a picture of the normal three buffers, all chained together:

```
9D00:  9CD3

 9CF7- Link addr               Link addr               Link addr
        ($9A80)                 ($982D)                 ($0000)
 9CF5- Data addr               Data addr               Data addr
        ($9AA6)                 ($9853)                 ($9600)
 9CF3- TSL addr                TSL addr                TSL addr
        ($9BA6)                 ($9953)                 ($9700)
 9CF1- FMW addr                FMW addr                FMW addr
        ($9CA6)                 ($9A53)                 ($9800)

       30-chars
 9CD3- filename          9A80- filename          982D- filename

       45 bytes
 9CA6- FMW area          9A53- FMW area          9800- FMW area

       256 bytes
 9BA6- TSL area          9953- TSL area          9700- TSL area

       256 bytes
 9AA6- Data area         9853- Data area         9600- Data area
```

The file buffer initializer gets called during the boot procedure, and by the MAXFILES command processor. There are two input parameters: the start of buffers address at $9D00, and the number of file buffers at $AA57. The job of the initializer is to fill in the four address values at the top of each buffer, to store a 00 byte in the first character of the filename of each buffer, and to store a new value in the HIMEM variable for the current language. Here's the way it was, without comments.

I rearranged the code, kept mental track of carry status, optimized register usage, and lopped off 11 bytes. Speed is no issue, because it is not a time critical operation anyway, but mine may be a tad quicker. Compare the two versions, and you can learn a few tricks for your own use.

I found it even more interesting to re-write this program using the 65802 capabilities. The 16-bit registers save a lot of byte shuffling, and eliminate the need for TEMP and PNTR. What's more, instead of saving only 11 bytes over the original DOS 3.3 version, this time I whacked out 46 bytes! And it could be made even smaller, if we could make some assumptions about the CPU status.

In general, we don't know whether we are in 65802 or 6502 mode until we peek at the "hidden" status bit (the E-bit). In the process of peeking we may change it, and may also change the M- and X-bits. Lines 1190 save the current status, flip into '802 mode and save the status again. The first PHP is there in case we were already in '802 mode. If we were, it saves the M- and X- bits and they will be restored by the PLP at line 1620. The second PHP saves the status of the mysterious E-bit (the XCE opcode swaps E and C). Lines 1600-1610 pull this saved status and do another XCE, restoring E to what it was when this sub- routine was called. If we could ASSUME that we were called in '802 mode, we could delete lines 1190-1210 and lines 1610-1620 (saving 5 more bytes). Or, if we could be sure we were always called from 6502 mode, we could delete 1190, 1220, and 1620, and change line 1600 to ".4  SEC" (saving 3 bytes). Probably better never to assume, at least until we are a lot more familiar with this marvelous chip.

The XCE instruction swaps the C- and E-bits, but that is not necessarily all. The M- and X-bits always come up in the 8-bit mode after an XCE. Therefore in line 1240, the LDX will load $00 into the high byte of the X-register and the number of buffers into the low byte. In line 1250 I turn on 16-bit mode for both indexing and memory-accumulator operations, and I will keep it that way until the PLP at line 1600.

6502 programs are always full of page zero pointer addressing modes, but in 65802 programs we may see a lot less of them. Now we can load a whole 16-bit address into the X- or Y- register.

```
       Instead of:            We can write:
       -----------            -------------
       LDA BUF.PNTR
```

```
       STA PNTR
       LDA BUF.PNTR+1
       STA PNTR+1
       LDY #$1E                LDY BUF.PNTR
       LDA DATA...             LDA DATA...
       STA (PNTR),Y            STA $1E,Y
```

Lines 1280-1290 zero the first byte of the filename. As an "extra" feature now, the second byte is also zeroed. In lines 1300-1380 I can compute and store the three area pointers in a very straightforward manner. It now occurs to me that by swapping the roles of the X- and Y-registers I could save six more bytes, since the STA $offset,X instructions would assemble in two bytes rather than three. (The only problem might be that the D-register must = $0000 for this to work.)

Since I don't have to use the X-register to hold temporary values during the buffer creation loop, I can use it instead to count buffers. Lines 1400-1410 do the counting.

If we have not just built the last buffer, lines 1420-1460 set the "next buffer" link address and branch back to build another buffer.

Lines 1470-1500 save the address of the data area in the X- register and store 0000 in the link address for the last buffer. The data area address is going to be the new HIMEM value.

Lines 1510-1590 store the new HIMEM value for the currently selected language. If we are in Applesoft, the string area normally bumps against HIMEM; we now empty that area, because HIMEM may have moved. If we are in Integer BASIS or the S-C Assembler (which fools DOS into believing it is I/B), the source program nestles against HIMEM; it is therefore emptied by storing the HIMEM value into PP.

Won't it be nice when we all have 65802's and can USE these new code segments? It may not be as long as you think. In the mean time, maybe we can develop our expertise. And we can carve enough holes in DOS to leave room for some great new features.

==============================================================================
DOCUMENT :AAL-8503:Articles:DOS.Numin.txt
==============================================================================

Improved DOS 3.3 Number Parsing............Bob Sander-Cederlof

Whether Apple knows or not, cares or not, likes it or not, DOS 3.3 is still alive.  And still the system of choice to most of their loyal customers.

The //c ROMs new //e ROMs and patch Applesoft so that lower case keywords and commands can be typed without penalty.  However, since they are promoting ProDOS and do not care about DOS, they did nothing to give it the freedom to accept lower case commands.  I am constantly chafing over the necessity of popping the shift lock key up and down, (down for DOS and up for word processing).  Surely a very small patch would do the trick.

I looked around and found the subroutine DOS uses to pick characters out of the command buffer, at $A193-$A1AD.  Six bytes of new code inserted right before the CMP #$AC at $A1A1 would do it.  By putting a JSR to a patch in place of the STX $AA5D at $A19E, a ten-byte patch subroutine would solve my problem.

But where do I get a ten-byte hole to fit this patch into?  All the holes I know about have already been used now, and I really don't want to eliminate any exisiting features.  The only solution is to find some loosely written code and rewrite it with compactness as the major criterion.

The code to be recoded must be relatively unused.  That is, not likely to be called at internal places by sneaky software.  I found a likely candidate in the number conversion subroutine used in parsing DOS commands.  This subroutine occupies from $A1B9 through $A228.  I ran a cross reference on the outer shell portion of DOS ($9D84-$A883) using Rak-Ware's DISASM program, and verified that there are no entry points into this code except at the beginning.  It is called from only two places, $A0AA and $A127.

Here is a commented disassembly of the subroutine.

<<<code here for Apple's version>>>

Now here is my revised version, which is sixteen bytes shorter.  It is also a little faster, though that is not important.  As far as I can tell, no features are changed.  There is room for my ten-byte lower-case patch and six bytes to spare!

Compare the two versions to see where I found the extra bytes.  Part of the savings was gained by using a better algorithm for reducing an ASCII character to a hex or decimal digit.  Changing the order of the sections of the program saved more bytes, by eliminating JMPs and "branch always" ops.  I kept the same local line numbers in the new version to aid you in locating similar sections.

<<<code for new version>>>

It is always nice to be able to make a self-installing patch, so I dug out the April 83 issue of AAL for Bill Morgan's PATCHER program.  I found the source on a Quarterly Disk, and merged it with the new number parser.  Then I added my lower case patch, and glued it all together.  The listing that follows is the result.  If the program is BRUN it will install the new parser and the lower case filter automatically.

==============================================================
DOCUMENT :AAL-8503:Articles:Front.Page.txt
==============================================================

$1.80

Volume 5 -- Issue 6                    March, 1985

In This Issue...



Videx Ultraterm Driver

We've just completed a Videx Ultraterm display driver for S-C Macro Assembler Version 2.0, so now you fine-print fans can use the assembler with that card's high-density modes.  (My favorite is the 48 x 80 inverse mode.)  As with the other Version 2.0 drivers, complete source code is supplied so you can tailor the card's performance to your tastes.

This driver is included on all Version 2.0 disks after number 1274. Those of you with lower serial numbers can return your original disk for updating.  Please include $1.00 to cover postage and handling.  We have also corrected several minor assembler bugs in the last month, so those of you with serial numbers below 1252 might want to update your disks as well.

Quarterly Disk #18

I'd also like to remind you that AAL Quarterly Disk #18 is now ready. This disk contains all of the source code from the January through March '85 issues, including the final install- ments of DP18 and this month's 65816 disassembler.  That's many hours' worth of typing saved, at a cost of only $15.  Remember that we also sell a year's subscription to the Quarterly disks for only $45.  That's four disks for the price of three!

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

Apple II Computer Info

Inc.)

```
=================================================================
DOCUMENT :AAL-8503:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                  Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains       1980       -         -         -         1
 the source code from     1981       2         3         4         5
 three issues of AAL,     1982       6         7         8         9
 saving you lots of       1983      10        11        12        13
 typing and testing.      1984      14        15        16        17
                          1985      18

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36

Blank Diskettes (Verbatim)......................... package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers.............................  6 cents each

quikLoader EPROM System (SCRG)..............................($179) $170
PROmGRAMER (SCRG).........................................($149.50) $140
D MAnual Controller (SCRG)....................................($90)  $85
Switch-a-Slot (SCRG)........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32
Write Guard Disk Mod Kit (Mark IV)................................  $40

Books, Books, Books.........................compare our discount prices!

    "Inside the Apple //e", Little........................($19.95)  $18
    "Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95)  $18
    "Apple ][ Circuit Description", Gayler..................($22.95)  $21
    "Understanding the Apple II", Sather....................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster............($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
```

```
    "Incredible Secret Money Machine", Lancaster.............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner....................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner.................($19.95)  $18
    "What's Where in the Apple", Second Edition.............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal.........($18.95)  $18
    "6502 Subroutines", Leventhal...........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers.........................($12.95)  $12
    "Assem. Lang. for Applesoft Programmers", Finley & Myers($16.95)  $16
```

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

=============================================================================
DOCUMENT :AAL-8503:Articles:OKI.6203.txt
=============================================================================

The Oki 6203 Multiply/Divide Chip..........Bob Sander-Cederlof

If you really need to multiply or divide in a hurry, the Oki 6203 may the ticket. This device sells for about $7, and can be almost directly connected to the Apple bus. All you need is one inverter and a prototyping board.

Assuming you built a little card with the device on it, with its two address lines connected to Apple's A0 and A1 lines, you could multiply two 8-bit numbers for a 16-bit product like this:

```
       MUL.6203 STA SLOT*16+$C080   1ST OPERAND
                STY SLOT*16+$C081   2ND OPERAND
                LDA #2              MULTIPLY COMMAND
                STA SLOT*16+$C083   COMMAND REGISTER
                NOP                 DELAY FOR RESULT
                LDA SLOT*16+$C081   HI-BYTE OF PRODUCT
                LDY SLOT*16+$C082   LO-BYTE OF PRODUCT
                RTS
```

A very similar program can divide a 16-bit value by an 8-bit value, producing a quotient and remainder. The time for the multiply is only 22 cycles (plus the JSR and RTS if you make a subroutine), and 24 cycles for the divide.

(Please don't try to order the chip from us, because we don't sell chips.)

```
==============================================================================
DOCUMENT :AAL-8503:Articles:Sather.on.65C02.txt
==============================================================================
```

65C02s in Old Apples...............................Jim Sather

I read Andrew Jackson's 12/84 AAL comments on 65C02 operation in an Apple II with interest since I had looked into the same subject while doing research for "Understanding the Apple IIe". I share Mr. Jackson's conclusion that the problem is short read data setup time from motherboard RAM, but I disagree with his analysis and conclusion that a 65C02 only gets a setup of 25 nsec in an Apple II.

The motherboard RAM read data setup time in an Apple II is

```
        70 nsec (one 14M period)
  minus LS174 pin 9 to data out propagation delay (B5/B8 latch)
  minus LS257 data propagation delay (B6/B7 mux)
  minus 8304 or 8T28 data propagation delay (H10/H11 driver)
   plus MPU PHASE 0 to PHASE 2 propagation delay
   plus 74LS08 propagation delay (B11 PHASE 0 gate).
```

Longer PHASE 0 - PHASE 2 delays result in longer read data setup time, not shorter. With the 6502s and 65C02s I have experimented with, PHASE 0 to PHASE 2 delay has always been in the 20-40 nsec region. Whatever the variation, I have found no NCR or GTE 65C02 that will work in my Apple II.

Taking all delays into account, the motherboard read data setup time for a 6502 or 65C02 is about 65 nsec. This is not good enough for 1 MHz 6502/65C02 specifications but it is good enough for 2 MHz 6502/65C02 specifications. In other words, the Apple II does not meet the read data setup spec of the 1 MHz 6502 that it was manufactured with. Based on this fact, the 100 nsec read data setup spec of 1 MHz 6502s is unrealistically conservative.

But why won't a 2 MHz 65C02 run in the Apple II if it requires only 50 nsec setup time and it gets 65 nsec? The answer, in my opinion, is that NCR and GTE 2 MHz 65C02s do not operate to spec. With certain instruction sequences, they require more than 50 (and, in fact, more than 65) nsec read data setup time. The instruction sequences that bomb are VERY limited, so the 65C02 only gets into trouble when a certain few code sequences are executed. The 65C02 symptom in the Apple II is, therefore, that most things work, but some don't.

Efforts to improve 65C02 operation in the Apple II can be concentrated on decreasing data delays (by replacing the LS174s and LS257s with equivalent devices from a faster logic family) or increasing MPU data clock delays (by adding TTL devices in series with the MPU PHASE 0 input). Possible reduction in data delays is limited, so increased MPU PHASE 0 delay is tempting. Be forewarned, though, that 6502 PHASE 2 is already very late for peripheral slot and serial input mux data transfer, and that such data transfer already depends on the long bleed off time of data from the floating data bus. It is certainly feasible that some Apples with heavy data bus loads will begin to show bugs if any MPU PHASE 0 delay is introduced. But in all probability, you can increase the MPU PHASE 0 delay in a given Apple until MPU PHASE 2 falls concurrently with RAM SELECT' after access to an address above $C00F in the Apple II. This point is 60 nsec after peripheral slot PHASE 0 falls in my Apple II.

AAL readers may be interested in the following excerpt from "Understanding the Apple IIe". It details some features of the 65C02 which are not clear from the data sheet and describes instruction sequences that I have found that make NCR and GTE 65C02s bomb in an Apple II. Note particularly that I have a Rockwell 1 MHz 65C02 that operates without a hitch in my Apple II. This may be a lucky coincidence, or Rockwell 65C02s may not have the read data setup problems of the NCR and GTE chips.

[ Following is an excerpt from "Understanding the Apple //e", copyright (c) 1985 by Quality Software, published here by permission of Quality Software. ]

THE 65C02 MICROPROCESSOR

A recent development in the 6502 world has been the introduction of the 65C02 MPU. This MPU (manufactured by NCR, Rockwell, and alternate sources) is fabricated using CMOS technology, instead of the NMOS used in the 6502. The general advantage of CMOS over NMOS is lower power consumption, but the 65C02 also has some new instructions which make it operationally more powerful than its NMOS brother. A 65C02 can execute any 6502 program that doesn't depend on fine instruction execution timing, but a 6502 cannot execute 65C02 programs that utilize the new 65C02 instructions.

Apple uses the 65C02 MPU in the Apple //c microcomputer, and they intend to convert the Apple //e over to the 65C02. The plan is to retrofit older Apple //e's with the 65C02 as part of the firmware upgrade package described in Chapter 6. This will maximize compatiblity betweeen the Apple //e and the Apple //c, and make it possible to write shorter and faster Apple //e assembly language programs. Because the Apple //e may become a 65C02 based computer in the future, some data on the 65C02 is given here and in other parts of "Understanding the Apple //e".

The 65C02 improvements consist of the addition of new instructions and addressing modes, and the removal of some old 6502 bugs. For the most part, differences between the 6502 and 65C02 are well documented in the partial NCR 65C02 data sheet in Appendix C at the back of this book. Descriptions here will therefore be limited to a few points whose ramifications are not made entriely clear by the data sheet. Please note also that details of 65C02 instruction execution are given in Tables 4.3 and 4.4 in an application note later in this chapter.

First, the NCR and Rockwell 65C02s are not identical. The Rockwell chip executes some instructions that are not part of the NCR 65C02 repertoire. These are the zero page instructions RMBn (Reset Memory Bit n) and SMBn (Set Memory Bit n), and the zero page relative branch instructions BBRn (Branch on Bit n Reset) and BBSn (Branch on Bit n Set). The opcodes of these Rockwell instructions ($X7 and $XF) represent NOPs in the NCR chip. Apple appears to be using NCR compatible 65C02s in its computers, but the Rockwell chip works fine in the Apple //e. Please refer to Tables 4.3 and 4.4 for details of the additional Rockwell instructions.

The READY line of a 6502 will not halt the MPU during a write cycle, but the 65C02 READY line will. This raises the question, "what happens to the Apple IIe data bus if READY is pulled low during a write cycle and is held low for a number of following write cycles?" If the 65C02 attempts to control the data bus constantly for a series of wait state write cycles, it will compete with motherboard RAM for control of the data bus near the end of PHASE 1. Investigation shows that this is not a problem. During a long series of wait state write cycles, the 65C02 control the data bus only during that portion of the machine cycle in which it controls the data bus during a normal write cycle. Therefore, its data bus connection is at high impedance during the majority of PHASE 1 in all wait state write cycles, and motherboard RAM is free to control the data bus near the end of PHASE 1.

The fact that interrupts do not cause abortion of a BREAK instruction is listed as an operational enhancement of the 65C02 on page 3 of the data sheet. The data sheet is referring to non-maskable interrupts, not interrupt requests. In a 6502 or 65C02, IRQ' falling after a BREAK op code fetch does not interfere with BREAK execution. However, if NMI' falls after a BREAK op code fetch and before the interrupt vector is fetched in a 6502, then the NMI' interrupt vector is fetched, and the NMI' handler is executed. An RTI at the end of the NMI' handler causes return to the address (plus two) of the BREAK instruction and probable program crashing. This bug is fixed in the 65C02. As the data sheet indicates, NMI' falling during BREAK execution results in NMI' execution after BREAK execution is complete.

The NCR data sheet refers to the new increment accumulator and decrement accumulator instrucions as INA and DEA. I don't know why they do this, because these instructions are clearly just new addressing modes of the INC and DEC instructions. The new mnemonics should be INC A and DEC A or just INC and DEC as given in the Rockwell data sheet. The addition of the INC and DEC accumulator addressing modes means these instructions have all the addressing modes of the other 6502 read-modify-write instructions (ASL, LSR, ROL, and ROR).

Another notable feature of the 65C02 data sheet is the 5000-microsecond maximum cycle time in the AC characteristics table on page 3. I take this to mean that you can stop the clock for a guaranteed minimum of 5000 microseconds with PHASE 0 high, but not with PHASE 0 low. The Rockwell data sheet is more specific about the difference. It states: "The input clock can be held in the high state indefinitely; however, if the input clock is held in the low state longer than 5 microseconds, internal register and data status can be lost". The significance is that, when the Apple IIe DMA' line is held low, it forces the PHASE 0 input to the MPU to a low state. I therefore conclude that long term continuous DMA in the Apple IIe cannot be performed with a 65C02 any easier than it can with a 6502. In either case, long term continuous DMA can only be performed by pulling DMA' low after the MPU has been stopped via READY low, and only after the X4 and X5 Apple IIe motherboard jumpers have been configured so the MPU clock is not stopped when DMA' is pulled low.

A feature of the 65C02 that does not show up in the NCR data sheet is that the new BIT immediate instruction operates differently than BIT in the other addressing modes. In the other addressing modes, BIT sets the negative, overflow, and zero flags based respectively on operand bit 7, operand bit 6, and the result of Accumulator AND operand. The 65C02 BIT immediate instruction affects only the zero flag, not the negative and overflow flags.

A final point about 65C02 operation that I'd like to make is mildly speculative. The 65C02 is pin compatible with the 6502, and was designed as a direct but more powerful substitute for the 6502. To make it work in the Apple IIe, you simply remove the 6502 and plug in the 65C02. However, the 65C02 does not work reliably in the older Apple II. I believe that the reason for this is that the 65C02 (or at least an NCR 65C02) requires read data to be set up longer than a 6502 operating at the same frequency. RAM read data in the Apple II becomes valid at the MPU (about 60 nsec before PHASE 2 falls) much later than it does in the Apple IIe (about 250 nsec before PHASE 2 falls). Whereas the 6502 can handle the short RAM read data set up time, the 65C02 seems to have trouble with it.

I have performed limited experiments with 65C02s in an Apple II. Basically, I found that two NCR 65C02As (2 MHz?) and one NCR compatible GTE G65SC02P-2 (2 MHz) caused intermittent program crashing that got worse as the peripheral card data bus load was increased. The Rockwell R65C02P1 (1 MHz) that I tried caused no program crashes. The NCR 65C02 program drashes occurred only with certain data bus sequences. If an RTS instruction is preceded by a NOP or SBC, and the Apple II video data preceding the RTS opcode fetch is $A0, $A2, or $A9 then the carry flag is set during otherwise normal execution of the RTS instruction. This unwanted setting of the carry flag occurred as mentioned with all three NCR type chips. One of the chips also set the carry flag if the video data preceding the RTS was $89, and another one also set the carry flag if the video data preceding RTS was $89 or $E9. Note that $89, $A0, $A2, $A9, and $E9 are all immediate mode 65C02 instructions.

In these experiments, I did not conclusively prove that the problem with the 65C02 in the Apple II is short set up time of RAM read data. This is merely a highly educated guess upon which I would be willing to bet a paycheck (if only I had one). Setting the data up quicker definitely helps, because the bugs mentioned in the previous paragraph do not exist when the program resides in a 16K RAM card whose read data becomes valid just after Q3 falls during PHASE 0. In any case, I am suspicious of the validity of the NCR claim of 50-nsec minimum read data set up time in its 65C02.

```
================================================================================
DOCUMENT :AAL-8503:DOS3.3:PatchDOS4LC.txt
================================================================================

d∑PATCH DOS FOR LOWER CASE-
n∑N:∑N»0∏∑Kx∑B:∑I»1∑N:∑D:∑B,D:B»B¿1:∑T∑∑110" Â∑
112,41401,160,0,132,68,132,69,32,164,161,240,46,201,164,240,62,73,176,
201,10,176,73,6,68,38,69,101,68,170,152,101,69,72,6,68,38,69,6,68,38,6
9,138,101,68,133,68,104,101,69,133,69,176,42,32,164,161,208·  Ô∑
214,166,68,165,69,24,96,10,10,10,10,162,4,10,38,68,38,69,202,208,248,3
2,164,161,240,231,73,176,201,10,144,231,105,136,201,250,176,225,56,96,
142,93,170,201,224,144,2,41,223,96,0,0,0,0,0,0 ˙     ˙∑
3,41374,32,25,162
\∑0
```

```
==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.65816.DISASM.txt
==============================================================================

 1000   .LIF
 1010         .TI 76,65816 DISASSEMBLER................FEBRUARY 14,
1985...........
 1020  *SAVE S.65816 DISASM
 1030  *---------------------------------
 1040  IMM.SIZE .EQ $00
 1050  LMNEM    .EQ $2C
 1060  RMNEM    .EQ $2D
 1070  FORMATL  .EQ $2E
 1080  LENGTH   .EQ $2F
 1090  FORMATH  .EQ $30
 1100  PCL      .EQ $3A
 1110  PCH      .EQ $3B
 1120  *---------------------------------
 1130  SCRN2  .EQ $F879
 1140  RELADR .EQ $F938
 1150  PRNTAX .EQ $F941
 1160  PRBLNK .EQ $F948
 1170  PRBL2  .EQ $F94A
 1180  PCADJ  .EQ $F953
 1190  CROUT  .EQ $FD8E
 1200  PRBYTE .EQ $FDDA
 1210  COUT   .EQ $FDED
 1220  *---------------------------------
 1230         .MA ON
 1240         .LIST OFF
 1250  L1     .SE ']1-64
 1260  L2     .SE ']2-64
 1270  L3     .SE ']3-64
 1280  *      .LIST ON
 1290  ]1]2]3]4 .DA L1*32+L2*32+L3*2
 1300         .EM
 1310  *---------------------------------
 1320         .MA OXA
 1330         .DA #]1-OPNAMES.A/2+128
 1340         .EM
 1350  *---------------------------------
 1360         .MA OXB
 1370         .DA #]1-OPNAMES.B/2
 1380         .EM
 1390  *---------------------------------
 1400  T      LDA $C083
 1410         LDA $C083
 1420         LDA #INSTDSP
 1430         STA $FE65
 1440         LDA /INSTDSP
 1450         STA $FE66
 1460         LDA IMM.SIZE
 1470         EOR #$FF
```

```
1480        STA IMM.SIZE
1490        RTS
1500 *--------------------------------
1510 OPNAMES.A
1520        >ON A,S,L,A
1530        >ON B,R,K
1540        >ON C,L,C
1550        >ON C,L,D
1560        >ON C,L,I
1570        >ON C,L,V
1580        >ON C,O,P
1590        >ON D,E,C,A
1600        >ON D,E,X
1610        >ON D,E,Y
1620        >ON I,N,C,A
1630        >ON I,N,X
1640        >ON I,N,Y
1650        >ON L,S,R,A
1660        >ON N,O,P
1670        >ON P,H,A
1680        >ON P,H,B
1690        >ON P,H,D
1700        >ON P,H,K
1710        >ON P,H,P
1720        >ON P,H,X
1730        >ON P,H,Y
1740        >ON P,L,A
1750        >ON P,L,B
1760        >ON P,L,D
1770        >ON P,L,P
1780        >ON P,L,X
1790        >ON P,L,Y
1800        >ON R,O,L,A
1810        >ON R,O,R,A
1820        >ON R,T,I
1830        >ON R,T,L
1840        >ON R,T,S
1850        >ON S,E,C
1860        >ON S,E,D
1870        >ON S,E,I
1880        >ON S,T,P
1890        >ON T,A,X
1900        >ON T,A,Y
1910        >ON T,C,D
1920        >ON T,C,S
1930        >ON T,D,C
1940        >ON T,S,C
1950        >ON T,S,X
1960        >ON T,X,A
1970        >ON T,X,S
1980        >ON T,X,Y
1990        >ON T,Y,A
2000        >ON T,Y,X
2010        >ON W,A,I
```

```
2020        >ON W,D,M
2030        >ON X,B,A
2040        >ON X,C,E
2050 *-------------------------------
2060 OPNAMES.B
2070        >ON A,D,C
2080        >ON A,N,D
2090        >ON A,S,L
2100        >ON B,C,C
2110        >ON B,C,S
2120        >ON B,E,Q
2130        >ON B,I,T
2140        >ON B,M,I
2150        >ON B,N,E
2160        >ON B,P,L
2170        >ON B,R,A
2180        >ON B,R,L
2190        >ON B,V,C
2200        >ON B,V,S
2210        >ON C,M,P
2220        >ON C,P,X
2230        >ON C,P,Y
2240        >ON D,E,C
2250        >ON E,O,R
2260        >ON I,N,C
2270        >ON J,M,L
2280        >ON J,M,P
2290        >ON J,S,L
2300        >ON J,S,R
2310        >ON L,D,A
2320        >ON L,D,X
2330        >ON L,D,Y
2340        >ON L,S,R
2350        >ON M,V,N
2360        >ON M,V,P
2370        >ON O,R,A
2380        >ON P,E,A
2390        >ON P,E,I
2400        >ON P,E,R
2410        >ON R,E,P
2420        >ON R,O,L
2430        >ON R,O,R
2440        >ON S,B,C
2450        >ON S,E,P
2460        >ON S,T,A
2470        >ON S,T,X
2480        >ON S,T,Y
2490        >ON S,T,Z
2500        >ON T,R,B
2510        >ON T,S,B
2520 *-------------------------------
2530 OPINDEX
2540 *---0X--------------------------
2550        >OXA BRK
```

```
2560        >OXB ORA
2570        >OXA COP
2580        >OXB ORA
2590        >OXB TSB
2600        >OXB ORA
2610        >OXB ASL
2620        >OXB ORA
2630        >OXA PHP
2640        >OXB ORA
2650        >OXA ASLA
2660        >OXA PHD
2670        >OXB TSB
2680        >OXB ORA
2690        >OXB ASL
2700        >OXB ORA
2710 *---1X---------------------------
2720        >OXB BPL
2730        >OXB ORA
2740        >OXB ORA
2750        >OXB ORA
2760        >OXB TRB
2770        >OXB ORA
2780        >OXB ASL
2790        >OXB ORA
2800        >OXA CLC
2810        >OXB ORA
2820        >OXA INCA
2830        >OXA TCS
2840        >OXB TRB
2850        >OXB ORA
2860        >OXB ASL
2870        >OXB ORA
2880 *---2X---------------------------
2890        >OXB JSR
2900        >OXB AND
2910        >OXB JSL
2920        >OXB AND
2930        >OXB BIT
2940        >OXB AND
2950        >OXB ROL
2960        >OXB AND
2970        >OXA PLP
2980        >OXB AND
2990        >OXA ROLA
3000        >OXA PLD
3010        >OXB BIT
3020        >OXB AND
3030        >OXB ROL
3040        >OXB AND
3050 *---3X---------------------------
3060        >OXB BMI
3070        >OXB AND
3080        >OXB AND
3090        >OXB AND
```

```
3100        >OXB BIT
3110        >OXB AND
3120        >OXB ROL
3130        >OXB AND
3140        >OXA SEC
3150        >OXB AND
3160        >OXA DECA
3170        >OXA TSC
3180        >OXB BIT
3190        >OXB AND
3200        >OXB ROL
3210        >OXB AND
3220 *---4X--------------------------
3230        >OXA RTI
3240        >OXB EOR
3250        >OXA WDM
3260        >OXB EOR
3270        >OXB MVP
3280        >OXB EOR
3290        >OXB LSR
3300        >OXB EOR
3310        >OXA PHA
3320        >OXB EOR
3330        >OXA LSRA
3340        >OXA PHK
3350        >OXB JMP
3360        >OXB EOR
3370        >OXB LSR
3380        >OXB EOR
3390 *---5X--------------------------
3400        >OXB BVC
3410        >OXB EOR
3420        >OXB EOR
3430        >OXB EOR
3440        >OXB MVN
3450        >OXB EOR
3460        >OXB LSR
3470        >OXB EOR
3480        >OXA CLI
3490        >OXB EOR
3500        >OXA PHY
3510        >OXA TCD
3520        >OXB JMP
3530        >OXB EOR
3540        >OXB LSR
3550        >OXB EOR
3560 *---6X--------------------------
3570        >OXA RTS
3580        >OXB ADC
3590        >OXB PER
3600        >OXB ADC
3610        >OXB STZ
3620        >OXB ADC
3630        >OXB ROR
```

```
3640        >OXB ADC
3650        >OXA PLA
3660        >OXB ADC
3670        >OXA RORA
3680        >OXA RTL
3690        >OXB JMP
3700        >OXB ADC
3710        >OXB ROR
3720        >OXB ADC
3730  *---7X---------------------------
3740        >OXB BVS
3750        >OXB ADC
3760        >OXB ADC
3770        >OXB ADC
3780        >OXB STZ
3790        >OXB ADC
3800        >OXB ROR
3810        >OXB ADC
3820        >OXA SEI
3830        >OXB ADC
3840        >OXA PLY
3850        >OXA TDC
3860        >OXB JMP
3870        >OXB ADC
3880        >OXB ROR
3890        >OXB ADC
3900  *---8X---------------------------
3910        >OXB BRA
3920        >OXB STA
3930        >OXB BRL
3940        >OXB STA
3950        >OXB STY
3960        >OXB STA
3970        >OXB STX
3980        >OXB STA
3990        >OXA DEY
4000        >OXB BIT
4010        >OXA TXA
4020        >OXA PHB
4030        >OXB STY
4040        >OXB STA
4050        >OXB STX
4060        >OXB STA
4070  *---9X---------------------------
4080        >OXB BCC
4090        >OXB STA
4100        >OXB STA
4110        >OXB STA
4120        >OXB STY
4130        >OXB STA
4140        >OXB STX
4150        >OXB STA
4160        >OXA TYA
4170        >OXB STA
```

```
4180        >OXA TXS
4190        >OXA TXY
4200        >OXB STZ
4210        >OXB STA
4220        >OXB STZ
4230        >OXB STA
4240 *---AX------------------------
4250        >OXB LDY
4260        >OXB LDA
4270        >OXB LDX
4280        >OXB LDA
4290        >OXB LDY
4300        >OXB LDA
4310        >OXB LDX
4320        >OXB LDA
4330        >OXA TAY
4340        >OXB LDA
4350        >OXA TAX
4360        >OXA PLB
4370        >OXB LDY
4380        >OXB LDA
4390        >OXB LDX
4400        >OXB LDA
4410 *---BX------------------------
4420        >OXB BCS
4430        >OXB LDA
4440        >OXB LDA
4450        >OXB LDA
4460        >OXB LDY
4470        >OXB LDA
4480        >OXB LDX
4490        >OXB LDA
4500        >OXA CLV
4510        >OXB LDA
4520        >OXA TSX
4530        >OXA TYX
4540        >OXB LDY
4550        >OXB LDA
4560        >OXB LDX
4570        >OXB LDA
4580 *---CX------------------------
4590        >OXB CPY
4600        >OXB CMP
4610        >OXB REP
4620        >OXB CMP
4630        >OXB CPY
4640        >OXB CMP
4650        >OXB DEC
4660        >OXB CMP
4670        >OXA INY
4680        >OXB CMP
4690        >OXA DEX
4700        >OXA WAI
4710        >OXB CPY
```

```
4720        >OXB CMP
4730        >OXB DEC
4740        >OXB CMP
4750 *---DX---------------------------
4760        >OXB BNE
4770        >OXB CMP
4780        >OXB CMP
4790        >OXB CMP
4800        >OXB PEI
4810        >OXB CMP
4820        >OXB DEC
4830        >OXB CMP
4840        >OXA CLD
4850        >OXB CMP
4860        >OXA PHX
4870        >OXA STP
4880        >OXB JML
4890        >OXB CMP
4900        >OXB DEC
4910        >OXB CMP
4920 *---EX---------------------------
4930        >OXB CPX
4940        >OXB SBC
4950        >OXB SEP
4960        >OXB SBC
4970        >OXB CPX
4980        >OXB SBC
4990        >OXB INC
5000        >OXB SBC
5010        >OXA INX
5020        >OXB SBC
5030        >OXA NOP
5040        >OXA XBA
5050        >OXB CPX
5060        >OXB SBC
5070        >OXB INC
5080        >OXB SBC
5090 *---FX---------------------------
5100        >OXB BEQ
5110        >OXB SBC
5120        >OXB SBC
5130        >OXB SBC
5140        >OXB PEA
5150        >OXB SBC
5160        >OXB INC
5170        >OXB SBC
5180        >OXA SED
5190        >OXB SBC
5200        >OXA PLX
5210        >OXA XCE
5220        >OXB JSR
5230        >OXB SBC
5240        >OXB INC
5250        >OXB SBC
```

```
*-------------------------------
OPFORMAT
F.0    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.1    .HS 26.16.12.1E.02.08.08.22.00.10.00.00.04.0A.0A.0C
F.2    .HS 04.14.06.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.3    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.0A.0A.0A.0C
F.4    .HS 00.14.00.1C.24.02.02.20.00.00.00.00.04.04.04.06
F.5    .HS 26.16.12.1E.24.08.08.22.00.10.00.00.06.0A.0A.0C
F.6    .HS 00.14.28.1C.02.02.02.20.00.00.00.00.18.04.04.06
F.7    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
F.8    .HS 26.14.28.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.9    .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.04.0A.0A.0C
F.A    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.B    .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.0A.0A.10.0C
F.C    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.D    .HS 26.16.12.1E.02.08.08.22.00.10.00.00.18.0A.0A.0C
F.E    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
F.F    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
*-------------------------------
FMTBL
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %1.0.0.1.0.0.0.0.0.0.0.0.0.0.01 -- IMMEDIATE    00
 .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.01 -- DIRECT       02
 .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.10 -- ABS          04
 .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.11 -- LONG         06
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.01 -- DIRECT,X     08
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.10 -- ABS,X        0A
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.11 -- LONG,X       0C
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.0.01 -- DIRECT,Y     0E
 .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.0.10 -- ABS,Y        10
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.0.01 -- IND          12
 .DA %0.0.1.1.1.1.0.1.0.0.0.0.0.0.01 -- INDX         14
 .DA %0.0.1.1.0.0.0.1.1.1.0.0.0.0.01 -- INDY         16
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.0.10 -- INDABS       18
 .DA %0.0.1.1.1.1.0.1.0.0.0.0.0.0.10 -- INDABSX      1A
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.0.1.1.0.1.0.0.0.0.0.0.0.01 -- STK          1C
 .DA %0.0.1.1.1.0.1.1.1.1.0.0.0.0.01 -- STKY         1E
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.1.1.1.0.0.0.1.0.0.0.0.0.0.01 -- INDLONG      20
 .DA %0.1.1.1.0.0.0.1.1.1.0.0.0.0.01 -- INDLONGY     22
 .DA %0.0.0.1.0.0.0.0.1.0.1.0.0.0.10 -- MVN & MVP    24
 .DA %0.0.0.0.0.0.0.0.0.0.1.0.0.0.01 -- RELATIVE     26
 .DA %0.0.0.0.0.0.0.0.0.0.1.0.0.0.10 -- LONG RELA.   28
*-------------------------------
FMTSTR .AS -/$Y,)SX,$(>#/
*-------------------------------
INSDS1 JSR CROUT
       LDA PCH
       JSR PRBYTE
```

```
5800        LDA PCL
5810        JSR PRBYTE
5820        LDA #"-"
5830        JSR COUT
5840        LDA #" "
5850        JSR COUT
5860        LDY #0
5870        LDA (PCL),Y  GET OPCODE
5880 INSDS2 TAY          SAVE IN Y-REG
5890        LDA OPINDEX,Y
5900        ASL
5910        TAX
5920        BCC .1       ...NOT SINGLE BYTE OPCODE
5930        LDA OPNAMES.A,X
5940        STA RMNEM
5950        LDA OPNAMES.A+1,X
5960        STA LMNEM
5970        LDA #0
5980        STA LENGTH
5990        RTS
6000 *---------------------------------
6010 .1     LDA OPNAMES.B,X
6020        STA RMNEM
6030        LDA OPNAMES.B+1,X
6040        STA LMNEM
6050        LDX OPFORMAT,Y
6060        LDA FMTBL+1,X
6070        STA FORMATH
6080        LDA FMTBL,X
6090        STA FORMATL
6100        AND #3
6110        STA LENGTH
6120        TXA          CHECK IF IMMEDIATE
6130        BNE .2       ...NO
6140        BIT IMM.SIZE CHECK IF 16-BIT MODE
6150        BPL .2       ...NO
6160        INC LENGTH   ...YES
6170 .2     RTS
6180 *---------------------------------
6190 INSTDSP
6200        JSR INSDS1
6210        LDY #0
6220 .1     LDA (PCL),Y
6230        JSR PRBYTE
6240        LDX #1       PRINT 1 BLANK
6250 .2     JSR PRBL2
6260        CPY LENGTH
6270        INY
6280        BCC .1
6290        LDX #3
6300        CPY #4
6310        BCC .2
6320 *---PRINT MNEMONIC---------------
6330        LDY #3
```

```
6340  .3     LDA #6
6350  .4     ASL RMNEM
6360         ROL LMNEM
6370         ROL
6380         BPL .4
6390         JSR COUT
6400         DEY
6410         BNE .3
6420         LDY LENGTH
6430         BEQ .8       ...SINGLE BYTE OPCODE
6440         LDA FORMATL
6450         AND #$20     SEE IF SPECIAL
6460         BNE .9       ...YES, MOVES OR RELATIVES
6470  *---PRINT NORMAL OPERANDS--------
6480         LDA #" "
6490         JSR COUT
6500         LDX #10      11 FORMAT BITS
6510  .5     ASL FORMATL
6520         ROL FORMATH
6530         BCC .7
6540         LDA FMTSTR,X
6550         JSR COUT
6560         CMP #"#"
6570         BNE .55
6580         BIT IMM.SIZE
6590         BPL .7
6600         JSR COUT
6610  .55    CMP #"$"
6620         BNE .7
6630  .6     LDA (PCL),Y
6640         JSR PRBYTE
6650         DEY
6660         BNE .6
6670  .7     DEX
6680         BPL .5
6690  .8     RTS
6700  *---SPECIAL CASES----------------
6710  .9     LDA #" "
6720         JSR COUT
6730         LDA #"$"
6740         JSR COUT
6750         LDA FORMATL
6760         BMI .11      MVN & MVP
6770         DEY          DISTINGUISH RELATIVES
6780         BNE .10      16-BIT RELATIVE
6790  *---8-BIT RELATIVE---------------
6800         INY          8-BIT RELATIVE
6810         LDA (PCL),Y  GET 8-BIT OFFSET
6820         SEC
6830         JMP RELADR
6840  *---16-BIT RELATIVE--------------
6850  .10    LDA (PCL),Y  LOW BYTE OF OFFSET
6860         STA FORMATL
6870         INY
```

```
6880        LDA (PCL),Y  HIGH BYTE OF OFFSET
6890        STA FORMATH
6900        JSR PCADJ
6910        CLC
6920        ADC FORMATL
6930        TAX
6940        TYA
6950        ADC FORMATH
6960        JMP PRNTAX
6970 *---MVN & MVP--------------------
6980 .11    LDA (PCL),Y
6990        JSR PRBYTE
7000        LDA #","
7010        JSR COUT
7020        LDA #"$"
7030        JSR COUT
7040        DEY
7050        LDA (PCL),Y
7060        JMP PRBYTE
7070 *--------------------------------
7080 TT     LDY #0
7090        LDA #$C0
7100        STA PCL
7110        LDA #2       $2C0...$3C3
7120        STA PCH
7130 .1     TYA
7140        STA $2C0,Y
7150        INY
7160        BNE .1
7170        STY $3C0
7180        INY
7190        STY $3C1
7200        INY
7210        STY $3C2
7220 .2     JSR INSTDSP
7230        LDY #0
7240        LDA (PCL),Y
7250        CMP #$FF
7260        BEQ .3
7270 .4     LDA $C000
7280        BPL .4
7290        STA $C010
7300        INC PCL
7310        BNE .2
7320        INC PCH
7330        BNE .2       ...ALWAYS
7340 .3     RTS
7350 *--------------------------------
```

```
==============================================================
DOCUMENT :AAL-8503:DOS3.3:S.DOS.NUMIN.txt
==============================================================

 1000  *SAVE S.DOS NUMIN
 1010  *--------------------------------
 1020  NUML   .EQ $44
 1030  NUMH   .EQ $45
 1040  *--------------------------------
 1050  GNNB   .EQ $A1A4
 1060  *--------------------------------
 1070         .OR $A1B9
 1080         .TA $09B9
 1090  *--------------------------------
 1100  *      RETURN .CC. WITH NUMBER IN A,X
 1110  *          OR .CS. IF BAD SYNTAX
 1120  *--------------------------------
 1130  CONVERT.NUMBER.IN.WBUF
 1140         LDA #0       INIT NUMBER = 0
 1150         STA NUML
 1160         STA NUMH
 1170         JSR GNNB     GET NEXT NON-BLANK CHAR
 1180         PHP
 1190         CMP #"$"     HEX OR DECIMAL?
 1200         BEQ .6       ...HEX
 1210         PLP
 1220         JMP .2       ...DECIMAL (OR NONE)
 1230  *---NEXT CHAR OF DECIMAL #-------
 1240  .1     JSR GNNB     GET NEXT NON-BLANK CHAR
 1250  .2     BNE .3       ...NOT COMMA OR CR
 1260         LDX NUML     END OF NUMBER
 1270         LDA NUMH     VALUE IN A,X
 1280         CLC          SIGNAL VALID NUMBER
 1290         RTS          RETURN
 1300  *---CONVERT DECIMAL NUMBER-------
 1310  .3     SEC          CONVERT CHAR TO DIGIT
 1320         SBC #$B0
 1330         BMI .4       ...NOT DIGIT
 1340         CMP #$0A
 1350         BCS .4       ...NOT DIGIT
 1360         JSR .5       SHIFT VALUE 1 LEFT
 1370         ADC NUML     2*VALUE + DIGIT
 1380         TAX
 1390         LDA #$00
 1400         ADC NUMH
 1410         TAY
 1420         JSR .5       SHIFT VALUE 1 LEFT
 1430         JSR .5       SHIFT VALUE 1 LEFT
 1440         TXA          ...+ 8*VALUE
 1450         ADC NUML
 1460         STA NUML
 1470         TYA
 1480         ADC NUMH
```

```
1490        STA NUMH
1500        BCC .1        ...NO OVERFLOW
1510 .4     SEC           SIGNAL BAD CHAR OR OVERFLOW
1520        RTS
1530 *---SHIFT VALUE 1 BIT LEFT-------
1540 .5     ASL NUML
1550        ROL NUMH
1560        RTS
1570 *---CONVERT HEX NUMBER-----------
1580 .6     PLP           POP USELESS STATUS
1590 .7     JSR GNNB      GET NEXT NON-BLANK CHAR
1600        BEQ .2        ...END OF NUMBER
1610        SEC           CONVERT ASCII TO DIGIT
1620        SBC #$B0
1630        BMI .4        ...NOT A DIGIT
1640        CMP #$0A
1650        BCC .8        ...0-9
1660        SBC #$07      TRY LETTERS
1670        BMI .4        ...NOT A DIGIT
1680        CMP #$10
1690        BCS .4        ...NOT A DIGIT
1700 .8     LDX #4        SHIFT VALUE 4 BITS LEFT
1710 .9     JSR .5        SHIFT VALUE 1 LEFT
1720        DEX
1730        BNE .9
1740        ORA NUML      MERGE VALUE WITH NEW DIGIT
1750        STA NUML
1760        JMP .7        ...NEXT DIGIT
1770 *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.DOSLCPatch.txt
==============================================================================

```
1000  *SAVE S.DOS LC PATCHES
1010  *--------------------------------
1020  PNTR   .EQ $00,01
1030  PATCH  .EQ $02,03
1040  *--------------------------------
1050         .OR $300
1060         .TF B.DOS LC PATCHES
1070  *--------------------------------
1080  PATCHER
1090         LDA #PATCHES-1
1100         STA PNTR
1110         LDA /PATCHES-1
1120         STA PNTR+1
1130         LDY #0
1140
1150  .1     JSR GET.BYTE LENGTH OF NEXT PATCH
1160         BEQ .4       FINISHED
1170         TAX          SAVE LENGTH IN X
1180         JSR GET.BYTE ADDRESS OF PATCH
1190         STA PATCH
1200         JSR GET.BYTE
1210         STA PATCH+1
1220
1230  .2     JSR GET.BYTE
1240         STA (PATCH),Y
1250         INC PATCH
1260         BNE .3
1270         INC PATCH+1
1280  .3     DEX
1290         BNE .2
1300         BEQ .1    ...ALWAYS
1310
1320  .4     RTS
1330  *--------------------------------
1340  GET.BYTE
1350         INC PNTR
1360         BNE .1
1370         INC PNTR+1
1380  .1     LDA (PNTR),Y
1390         RTS
1400  *--------------------------------
1410  NUML   .EQ $44
1420  NUMH   .EQ $45
1430  *--------------------------------
1440  GNNB   .EQ $A1A4
1450  *--------------------------------
1460  PATCHES
1470         .DA #P1.LENGTH,$A1B9
1480         .PH $A1B9
```

```
1490  *--------------------------------
1500  *       RETURN .CC. WITH NUMBER IN A,X
1510  *           OR .CS. IF BAD SYNTAX
1520  *--------------------------------
1530  CONVERT.NUMBER.IN.WBUF
1540         LDY #0       INIT NUMBER = 0
1550         STY NUML     (AND LEAVE Y=0 TOO)
1560         STY NUMH
1570         JSR GNNB     GET NEXT NON-BLANK CHAR
1580         BEQ .2       ...NO NUMBER, RETURN 0
1590         CMP #"$"     HEX OR DECIMAL?
1600         BEQ .7       ...HEX
1610  *---CONVERT DECIMAL NUMBER-------
1620  .3     EOR #$B0     CONVERT CHAR TO DIGIT
1630         CMP #10
1640         BCS .4       ...NOT DIGIT
1650         ASL NUML     SHIFT VALUE 1 LEFT
1660         ROL NUMH
1670         ADC NUML     2*VALUE + DIGIT
1680         TAX
1690         TYA          A = Y = 0
1700         ADC NUMH
1710         PHA
1720         ASL NUML     SHIFT VALUE 1 LEFT
1730         ROL NUMH
1740         ASL NUML     SHIFT VALUE 1 LEFT
1750         ROL NUMH
1760         TXA          ...+ 8*VALUE
1770         ADC NUML
1780         STA NUML
1790         PLA
1800         ADC NUMH
1810         STA NUMH
1820         BCS .4       ...OVERFLOW
1830  .1     JSR GNNB     GET NEXT NON-BLANK CHAR
1840         BNE .3       ...NOT COMMA OR CR
1850  *---NUMBER IS FINISHED-----------
1860  .2     LDX NUML     END OF NUMBER
1870         LDA NUMH     VALUE IN A,X
1880         CLC          SIGNAL VALID NUMBER
1890         RTS          RETURN
1900  *---MERGE NEXT HEX DIGIT---------
1910  .8     ASL          POSITION DIGIT
1920         ASL
1930         ASL
1940         ASL
1950         LDX #4       SHIFT VALUE 4 BITS LEFT
1960  .9     ASL          SHIFT DIGIT INTO VALUE
1970         ROL NUML
1980         ROL NUMH
1990         DEX
2000         BNE .9
2010  *---CONVERT HEX NUMBER-----------
2020  .7     JSR GNNB     GET NEXT NON-BLANK CHAR
```

```
2030        BEQ .2       ...END OF NUMBER
2040        EOR #$B0     CONVERT ASCII TO DIGIT
2050        CMP #10      0...9?
2060        BCC .8       ...YES, 0-9
2070        ADC #$88     SHIFT RANGE FOR A-F TEST
2080        CMP #$FA     A...F?
2090        BCS .8       ...A-F
2100 *---SYNTAX ERROR----------------
2110 .4     SEC          SIGNAL BAD CHAR OR OVERFLOW
2120        RTS
2130 *-------------------------------
2140 GNC.LC.PATCH
2150        STX $AA5D
2160        CMP #$E0
2170        BCC .1
2180        AND #$DF
2190 .1     RTS
2200 *-------------------------------
2210        .BS $A229-*
2220 *-------------------------------
2230 P1.LENGTH .EQ *-$A1B9
2240        .EP
2250 *-------------------------------
2260        .DA #3,$A19E
2270        .PH $A19E
2280        JSR GNC.LC.PATCH
2290        .EP
2300 *-------------------------------
2310        .DA #0       END OF PATCHES
2320 *-------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8503:DOS3.3:S.DOSNuminRBSC.txt
==========================================================================

 1000  *SAVE S.DOS NUMIN (RBSC)
 1010  *--------------------------------
 1020  NUML   .EQ $44
 1030  NUMH   .EQ $45
 1040  *--------------------------------
 1050  GNNB   .EQ $A1A4
 1060  *--------------------------------
 1070         .OR $A1B9
 1080         .TA $09B9
 1090  *--------------------------------
 1100  *      RETURN .CC. WITH NUMBER IN A,X
 1110  *          OR .CS. IF BAD SYNTAX
 1120  *--------------------------------
 1130  CONVERT.NUMBER.IN.WBUF
 1140         LDY #0       INIT NUMBER = 0
 1150         STY NUML     (AND LEAVE Y=0 TOO)
 1160         STY NUMH
 1170         JSR GNNB     GET NEXT NON-BLANK CHAR
 1180         BEQ .2       ...NO NUMBER, RETURN 0
 1190         CMP #"$"     HEX OR DECIMAL?
 1200         BEQ .7       ...HEX
 1210  *---CONVERT DECIMAL NUMBER-------
 1220  .3     EOR #$B0     CONVERT CHAR TO DIGIT
 1230         CMP #10
 1240         BCS .4       ...NOT DIGIT
 1250         ASL NUML     SHIFT VALUE 1 LEFT
 1260         ROL NUMH
 1270         ADC NUML     2*VALUE + DIGIT
 1280         TAX
 1290         TYA          A = Y = 0
 1300         ADC NUMH
 1310         PHA
 1320         ASL NUML     SHIFT VALUE 1 LEFT
 1330         ROL NUMH
 1340         ASL NUML     SHIFT VALUE 1 LEFT
 1350         ROL NUMH
 1360         TXA          ...+ 8*VALUE
 1370         ADC NUML
 1380         STA NUML
 1390         PLA
 1400         ADC NUMH
 1410         STA NUMH
 1420         BCS .4       ...OVERFLOW
 1430  .1     JSR GNNB     GET NEXT NON-BLANK CHAR
 1440         BNE .3       ...NOT COMMA OR CR
 1450  *---NUMBER IS FINISHED-----------
 1460  .2     LDX NUML     END OF NUMBER
 1470         LDA NUMH     VALUE IN A,X
 1480         CLC          SIGNAL VALID NUMBER
```

```
1490         RTS          RETURN
1500  *---MERGE NEXT HEX DIGIT---------
1510  .8     ASL          POSITION DIGIT
1520         ASL
1530         ASL
1540         ASL
1550         LDX #4       SHIFT VALUE 4 BITS LEFT
1560  .9     ASL          SHIFT DIGIT INTO VALUE
1570         ROL NUML
1580         ROL NUMH
1590         DEX
1600         BNE .9
1610  *---CONVERT HEX NUMBER-----------
1620  .7     JSR GNNB     GET NEXT NON-BLANK CHAR
1630         BEQ .2       ...END OF NUMBER
1640         EOR #$B0     CONVERT ASCII TO DIGIT
1650         CMP #10      0...9?
1660         BCC .8       ...YES, 0-9
1670         ADC #$88     SHIFT RANGE FOR A-F TEST
1680         CMP #$FA     A...F?
1690         BCS .8       ...A-F
1700  *---SYNTAX ERROR-----------------
1710  .4     SEC          SIGNAL BAD CHAR OR OVERFLOW
1720         RTS
1730  *--------------------------------
1740         .OR $800
1750  TEST   JSR $FD67
1760         TXA
1770         BEQ .1
1780         LDX #0
1790         STX $AA5D
1800         JSR CONVERT.NUMBER.IN.WBUF
1810         JSR $F941
1820         JMP TEST
1830  .1     RTS
```

```
==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.INIT.BUFFERS.txt
==============================================================================

 1000  *SAVE S.INIT BUFFERS
 1010  *--------------------------------
 1020  PNTR       .EQ $40,41
 1030  HIMEM      .EQ $4C,4D
 1040  FP.STRINGS .EQ $6F,70
 1050  FP.HIMEM   .EQ $73,74
 1060  PP         .EQ $CA,CB
 1070  *--------------------------------
 1080  BUF.START         .EQ $9D00
 1090  NO.FILES          .EQ $AA57
 1100  TEMP              .EQ $AA63
 1110  ACTIVE.BASIC.FLAG .EQ $AAB6
 1120  *--------------------------------
 1130         .OR $A7D4
 1140         .TA $08D4
 1150  *--------------------------------
 1160  INIT.FILE.BUFFERS
 1170         SEC
 1180         LDA BUF.START
 1190         STA PNTR
 1200         LDA BUF.START+1
 1210         STA PNTR+1
 1220         LDA NO.FILES
 1230         STA TEMP
 1240  *--------------------------------
 1250  .1     LDY #0
 1260         TYA
 1270         STA (PNTR),Y
 1280         LDY #$1E
 1290         SEC
 1300         LDA PNTR
 1310         SBC #$2D
 1320         STA (PNTR),Y
 1330         PHA
 1340         LDA PNTR+1
 1350         SBC #0
 1360         INY
 1370         STA (PNTR),Y
 1380         TAX
 1390         DEX
 1400         PLA
 1410         PHA
 1420         INY
 1430         STA (PNTR),Y
 1440         TXA
 1450         INY
 1460         STA (PNTR),Y
 1470         TAX
 1480         DEX
```

```
1490        PLA
1500        PHA
1510        INY
1520        STA (PNTR),Y
1530        INY
1540        TXA
1550        STA (PNTR),Y
1560        DEC TEMP
1570        BEQ .2
1580        TAX
1590        PLA
1600        SEC
1610        SBC #$26
1620        INY
1630        STA (PNTR),Y
1640        PHA
1650        TXA
1660        SBC #0
1670        INY
1680        STA (PNTR),Y
1690        STA PNTR+1
1700        PLA
1710        STA PNTR
1720        JMP .1
1730 *--------------------------------
1740 .2     PHA
1750        LDA #0
1760        INY
1770        STA (PNTR),Y
1780        INY
1790        STA (PNTR),Y
1800        LDA ACTIVE.BASIC.FLAG
1810        BEQ .3
1820        PLA
1830        STA FP.HIMEM+1
1840        STA FP.STRINGS+1
1850        PLA
1860        STA FP.HIMEM
1870        STA FP.STRINGS
1880        RTS
1890 *--------------------------------
1900 .3     PLA
1910        STA HIMEM+1
1920        STA PP+1
1930        PLA
1940        STA HIMEM
1950        STA PP
1960        RTS
1970 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.InitBuf802.XY.txt
==============================================================================

 1000  *SAVE S.INIT BUFFERS (802) X/Y
 1010         .OP 65816
 1020  *---------------------------------
 1030  *   REPLACEMENT FOR DOS 3.3 CODE
 1040  *      (SAVES 52 BYTES, NO CHANGE IN FUNCTION)
 1050  *---------------------------------
 1060  HIMEM             .EQ $4C,4D
 1070  FP.STRINGS        .EQ $6F,70
 1080  FP.HIMEM          .EQ $73,74
 1090  PP                .EQ $CA,CB
 1100  *---------------------------------
 1110  BUF.START         .EQ $9D00
 1120  NO.FILES          .EQ $AA57
 1130  ACTIVE.BASIC.FLAG .EQ $AAB6
 1140  *---------------------------------
 1150         .OR $A7D4
 1160         .TA $08D4
 1170  *---------------------------------
 1180  INIT.FILE.BUFFERS
 1190         PHP                 SAVE CURRENT STATUS AND
 1200         CLC                 TURN ON 802 MODE
 1210         XCE
 1220         PHP
 1230  *---------------------------------
 1240         LDY NO.FILES        DO (NO.FILES) TIMES
 1250         REP #$30            16-BIT OPERATIONS
 1260         LDX BUF.START       POINT TO FIRST BUFFER
 1270  *---------------------------------
 1280  .1     LDA ##0             STORE ZERO OVER 1ST & 2ND CHARS
 1290         STA 0,X             OF FILENAME TO FREE BUFFER
 1300  *---FILL IN 3 PNTRS--------------
 1310         SEC                 COMPUTE LOW BYTE OF POINTERS
 1320         TXA                 FROM FILENAME ADDR
 1330         SBC ##$2D
 1340         STA $1E,X           ...FMW ADDR
 1350         SBC ##$100
 1360         STA $20,X           ...TSL ADDR
 1370         SBC ##$100
 1380         STA $22,X           ...DATA ADDR
 1390  *---IS THAT THE LAST BUFFER?-----
 1400         DEY
 1410         BEQ .2        ...NO MORE BUFFERS
 1420  *---BUILD LINK TO NEXT BUFFER----
 1430         SBC ##$26     ADDR OF FILENAME IN NEXT BUFFER
 1440         STA $24,X
 1450         TAX           BASE ADDRESS FOR NEXT BUFFER
 1460         BRA .1             ...ALWAYS
 1470  *---SET FORWARD PNTR = 0000------
 1480  .2     TAY           SAVE HIMEM VALUE
```

```
1490        LDA ##0
1500        STA $24,X
1510 *---SET HIMEM AND EMPTY BLOCK----
1520        LDA ACTIVE.BASIC.FLAG
1530        AND ##$FF
1540        BEQ .3            INTEGER BASIC
1550        STY FP.HIMEM      APPLESOFT
1560        STY FP.STRINGS
1570        BRA .4
1580 .3     STY HIMEM         INTEGER BASIC
1590        STY PP
1600 .4     PLP
1610        XCE
1620        PLP
1630        RTS
1640 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.InitBufs.802.txt
==============================================================================

 1000  *SAVE S.INIT BUFFERS (802)
 1010         .OP 65816
 1020  *--------------------------------
 1030  *   REPLACEMENT FOR DOS 3.3 CODE
 1040  *      (SAVES 46 BYTES, NO CHANGE IN FUNCTION)
 1050  *--------------------------------
 1060  HIMEM             .EQ $4C,4D
 1070  FP.STRINGS        .EQ $6F,70
 1080  FP.HIMEM          .EQ $73,74
 1090  PP                .EQ $CA,CB
 1100  *--------------------------------
 1110  BUF.START         .EQ $9D00
 1120  NO.FILES          .EQ $AA57
 1130  ACTIVE.BASIC.FLAG .EQ $AAB6
 1140  *--------------------------------
 1150         .OR $A7D4
 1160         .TA $08D4
 1170  *--------------------------------
 1180  INIT.FILE.BUFFERS
 1190         PHP               SAVE CURRENT STATUS AND
 1200         CLC               TURN ON 802 MODE
 1210         XCE
 1220         PHP
 1230  *--------------------------------
 1240         LDX NO.FILES      DO (NO.FILES) TIMES
 1250         REP #$30          16-BIT OPERATIONS
 1260         LDY BUF.START     POINT TO FIRST BUFFER
 1270  *--------------------------------
 1280  .1     LDA ##0           STORE ZERO OVER 1ST & 2ND CHARS
 1290         STA 0,Y           OF FILENAME TO FREE BUFFER
 1300  *---FILL IN 3 PNTRS--------------
 1310         SEC               COMPUTE LOW BYTE OF POINTERS
 1320         TYA               FROM FILENAME ADDR
 1330         SBC ##$2D
 1340         STA $1E,Y         ...FMW ADDR
 1350         SBC ##$100
 1360         STA $20,Y         ...TSL ADDR
 1370         SBC ##$100
 1380         STA $22,Y         ...DATA ADDR
 1390  *---IS THAT THE LAST BUFFER?-----
 1400         DEX
 1410         BEQ .2       ...NO MORE BUFFERS
 1420  *---BUILD LINK TO NEXT BUFFER----
 1430         SBC ##$26    ADDR OF FILENAME IN NEXT BUFFER
 1440         STA $24,Y
 1450         TAY          BASE ADDRESS FOR NEXT BUFFER
 1460         BRA .1            ...ALWAYS
 1470  *---SET FORWARD PNTR = 0000------
 1480  .2     TAX          SAVE HIMEM VALUE
```

```
1490        LDA ##0
1500        STA $24,Y
1510 *---SET HIMEM AND EMPTY BLOCK----
1520        LDA ACTIVE.BASIC.FLAG
1530        AND ##$FF
1540        BEQ .3            INTEGER BASIC
1550        STX FP.HIMEM      APPLESOFT
1560        STX FP.STRINGS
1570        BRA .4
1580 .3     STX HIMEM         INTEGER BASIC
1590        STX PP
1600 .4     PLP
1610        XCE
1620        PLP
1630        RTS
1640 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8503:DOS3.3:S.InitBufs.SC.txt
==============================================================================
```

```
 1000  *SAVE S.INIT BUFFERS (S-C)
 1010  *--------------------------------
 1020  *   REPLACEMENT FOR DOS 3.3 CODE
 1030  *      (SAVES 11 BYTES, NO CHANGE IN FUNCTION)
 1040  *--------------------------------
 1050  PNTR              .EQ $40,41
 1060  HIMEM             .EQ $4C,4D
 1070  FP.STRINGS        .EQ $6F,70
 1080  FP.HIMEM          .EQ $73,74
 1090  PP                .EQ $CA,CB
 1100  *--------------------------------
 1110  BUF.START         .EQ $9D00
 1120  NO.FILES          .EQ $AA57
 1130  TEMP              .EQ $AA63
 1140  ACTIVE.BASIC.FLAG .EQ $AAB6
 1150  *--------------------------------
 1160         .OR $A7D4
 1170         .TA $08D4
 1180  *--------------------------------
 1190  INIT.FILE.BUFFERS
 1200         LDA NO.FILES      DO (NO.FILES) TIMES
 1210         STA TEMP          USE TEMP FOR COUNTER
 1220         LDA BUF.START     POINT TO FIRST BUFFER
 1230         LDX BUF.START+1
 1240  *--------------------------------
 1250  .1     STA PNTR
 1260         STX PNTR+1
 1270         LDY #0            Store zero over 1st char of
 1280         TYA               filename to mark it as a
 1290         STA (PNTR),Y      free buffer.
 1300  *---FILL IN 3 PNTRS-------------
 1310         SEC               COMPUTE LOW BYTE OF POINTERS
 1320         LDA PNTR
 1330         SBC #$2D
 1340         LDY #$1E          ...FMW ADDR
 1350         STA (PNTR),Y
 1360         LDY #$20          ...TSL ADDR
 1370         STA (PNTR),Y
 1380         LDY #$22          ...DATA ADDR
 1390         STA (PNTR),Y
 1400         PHA
 1410         LDA PNTR+1        COMPUTE HIGH BYTE OF FMW ADDR
 1420         SBC #0
 1430         LDY #$1F          ...FMW ADDR
 1440         STA (PNTR),Y
 1450         SBC #1
 1460         LDY #$21          ...TSL ADDR
 1470         STA (PNTR),Y
 1480         SBC #1
```

```
       LDY #$23          ...DATA ADDR
       STA (PNTR),Y
*---IS THAT THE LAST BUFFER?-----
       INY               POINT AT FWD LINK LO-BYTE
       TAX          SAVE HI BYTE OF DATA ADDR
       DEC TEMP
       BEQ .2       ...NO MORE BUFFERS
*---BUILD LINK TO NEXT BUFFER----
       PLA          GET LO BYTE
       SBC #$26     ADDR OF FILENAME IN NEXT BUFFER
       STA (PNTR),Y      ...LO BYTE
       PHA               SAVE ON STACK
       TXA               GET HI BYTE
       SBC #0
       INY               ...HI BYTE
       STA (PNTR),Y
       TAX               SAVE IN X
       PLA               GET LO BYTE AGAIN
       BCS .1            ...ALWAYS
*---SET FORWARD PNTR = 0000------
.2     LDA #0
       STA (PNTR),Y
       INY
       STA (PNTR),Y
*---SET HIMEM AND EMPTY BLOCK----
       LDA ACTIVE.BASIC.FLAG
       BEQ .3            INTEGER BASIC
       STX FP.HIMEM+1    APPLESOFT
       STX FP.STRINGS+1
       PLA
       STA FP.HIMEM
       STA FP.STRINGS
       RTS
.3     STX HIMEM+1
       STX PP+1
       PLA
       STA HIMEM
       STA PP
       RTS
*--------------------------------
```

==================================================================
DOCUMENT :AAL-8504:Articles:AD.8086.XASM.txt
==================================================================

8086/8088 Cross Assembler
Use your Apple to learn 8086 programming! You can program for the IBM PC, the clones, and ALF's co-processor board without ever leaving the friendly environment of Apple DOS 3.3.

This easy-to-use cross assembler, based on the S-C Assembler II (Version 4.0), covers all the 8086 and 8088 instructions and all the addressing modes. Instruction mnemonics are based on the Microsoft 8086 assembler. Does not include newer S-C Assembler features like macros or the EDIT command.

Documentation covers the differences from standard S-C Assembler operation and syntax. Sample source programs help you become familiar with the assembler syntax.

With permission from S-C Software, XSM 8086/8088 is available to owners of any S-C Assembler for $80.00 post-paid. (No credit cards or purchase orders.)

Don Rindsberg
The Bit Stop
5958 S. Shenandoah Rd.
Mobile, AL 36608

(205) 342-1653
==================================================================
DOCUMENT :AAL-8504:Articles:Cross.8086.8088.txt
==================================================================

An 8086/8088 Cross Assembler......................Don Rindsberg

As one of S-C's avid fans, I have developed an 8086/8088 Cross Assembler for your Apple which will enable you to generate code to run on the IBM PC's and their clones as well as many other 16-bit machines. All the 8086/8088 instructions are covered as well as the multiplicity of addressing modes. The mnemonics are based on Microsoft's assembler. This assembler is based on S-C Assembler II Version 4.0 (the one before Macro Assembler), so it doesn't include the newer features like macros or the EDIT command. Documentation covering the differences from the 6502 version is included.

With Bob's permission, XSM 8086/8088 is available to owners of the S-C 6502 assembler (Version 4.0 or later) for $80.00 post- paid. Included on the disk are sample source programs so you can become familiar with the syntax. Send personal check or money order (no credit cards or purchase orders) to:

The Bit Stop
5958 S. Shenandoah Rd.

Mobile, AL  36608
Attn:  Don Rindsberg
(205) 342-1653

===========================================================================
DOCUMENT :AAL-8504:Articles:Fast.Windows.txt
===========================================================================

Fast Text Windows for Applesoft..................Michael Ching
2118 Kula Street, Honolulu, HI 96817

The program WINDER by Mike Seeds in the January 1985 NIBBLE was found to be very interesting. This was especially so because we, coincidentally, had been working on a similar routine for use in an upcoming strategy sports game.

The main difference between our programs was that the routines used in WINDER are written completely in Applesoft, and thus suffer from the relatively slow speed of the Applesoft interpreter. This is especially evident in the opening of the windows. Our routine, on the other hand, is written in assembly language and executes more quickly.

There are a couple of other major differences. Seeds' routine saves the text, to be overwritten by the window, in a string array WS$. Our routine saves the text in the secondary text page (memory locations $800 through $BFF). One advantage of doing this is that more than one window can be opened at the same time, (although the windows may not overlap). A disadvantage is that the secondary text page occupies the same space that an Applesoft program normally would start at. This makes it necessary to relocate the Applesoft program above the secondary text page.

Another difference is that WINDER specifies the window dimensions with the width and height of the window, along with the top and left coordinates. We chose to specify directly the top, bottom, left, and right boundaries.

The assembly language routine is called by the familiar & followed by the appropriate parameters. The format is & WT,WB,WL,WR,TP where WT is the top coordinate of the window, WB is the bottom coordinate, WL is the left coordinate, WR is the right coordinate, and TP is the text page number. If TP is set to 1, the text to be replaced by the window is saved to the secondary text page and the window is formed. If TP is set to 2, the text is restored to the primary text page from the secondary text page. At present, there is no error checking of the parameter values, and care must be taken to ensure that WB is set greater than WT, and WR greater than WL.

The program is assembled to load into the tail end of the input buffer and the free space in page 3 ($2F5-3C9). The portion inside page 2 is only used to set up the ampersand hook, so it is not a problem if this code gets wiped out by long input lines after loading. This setup is done in lines 1250-1290.

Lines 1320-1470 perform the task of getting the parameter values from Applesoft and placing them into temporary storage. The routines GETBYT and COMBYTE are used, and will evaluate expressions used in the calling Applesoft program. The width of the window is also calculated here. The text page value is decremented by one for ease of future manipulation. Line 1340 initializes the beginning of a loop which will copy the characters in the designated text page to the opposite text page.

Lines 1500-1510 call the monitor routine BASCALC. BASCALC calculates the starting (leftmost) memory address of the screenline, and stores it in the pointers BASL and BASH.

Lines 1520-1640 set up two pointers, one in the real screen and one in the alternate screen area. The pointers point to the beginning of the current line starting at the left edge of the caller's window. A1 points at the source, and A2 at the destination, for a move loop which will copy the characters within the window on the current line.

The destination address is the source address offset by $400 (up or down depending on the source text page). The calculation is done by exclusive ORing the source address with #$0C (or 00001100 in binary). For example, if BASH was $07, exclusive ORing will yield $0B. If it was $0B, exclusive ORing will yield $07.

Lines 1660-1700 comprise the move loop.

Lines 1720-1850 check to see if the frame of the window needs to be drawn. If the text page is being restored (window being closed), then the frame routine is skipped. If the window is being cleared, the frame is drawn.

First I store an inverse blank at each end of the line, which is sufficient for all except the top and bottom lines. Then I check: if it is the top or bottom line, I fill in the rest of the line with inverse blanks.

Lines 1870-1900 check whether the entire window has been processed. If not, the program loops back to process the next line.

Lines 1920-2050 check to see whether the window boundaries need to be set. If the window is being opened (TPAGE = 0), then they are set, and HOME clears out the window. Note that the window parameters are set so that the frame is outside it.

<<<assembly listing here.....

The next listing shows the revised WINDER routine using the assembly language routines. Line 40 checks to see if the program has been relocated above the secondary text page. If not, the start of program pointers are changed and the program is re-RUN. This causes DOS to position the program above the secondary text page. Line 50 BRUNS the assembly language routine.

The program is really quite different from that of Mike Seeds, as you can see if you compare them. Clearing and restoring windows is now very efficient, due to the &-routine. I moved the delay and closing logic into a common subroutine.  I also added a randomly sized and positioned window in lines 400-410.

<<<<Applesoft listing>>>>

==============================================================================
DOCUMENT :AAL-8504:Articles:Front.Page.txt
==============================================================================

$1.80

Volume 5 -- Issue 7 April, 1985

In This Issue...



A New Book Appears

Jim Sather's new book, Understanding the Apple //e, arrived today. We'll have a complete review next month, but at first glance it looks even better than his first book. Check our ad on page 3 for pricing.

And an Old Book Reappears

Roger Wagner Publishing has obtained the rights to Roger's "Assembly Lines -- the Book" from Softalk. A new edition is now available, still at $19.95. We sold hundreds of copies of this book, which in excellent tutorial fashion leads a beginner into the fascinating world of assembly language. "Assembly Lines -- the Disk" is also available, with all the sample source code formatted for the Merlin assembler. If you wish to order the book from us, our price is only $18 plus shipping.

Postage Increases

The recent Post Office rate increases had little effect on the Bulk and First Class rates, only $.015-.03 per piece, or $.18-.36 per year per subscription. We'll accept that much of a cost increase. Foreign Air Mail is another matter, though. Those rates went up by $.16-.19 per piece, or $1.92-2.28 per year per subscription. Therefore, the foreign subscription rate is now $32 per year.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

=============================================================================
DOCUMENT :AAL-8504:Articles:Hard.Cat.txt
=============================================================================

Volume Catalog for Corvus and Sider........Bob Sander-Cederlof

When I have a stack of floppies, I can quickly shuffle through them reading labels to find the two or three most likely to have the elusive file I want.  On a hard disk it is hard to read the labels....

The last time I had a Corvus sitting in this room, there was a program on the utility disk which would list the first file name from each volume.  If you were careful about making the first file name descriptive, it could act like a label.  Of course, nearly every floppy around here has a first file named HELLO.  Not too helpful.

Several years ago Bill Morgan wrote a program we published in AAL called the Catalog Arranger.  It allows you to re-arrange the filenames in any catalog to any order you wish, and to rename the files using any combination of upper/lower case, inverse, flashing, and control-characters.  I use Catalog Arranger to make a "title" file at the beginning of each hard disk volume.  (If you never heard of Catalog Arranger, you can type it in from AALs of October 1982 and January 1983.  It is also available on a Quarterly disk for only $15.)

Now that I don't have the Corvus, or its handy program for listing the names of the first file in each volume, I decided to write my own. The program that follows prints out the volume number, two spaces, and then the name of the first file.  If the volume is empty, it prints "<<<EMPTY VOLUME>>>".  You can abort the listing by pressing RETURN or ESCAPE, or pause it by pressing any other key.

Lines 1090-1100 set the origin at $803 and cause the object program to be written on a BRUNnable file called CAT.  We write it at $803 rather than $800 so that Applesoft will work correctly after CAT is finished. Applesoft gets upset if $800 has any non-zero value in it.

I used two monitor routines.  $FD8E prints a carriage return, and $FDED prints any character from the A-register.

I also used routines inside DOS.  $AFF7 reads the VTOC of the current volume, using the inverse volume number from the variable R.VOLUME. If there is any error in trying to read the VTOC, DOS would normally go through its procedure of printing the message and returning to Applesoft.  We cannot allow that, so I install a temporary patch to make the error condition cause a return to my code with carry set.  If there is no error, carry will be clear.  The only likely error is that I am asking for the VTOC of a non-existing volume, which means I have already processed them all.  The patching, call, and de-patching take place in lines 1160-1220.  Line 1230 branches to my exit routine if there was an error reported.

I also call on $B011 to read the first sector of the catalog. If you call $B011 with carry clear it reads the first sector of the catalog; with carry set, it reads the next sector of the catalog. The sector is read into a standard buffer at $B4BB-B5BA. See "Beneath Apple DOS" for a complete description of the catalog sectors.

Lines 1270-1440 convert the volume number to decimal and print it out. Lines 1450-1480 check for an empty directory. If it is empty, lines 1740-1800 print the empty volume message. Otherwise, lines 1490-1550 print the file name. Right here my program could use some improvement. It is possible for an empty volume to not look empty, because deleted files are not physically removed from the catalog. The byte we check for an empty volume could have $FF in it, signifying a deleted file. In this case my program should continue searching through the catalog for either the end or a non-deleted file. I didn't think it was absolutely necessary, since I was using Catalog Arranger to remove all deleted files from the catalog and position the title line at the very top.

Line 1730 returns back to DOS by JMP $3D0. This reminds me of glitch we all run into from time to time. If you intend to BRUN a program from the command level of the assembler or of Applesoft, it needs to end with JMP $3D0. Ending with an RTS will not do, because BRUN does not leave any return address on the stack. On the other hand, if you intend to start the program by using a CALL or MGO or $...G command, it is all right to end with an RTS. In fact, with a CALL from inside a running Applesoft program you MUST use an RTS. Just something to watch out for.

==============================================================================
DOCUMENT :AAL-8504:Articles:Inside.IIc.Book.txt
==============================================================================

New Book:  Inside the Apple //c

What Gary Little did for the //e he has repeated for the //c.  Of course a lot of the material is the same for both computers and both books, but there is much new material.  If you have a //c and not a //e, then this book will be much more helpful.

For one thing, when explaining assembly language he includes the new opcodes and address modes of the 65C02.  For another, the chapter on Disk Operating Systems is now 100% ProDOS, and includes more detail on ProDOS than the //e book.  Naturally, since the //c has no cassette port or I/O slots, that material has been left out.  On the other hand there is a lot of new data about the Apple mouse port and the built-in serial ports.

The book is published by Brady (Prentice-Hall), is 363 + xv pages, and sells for $19.95.  (We'll send you one for a little less, see page 3 of this newsletter.)

```
==============================================================================
DOCUMENT :AAL-8504:Articles:ListMajorLabels.txt
==============================================================================
```

USR Command to List Major Labels Only......Bob Sander-Cederlof

Sometimes when I am working with a large source file in the S-C Macro Assembler it would be nice to be able to list only those lines that define major labels. Seeing only them would give an overview of an entire file, and enable me to quickly find the section I want to work on.

A major label is one that starts with a letter. Local labels start with a period, macro private labels start with a colon. Lines might also start with an asterisk or semicolon, if they are comments, or with blank.

You can add commands to the Macro Assembler in several ways. One easy built in one is the USR command. A vector at $D007 (or $1007 with the low memory version) can point to the code to process a command of your own making. Lines 1080-1140 in the following listing set up the vector for my special USR command. Since it is in the high RAM area (sometimes called "language card"), I reference $C083 twice to write enable the RAM.

Once the USR vector is loaded, typing a command "USR" will execute my code. When this happens, the entire command I typed will be in a buffer starting at $200. Some routines exist inside S-C Macro which can help in parsing the command further and in implementing its functions, and I will use them in this example. If you have the source code to one of the S-C Macro versions, it is not too difficult to find these routines. And if you don't have it, you can always disassemble and analyze, a true form of adventure. The addresses shown in lines 1040-1060 correspond to version 2.0 of the S-C Macro Assembler.

Line 1165 calls on a subroutine I call PARSE.LINE.RANGE (PLR). PLR starts by setting up SRCP to point to the beginning of the source program, and ENDP to the end of same. Then it looks at the command line for various forms of line numbers. You might have none at all, in which case PLR is finished. You might have one number alone, or a period. (A period is shorthand for the last remembered line number.) That might be preceded by or followed by a comma. You might have two numbers separated by a comma. Here is a table showing what happens in each case:

| | SRCP | ENDP | CARRY |
|---|---|---|---|
| none | pstart | pend | set |
| # | #start | #end | clear |
| #, | #start | pend | clear |
| ,# | pstart | #end | clear |
| #1,#2 | #1start | #2end | clear |

```
       where # means number or "."
             pstart = address of start of source code
             pend = address of end of source code
             #start = address of starting line #
             #end = address of ending line #
```

Line 1170 call a routine in the assembler to compare SRCP and ENDP to see if we are finished or not. The code is simply:

```
               LDA SRCP
               CMP ENDP
               LDA SRCP+1
               SBC ENDP+1
```

Lines 1200-1210 pick up the first character after the line number. The source line format in memory is one byte for a byte count, two bytes for the line number, the text of the line, and a final terminating 00 byte. The blank which follows just after the line number in listings is not actually stored.

Characters in a source line are stored in "low" ASCII, values between $01 and $7F. Values from $81 through $BF indicate 1 to 63 blanks. The value $C0 indicates repetitions of some other character. The byte following a $C0 is the repetition count, and the byte after that is the character to be repeated. Lines 1220-1240 check for blanks and repeat tokens. Lines 1340-1350 pick up the repeated character if we found a repeat token.

Lines 1360-1390 check if the first character is a letter. If not, this line will not be listed. Lines 1250-1320 are executed to skip over the current line without listing it. Since the first byte of the line has a byte count, it is added to SRCP to move up the next line.

At line 1400 I call LIST.CURRENT.LINE to ... you guessed it. This subroutine also advances SRCP, so after it is finished I jump back to the top to check pointers and get the next line.

After assembling the program, I type MGO INIT to hook it in. Then "USR 1070," would list just lines 1080 and 1160.

```
=============================================================================
DOCUMENT :AAL-8504:Articles:LovesConversion.txt
=============================================================================
```

Improving the Single-Byte Converter................Bruce Love
New Zealand

Bob's single byte converter (see Jan 85 issue, pages 31-32) can be shortened by one byte. The left column is from Bob's code, the right a shorter version:

```
       1040 .1  LDX #"0"              .1  LDX #"0"-1
       1050 .2  CMP DECTBL,Y              SEC
       1060     BCC .3                .2  SBC DECTBL,Y
       1070     SBC DECTBL,Y              INX
       1080     INX                       BCS .2
       1090     BNE .2                    ADC DECTBL,Y
```

I also tried a different approach, using the decimal mode to count tens, then printing the tens as a hex value with the monitor routine at $FDDA and the remainder (units digit) with $FDED. This routine takes longer time, but does not need to use the X-register.

==============================================================================
DOCUMENT :AAL-8504:Articles:Micro.Magic.txt
==============================================================================

A Powerful 65816 Board on the Horizon......Bob Sander-Cederlof

Some of you may have heard of Micro Magic, a company in Mary- land that is planning to produce a plug-in card for your Apple with fast RAM and a fast 65816. Well, if not, now you have.

I spoke yesterday with Will Troxell, and got an overview of their plans. He and Frank Krol are working together on the project. Their goal is to produce the most powerful and flexible card they can and yet still bring it in for a low price. The card will basically be similar to the Accelerator //e, in that it consists of a fast microprocessor, fast RAM, and the logic to take control away from the 6502 or 65C02 on your Apple motherboard.

But instead of a 65C02 running at 3.58 MHz, you will get a 65816 running at 6 MHz. Instead of one row of RAM chips, you get two. Troxell's board will probably come with 64K or 128K of 6MHz dynamic RAM, but later this year they have been promised that 256K RAMs fast enough for 6 MHz operation will be in production; then you will be able to expand your board to 256K or 512K bytes of RAM.

There is a firmware socket on the board which can accept a 27128 (16K bytes of firmware, the same as you find in a //c). They do not plan to include any firmware at the beginning, but it certainly can be filled up with your own goodies.

There are two external connectors on the board. One of these allows you to add another 512K RAM. Remember, this is directly addressable RAM, not bank-switched. The 65816 can directly address up to 16 megabytes, with its 24-bit address bus.

It is also exciting to remember that a plain ol 6502 running at 1 MHz (what you have now) is roughly equivalent in speed to most of the 8088 and Z-80 computers on the market. A 6 MHz 6502 could beat a 20MHz Z-80 (were they to make one so fast). A 6 MHz 65816 will beat out 68000's, 80286's, and so on. Why is this true? Because all those other chips use micro- programmed instruction sets, taking many clock cycles for each instruction. The 6502 and its progeny are fully implemented in hardware gates, so only a handful of clock cycles are needed.

Furthermore, a 65816 instruction will take from one to four bytes of memory, while a 68000 instruction will take 2, 4, 6, 8, or 10 bytes. Now I am not trying to deny the power of some of those 68000 instructions. One of them may take many steps in 65816 code. Especially if you need to deal with 32-bit operands. But it is my experience that those super instructions are relatively infrequent in practical programs. Most programs spend most of their time just moving bytes from here to there and back again.

Now if we could only get one! For about fifteen months we have been hearing "in two to four weeks". We could despair, were it not for our historical perspective. The same thing happened with the 65C02, and now we really do have them in abundance. By this time next year, you may be hearing solid confirmation of the rumor (heard this week) that Apple and GTE are discussing large orders of 65816s.

But I digress. Back to Troxell and Krol. There new board will be called the MAX-816, and a new operating system they are designing for it will be MAX-OS. A special circuit on the card will optimize memory re-mapping for both DOS and ProDOS, automatically, so that maximum possible use is made of the fast RAM on the card. THe fewer times the card has to slow down to use motherboard RAM, the faster your programs fly.

MAX-OS will not be necessary for you to get a bang out of MAX-816, because it will work like the Accelerator //e and make most existing programs six times faster (exclusive of I/O). But when it is ready, it will open up new vistas, with RAM stretching out in every direction as far as the eye can see. In a design reminiscent of one from a certain large phone company, the kernel is written in assembly language, with a C-shell wrapped around it.

Personally, I am no great fan of complex operating systems. The simpler and smaller the better, in my book. I still like DOS 3.3, especially with enhancements I regularly patch in. Nevertheless it does take more management when you have the magnitude and variety of resources that will be in the Apple of the future. Maybe MAX-OS will be the winner.

If Will and Frank are whetting your appetite, you can write to them at Micro Magic, Box 281, Millersville, MD 21108. Or you might be able to reach them at (301) 987-6083.

```
================================================================================
DOCUMENT :AAL-8504:Articles:My.Ad.txt
================================================================================

S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code for Version 1.1 (on two disk sides).....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code)....................$20
S-C Cross Reference Utility (with complete source code)..............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code).......................$50
DP18 Source and Object...............................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                  Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains        1980      -         -         -         1
 the source code from      1981      2         3         4         5
 three issues of AAL,      1982      6         7         8         9
 saving you lots of        1983     10        11        12        13
 typing and testing.       1984     14        15        16        17
                           1985     18

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Visible Computer: 6502 (Software Masters).................(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)........................ package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers............................  6 cents each

quikLoader EPROM System (SCRG)..............................($179) $170
PROmGRAMER (SCRG).........................................($149.50) $140
D MAnual Controller (SCRG)...................................($90)  $85
Switch-a-Slot (SCRG)........................................($190) $175
Extend-a-Slot (SCRG).........................................($35)  $32
Write Guard Disk Mod Kit (Mark IV)...............................  $45

Books, Books, Books..........................compare our discount prices!

"Inside the Apple //c", Little.............................($19.95)  $18
"Inside the Apple //e", Little.............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler......................($22.95)  $21
"Understanding the Apple II", Sather........................($22.95)  $21
"Understanding the Apple //e", Sather.......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
```

```
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
"Incredible Secret Money Machine", Lancaster.................($7.95)   $7
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
"What's Where in the Apple", Second Edition.................($19.95)  $19
"6502 Assembly Language Programming", Leventhal.............($18.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18
```

Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.


*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***<br>
*** (214) 324-2050 ***<br>
*** We accept Master Card, VISA and American Express ***

=============================================================================
DOCUMENT :AAL-8504:Articles:ProDOS.numout.txt
=============================================================================

Shrinking Code Inside ProDOS...............Bob Sander-Cederlof

David Johnson challenged me a few days ago. We were talking about ProDOS: the need for a ProDOS version of the S-C Macro Assembler, the merits vs. enhanced DOS 3.3, and the rash of recent articles on shrinking various routines inside DOS to make room for more features.

I've been avoiding ProDOS as much as possible, trying not to notice its ever-increasing market-share. Dave's comment, "ProDOS is a fertile field for your shrinking talent," may have finally pushed me into action.

I am trying to make the ProDOS version of the S-C Macro Assembler, but is hard. I have Apple's manuals, Beneath Apple ProDOS, and the supplement to the latter book which explains almost every line of ProDOS code. Nevertheless, version 1.1.1 of ProDOS doesn't seem to conform to all these descriptions in every particular. I spent four hours last night chasing one little discrepancy. (Turned out to be my own bug, though.)

In the process, I ran across the subroutine ProDOS uses to convert binary numbers to decimal for printing. In version 1.1.1 it starts at $A62F, and with comments looks like this.

<<<< prodos listing >>>>

The conversion routine is designed to handle values between 0 and $FFFFFF. The heghest byte must already have been stored at ACCUM+2 before calling CONVERT.TO.DECIMAL. The middle byte must be in the X-register, and the low byte in the A-register. The decimal digits will be stored in ASCII in the $200 buffer, starting and $201+Y and working backwards.

One way of converting from binary to decimal is to perform a series of divide-by-ten operations. After each division, the remainder will be the next digit of the decimal value, working from right to left. That is the technique ProDOS uses, and the division is done by the subroutine in lines 1280-1420.

The dividend is in ACCUM, a 3-byte variable. The low byte is first, then the middle, and finally the high byte. One more byte is set aside for the remainder. A 24-step loop is set up to process all 24 bits of ACCUM. In the loop ACCUM and REMAINDER are shifted left. If REMAINDER is 10 or more, it is reduced by ten and the next quotient bit set to 1; otherwise the next quotient bit is 0.

The first possible improvement I noted was in the area of lines 1330-1360. the ROL REMAINDER will always leave carry status clear, because we never let REMAINDER get larger than 9. If we delete the SEC instruction, and change SBC #10 to SBC #9 (because carry clear means we need to borrow), we can save one byte. But that's not really worth the effort.

Next I realized that REMAINDER could be carried in the A-register within the 24-step loop, and not stored until the end of the loop. Here is that version, which saves seven bytes (original = 31 bytes, this one = 24 bytes):

<<<< listing of my lines 1260-1380 >>>>

To make sure my version really worked, I re-assembled the conversion program with an origin of $800, and appended a little test program. Here is my test program, which converts the value at $0000...0002 and prints it out.

<<<<listing of my lines 1510-1620 >>>>

My best version is yet to come. I considered the fact that we could SHIFT the next quotient bit into the low end of ACCUM rather than using INC ACCUM to set a one-bit. I rearranged the loop so that the remainder reduction was done first, followed by the shift-left operation. I had to change the remainder reduction to work modulo 5 rather than 10, because the shifting operation came afterwards. I also had to inlcude my own three lines of code to ROL ACCUM, since the little subroutine in ProDOS started with ASL ACCUM. The result is still shorter than 31 bytes, but only four bytes shorter. Nevertheless, it is faster and neater, in my opinion.

<<<<lines 1640-1770>>>>

================================================================================
DOCUMENT :AAL-8504:Articles:Q.n.A.txt
================================================================================

Questions and Answers

I noticed in your article about making DOS-less disks that you shortened the catalog so as to make some of the track 17 sectors available.  That's nice, but DOS will not allocate sectors out of track 17 unless a small patch is made to the allocation routine.

Change the byte at $B292 from $69 to $A9, and those sectors your freed will be usable.

Larry Anderson

Thanks, Larry!  You are exactly right.  Evidently FID uses a different scheme for allocation, because when I filled up my DOS-less disk my sectors in track 17 were used.  If I had used LOAD-SAVE to move the files, those sectors would have not been found.  From now on I am going to use your patch.

```
=====================================================================
DOCUMENT :AAL-8504:Articles:QuikLoader.Euge.txt
=====================================================================
```

Putting S-C Macro on a QuikLoader Card..........Jan Eugenides

The QuikLoader by Southern California Research Group is one of those rare devices that causes you to wonder how you ever got along without one. I have had mine for about a year now, and I would never go back to the old way of loading programs!

Briefly, the QuikLoader allows you to put whatever programs you desire on EPROMS, which then plug into the QuikLoader. EPROMS from 2716-27512 can be used, for a possible 512K bytes of program space on one QuikLoader (equivalent to four Apple floppies!). You can have more than one card, of course, so there's lots of room available for just about anything. The QuikLoader also comes with DOS 3.3 already installed, along with FID, and COPYA. When you turn on your machine, you'll hear a little whoop instead of the familiar beep. DOS has just been loaded in about 2 seconds. No more booting! In fact, I seldom put DOS on a disk anymore, and I can use the space for programs instead.

Programs which are on the QuikLoader can be loaded into RAM and executed in about 2 seconds, with just two keystrokes! Since they are loaded into their regular RAM locations, they do NOT need to be modified in any way.

You can see a catalog of the QuikLoader by typing "Q" followed by RESET. The program names appear with letters A-Z next to them. Then you can select and run the programs by typing the letter corresponding to that program. Alternatively, if you want to run the primary routine on a chip, just press the number of the socket it is in followed by RESET. More on this later.

Putting programs on the QuikLoader is somewhat problematical, however. The manual is STILL in it's draft form, although they have been promising a better one for over a year. Oh well...a little trial and error is good for the soul.

In order to put the S-C Macro Assembler on the QuikLoader, it is necessary to write what's known as a "primary" routine. The QuikLoader has a built-in operating system which allows you to move blocks of memory to their RAM locations from the various EPROMS on the QuikLoader card, and then execute them however you wish. The following program is intended to be used on a 27128 EPROM, which will hold the entire S-C Macro Assembler, with driver (I used the Ultraterm driver for this program) and the Fast Bload patches, which I chose to load between DOS and its buffers, rather than actually patch the DOS. You can do it either way, it's up to you.

This program is called the "overhead" for the EPROM. It goes at $FEB0 in the actual chip. The catalog must appear at $FF00. These are the addresses as the Apple would see them, not the absolute addresses relative to the chip. A 27128 will address as though it runs from $C000 to $FFFF as far as the Apple is concerned. In other words, the chip's address $0000 equals the Apple's address $C000. Things are further complicated by the fact that an Apple II+ cannot address the range from $C000 to $C7FF without a small circuit modification. In this case it's no problem, the space from $C800-$FFFF is more than enough to house the entire assembler. If you needed more space, you could put your primary routine in the $C000-$C7FF space.

The rest of the EPROM contains the code for the assembler itself, and the fast Bload patch. The assembler goes from $C800-$EFFF, and the Bload patch from $F000 to $F04D. You must pack these files together in RAM somewhere prior to burning the chip. In other words, Bload the assembler at, say, $2800-4FFF. Put the Bload patch at $5000-504D Then Bload the overhead program at $5EB0. The rest of the EPROM doesn't matter. Then burn all this stuff into the EPROM starting at $800 relative to the chip. Thus, when you install the chip on the card, it will show up at $C800-FFFF like it should. If your EPROM burner won't burn partial chips, just start the burn from $2000 and it'll work out.

That's it. Just install the chip on the QuikLoader in any socket. To run the assembler just type the socket number followed by RESET. In two seconds the assembler will load and start! No more waiting to boot DOS, load the program, etc. You don't even have to look for a disk! Sure speeds up the work.

This should help augment the information in the manual a little, and get you on your way. I have installed the S-C assembler, Rak-ware's DISASM, a modified SOURCEROR (it now ouputs S-C format code, heh heh), the S-C Word Processor, a terminal program of my own design (it's capture buffer exactly coincides with the S-C Word Processor buffer! I can come off-line and begin editing with two keystrokes, and no disk access!), and some other utilities. All stored inside the Apple, available instantly at any time. For $170 (the price from S-C Software), the QuikLoader is a MUST.

By the way, for a reasonable fee I will install programs on EPROMS for you. You supply the programs and EPROMS, and I'll do the rest. Some programs are not suitable...particularly those which access the disk a lot. They would require extensive modification and that's best left to the original author. Also, copy-protected stuff cannot be loaded, because there's no way to ge at the files. Contact me if you're interested, at 11601 NW 18th St., Pembroke Pines, FL 33026.

[ For $20, S-C Software will send registered owners of version 2.0 a 27128 with the S-C Macro Assembler on it. This adds five lines to the QuikLoader menu, allowing you to choose the screen driver you wish. Only the $D000 (language card) version is provided. ]

Here's the overhead program, with GETSLOT overhead taken from the QuikLoader manual.

=============================================================================
DOCUMENT :AAL-8504:Articles:Review.Sider.txt
=============================================================================

Review of the FCP Hard Disk................Bob Sander-Cederlof

First Class Peripherals has been advertising for some months now their 10 megabyte hard disk system (The Sider) for the Apple. At only $695, including drive, controller, cable, and software, it sounds too good to be true. We called them and asked for a chance to write a review, and they loaned us one for a month.

I first tried hooking it up to an Apple II Plus, the same one we have used with hard disks in the past. However, after 5 or 6 wasted hours, it still would not function. We could not even get the disk to completely initialize. I finally called the 800 number for customer service, and found out that there have been problems hooking the Sider to some II+'s. They suggested trying it on a //e before giving up. Sure enough, it worked perfectly on our //e. The Sider is sold subject to a 15-day trial period, so there is plenty of time to find out if it will work with your II+.

I am very pleased. The Sider works well, looks good, and is not too noisy. We have heard of at least one customer who did complain of the noise level, but I have never listened to a quieter one. Because of the venting design there is no internal fan, so the only noise is the spinning disk. Anyway, my office already has two fans going on Apples and another in a Minolta copier. The Sider nicely masks them all.

The size and shape are nice, too. It is somewhat smaller than I expected: less than 4x8x16 inches. At first I set it along side of my Apple (after all it is called the Sider), but now it is along the back edge of my work table. This way it takes practically no space at all, yet I can still easily reach the on/off switch.

The installation software that comes with the Sider initializes the 10 megabytes into four separate partitions. One is for DOS, one for ProDOS, one for CP/M, and one for Pascal. You can vary the partition size for each one, although a certain minimum amount must be allocated; you cannot squeeze one all the way out. The DOS partition allows a combination of floppy size volumes and large volumes. The large volumes give you three times the amount of a regular Apple floppy. I set mine up with 32 small volumes and one large volume.

The ProDOS partition divides the allocated space into two equal size volumes, designated /HARD1/ and /HARD2/. Since I shrank CP/M and Pascal to the minimum, the ProDOS volumes are about 2.5 megabytes each.

If you want to change the partitions, you have to completely re-initialize. That means all your files will disappear. Of course you can restore them from your backup floppy copies.

The only modification to DOS 3.3 that the Sider makes is to put a call to their firmware at $BD00. I decided to apply my own set of patches, which among other things speed up LOAD, BLOAD, RUN, and BRUN. They were not only compatible, they even speeded up the hard disk! Here is a table comparing the Sider with floppies, both with and without my patches:

```
BLOAD              ----floppies-----  ----The Sider----
       # sectors  standard  patched  standard  patched
       ---------------------------------------------
          22         7.7      3.8       3.0      1.3
          69        18.7      5.6       6.7      2.4
         131        32.6      8.6      12.3      3.8
```

I also timed the assembly of a large program, whose source was on two disks (the S-C Macro Assembler itself, in fact). With my speed up patches the floppy assembly took 4 minutes 50 seconds; the Sider with standard DOS took 3 minutes 50 seconds; the Sider with my patches took only 2 minutes 32 seconds.

All these times are under DOS 3.3 of course. ProDOS is about the same as my patched version of DOS in speed, but has other advantages like larger volumes and files.

The main competition for the Sider comes from the two most popular companies, Apple and Corvus. Apple's ProFILE hard disk is sleek and nice, and only costs three times what the Sider does. Since you are paying more, you also get less: Apple only supports ProDOS. The ProFILE doesn't work with CP/M, Pascal, or DOS 3.3. (Unless there is a new ProDOS compatible Pascal.) Corvus costs even more than ProFILE, last time I checked. On the other hand, they have an excellent reputation.

Its always hard to trust some new little company, even when they have a great product and price. Just who is First Class Peripherals, anyway? Well, they are a subsidiary of Xebec, one of the bigger makers of hard disks. Xebec has been around a long time (over ten years) and has a first class reputation. I think we can depend on them. The Sider comes with a one-year limited warranty, which I think means that if it breaks you send it in and they will fix it or replace it. (Note: a whole year, not just 90 days!) After the warranty has expired there is a flat $150 charge for repairs.

The only way to buy a Sider is directly from First Class Peripherals. You can call them at 1-800-538-1307, or write to 2158 Avenue C, Bethlehem, PA 18001. If you are in a user group of significant size, I understand someone at FCP might want to visit with a demo unit. You might give them a call.

```
===============================================================
DOCUMENT :AAL-8504:DOS3.3:Asm2.0FastBLOAD.txt
===============================================================

 1000  *SAVES.ASSEM.2.0.OH(FAST BLOAD)
 1010  *--------------------------------
 1020  *1/31/85
 1030  *--------------------------------
 1040  *
 1050  *S-C MACRO ASSEMBER OVERHEAD - ULTRATERM VERSION
 1060  *               by Jan Eugenides
 1070  * 3/9/85
 1080  *
 1090  *--------------------------------
 1100  *CHIP 0 ROUTINE EQUATES
 1110  *--------------------------------
 1120  *Y-register indexes of the chip 0 routines
 1130  *--------------------------------
 1140  *
 1150  MOVEBLK    .EQ 0       Move data block to RAM
 1160  GOMRBRD    .EQ 8       Go to motherboard
 1170  *--------------------------------
 1180  *
 1190  * GENERAL EQUATES
 1200  *
 1210  *--------------------------------
 1220  PRISLOT    .EQ $26  Storage for primary slot
 1230  QLMAP      .EQ $2D  bitmap of QL slots
 1240  SRCL       .EQ $3A  indirect source
 1250  SAVCTRL    .EQ $20A save control word
 1260  QLCTRL     .EQ $C081  QL control register
 1270  *--------------------------------
 1280  *
 1290  * GET SLOT EQUATES
 1300  *
 1310  *--------------------------------
 1320  QLOFF      .EQ $18  00011000 QLOFF; CHIP 0
 1330  CHKNUM     .EQ $20  NUMBER OF FIND SLOT CHECKS
 1340  GSCL       .EQ $40  GET SLOT C PARAMETER.
 1350  GSCH       .EQ $41
 1360  GSEL       .EQ $42  GET SLOT E PARM
 1370  GSEH       .EQ $43
 1380  SLTXROM    .EQ $C006     IIE SOFT SWITCH
 1390  INT3ROM    .EQ $C00A         "
 1400  SLT3ROM    .EQ $C00B         "
 1410  CLRROM     .EQ $CFFF
 1420  *--------------------------------
 1430         .OR $FEB0
 1440         .TF ASM.2.0.OH
 1450  *--------------------------------
 1460  * This program will start the assembler in 80x32
 1470  * mode with ultraterm.  Assumes that assembler has
 1480  * been patched at $DBC9 and $DC11 for 32 line mode,
```

```
1490  * i.e. the normal $17 is now $1F.  If mode is changed
1500  * these bytes must be re-patched. ($2F for 48 line mode)
1510  * For S-C assember 2.0 March 1985 version with Bob's
1520  * ultraterm driver attached at $F700.
1530  *--------------------------------
1540  START.PROG LDA #0    Turn on Ultraterm
1550             JSR $C300
1560             LDA #22   bring up in 80x32 mode
1570             JSR $FDED
1580             LDA #"5   Mode 5
1590             JSR $FDED
1600             LDA #$CB
1610             STA $3D1  set warmstart vector
1620             LDA #0
1630             STA $9D00 make room between DOS a buffers
1640             JSR $A7D4 for fast BLOAD patch
1650             LDA #$30
1660             STA $ACA6 patch dos to call fast Bload
1670             LDA #$9C
1680             STA $ACA7 which is now at $9C30
1690             LDA #$4C
1700             STA $E000
1710             LDA #0
1720             STA $E001  put assembler coldstart vector at $E000
1730             LDA #$D0
1740             STA $E002
1750             LDA $C080 select ram card
1760             JMP $D000 coldstart assembler
1770  SP.END
1780  *--------------------------------
1790         .BS $FF00-*   SKIP TO FF00
1800  *--------------------------------
1810  *KATALOG ENTRIES START HERE
1820  *--------------------------------
1830  ASMK   .DA #$90     PRIMARY
1840         .DA N.RESET SOURCE
1850         .DA $0000    LENGTH
1860         .DA $0000    DESTINATION
1870         .AS -"ASM"
1880  *--------------------------------
1890         .DA #$86     END OF KAT RECORD
1900  *--------------------------------
1910  ASMPARM1   .DA $C800     SOURCE  assembler + driver goes here
1920             .DA $27FF     LENGTH  will load from $D000-$F7FF
1930             .DA $D000     DESTINATION
1940  ASMPARM2   .DA $F000     SOURCE  fast bload routine
1950             .DA $004D     LENGTH
1960             .DA $9C30
1970  *--------------------------------
1980  INVERT     LSR
1990             ROR
2000             ROR
2010             ROR
2020             AND #$E0
```

```
2030            STA SAVCTRL
2040            RTS
2050 *--------------------------------
2060        .BS $FF53-*   SKIP TO FF53
2070 OFFLP  LDA #QLOFF
2080        STA QLCTRL,X       TURN OFF THE QL
2090 RTSLOC JSR GETSLOT        THIS INSTRUCTION AT $FF58 (RTS)
2100        BNE OFFLP
2110 *--------------------------------
2120 *
2130 * FIND SLOT NUMBER BY COMPARING CNXX TO ENXX FOR EACH SLOT
2140 * START WITH SLOT 7.  USR MUST BE RESET FOR SEARCH TO BE
2150 * EFFECTIVE IN II OR IIE.
2160 *--------------------------------
2170 GETSLOT    STA SLTXROM    ENABLE IIE I/O SELECTS
2180            STA SLT3ROM
2190            LDA #0
2200            STA GSCL
2210            STA GSEL
2220 TRYAGEN    LDA #$C1       START WITH SLOT 1
2230            STA GSCH
2240            LDA #$E1
2250            STA GSEH       DESTINATION = $EN00
2260            LDY #CHKNUM    GET NUMBER OF CHECKS TO VERIFY
2270 LOOKLP     LDA (GSCL),Y
2280            CMP (GSEL),Y
2290            BNE NOTHERE    BRANCH IF QL NOT IN THIS SLOT
2300            DEY
2310            BNE LOOKLP
2320            LDA GSCH
2330            TAY
2340            ASL            GET SLOTNUM TIMES $10 TO X
2350            ASL
2360            ASL
2370            ASL
2380            TAX
2390            LDA $FE86,Y  GET BIT MAP
2400            ORA QLMAP
2410            STA QLMAP       SET BIT IN QLMAP
2420            STA INT3ROM     LEAVE INT3ROM AS NORMAL RESET DOES
2430 *--------------------------------
2440 *NORMAL RESET FORCES SLTXROM
2450 *LEAVE 3ROM AND XROM AS WITH NORMAL RESET
2460 *--------------------------------
2470        LDA CLRROM   EXPANSION ROM OFF
2480        RTS
2490 NOTHERE    INC GSCH
2500            INC GSEH       CHECK IN NEXT SLOT
2510            BNE LOOKLP     BRANCH ALWAYS
2520 *--------------------------------
2530 *EQU $C0 SHOULDN'T OCCUR; BOMB IF DOES
2540 *--------------------------------
2550 MAP    .DA #$80
2560        .DA #$40
```

```
 2570         .DA #$20
 2580         .DA #$10
 2590         .DA #$08
 2600         .DA #$04
 2610         .DA #$02
 2620  *--------------------------------
 2630  * THIS IS N.RESET ROUTINE OF THIS CHIP
 2640  *--------------------------------
 2650  N.RESET    JSR INVERT       Invert the control word
 2660             LDY #5
 2670  .1         LDA ASMPARM1,Y   Move ASSEMBLER parms
 2680             STA SRCL,Y
 2690             DEY
 2700             BPL .1
 2710             LDA SAVCTRL      get control word
 2720             LDX PRISLOT      slot in X reg
 2730             LDY #MOVEBLK     Command index for move block
routine
 2740             JSR GOCHIP0      Call chip 0 to move block
 2750             LDY #5
 2760  .2         LDA ASMPARM2,Y   Move Fast Bload routine parms
 2770             STA SRCL,Y
 2780             DEY
 2790             BPL .2
 2800             LDA SAVCTRL      get control word
 2810             LDX PRISLOT      Slot in X reg
 2820             LDY #MOVEBLK     Command index - move block
 2830             JSR GOCHIP0      Call chip 0
 2840             LDY #SP.END-START.PROG
 2850  .3         LDA START.PROG,Y Move startup program to $300
 2860             STA $300,Y
 2870             DEY
 2880             BPL .3
 2890             LDA #$02        put address-1 on stack
 2900             PHA
 2910             LDA #$FF
 2920             PHA
 2930             LDY #GOMRBRD     jmp to $300 to start
 2940             LDA SAVCTRL
 2950             LDX PRISLOT
 2960             JMP GOCHIP0
 2970  *--------------------------------
 2980         .BS $FFEC-*    SKIP TO FFEC
 2990  GOCHIP0    STA QLCTRL,X  GO TO CHIP 0
 3000             JMP N.RESET   DO N.RESET ROUTINE OF THIS CHIP
 3010             .BS 3
 3020             RTS
 3030             .BS 2
 3040             .DA ASMK      FIRST KATALOG LOCATION
 3050             .DA $3FB      NMI VECTOR
 3060             .LIST OFF
```

```
=========================================================================
DOCUMENT :AAL-8504:DOS3.3:S.Hard.Cat.txt
=========================================================================

 1000  *SAVE S.HARD CAT
 1010  *---------------------------------
 1020  RWTS       .EQ $03D9
 1030  GETIOB     .EQ $03E3
 1040  *---------------------------------
 1050  CATALOG.BUFFER .EQ $B4BB
 1060  *---------------------------------
 1070  R.VOLUME   .EQ $B5F9
 1080  *---------------------------------
 1090         .OR $803
 1100         .TF CAT
 1110  *---------------------------------
 1120  HARD.CAT
 1130         JSR $FD8E
 1140         LDA #$FE     FOR VOLUME=1 TO 254
 1150         STA R.VOLUME      (.EOR.FF OF VOLUME #)
 1160  *---PATCH DOS TO TRAP ERROR------
 1170  .1     LDA #$60     'RTS'
 1180         STA $B09E
 1190         JSR $AFF7    READ VTOC OF VOLUME
 1200  *---REMOVE PATCH-----------------
 1210         LDA #$B0     'BCS'
 1220         STA $B09E
 1230         BCS .7       OUT OF LOOP, BEYOD LAST VOLUME
 1240  *---READ 1ST CATALOG SECTOR------
 1250         CLC
 1260         JSR $B011
 1270  *---PRINT VOLUME #---------------
 1280         LDA R.VOLUME      INVERSE OF #
 1290         EOR #$FF          BACK TO NORMAL FORM
 1300         LDX #"0"          CONVERT TO DECIMAL
 1310  .2     CMP #10           ANY 10'S?
 1320         BCC .3            ...NONE LEFT
 1330         SBC #10           ...YES, DIMINISH
 1340         INX                   AND COUNT IT
 1350         BNE .2            ...ALWAYS
 1360  .3     PHA               SAVE UNITS
 1370         TXA               PRINT TENS
 1380         JSR $FDED
 1390         PLA               GET UNITS
 1400         ORA #"0"          AND PRINT IT
 1410         JSR $FDED
 1420         LDA #" "          PRINT "  "
 1430         JSR $FDED
 1440         JSR $FDED
 1450  *---PRINT NAME OF FIRST FILE-----
 1460         LDY #11
 1470         LDA $B4BB,Y
 1480         BEQ .8       ...EMPTY VOLUME
```

```
       LDX #0
.4     LDA $B4BB+3,Y
       INY
       JSR $FDED
       INX
       CPX #30
       BCC .4
*---PRINT CARRIAGE RETURN--------
.5     JSR $FD8E
*---NEXT VOLUME------------------
       DEC R.VOLUME
*---POSSIBLE PAUSE OR ABORT------
       LDA $C000    ANY KEY PAUSES
       BPL .1       NO KEY
       STA $C010
       CMP #$8D     <RETURN> ABORTS
       BEQ .7
.6     LDA $C000    PAUSE LOOP
       BPL .6
       STA $C010
       CMP #$8D     AGAIN, RETURN AGORTS
       BNE .1
*--------------------------------
.7     JSR $FD8E    <RETURN>
       JMP $3D0     BACK TO DOS
*---EMPTY VOLUME-----------------
.8     LDX #0
.9     LDA MT,X     PRINT STRING BELOW
       BEQ .5
       JSR $FDED
       INX
       BNE .9       ...ALWAYS
*--------------------------------
MT     .AS -/<<<EMPTY VOLUME>>>/
       .HS 00
*--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8504:DOS3.3:S.List.Mjr.Lbl.txt
==============================================================================
```

```
 1000  *SAVE S.LIST MAJOR LABELS
 1010  *--------------------------------
 1020  SRCP   .EQ $DD,DE
 1030  *--------------------------------
 1040  PARSE.LINE.RANGE    .EQ $DEAF OR 1EAF
 1050  CMP.SRCP.ENDP       .EQ $DF11 OR 1F11
 1060  LIST.CURRENT.LINE   .EQ $D737 OR 1737
 1070  *---LINK COMMAND----------------
 1080  INIT   LDA $C083    ENABLE LANGUAGE CARD
 1090         LDA $C083
 1100         LDA #USR.LIST    SET UP USR VECTOR
 1110         STA $D007
 1120         LDA /USR.LIST
 1130         STA $D008
 1140         RTS
 1150  *---USR COMES HERE---------------
 1160  USR.LIST
 1165         JSR PARSE.LINE.RANGE
 1170  .1     JSR CMP.SRCP.ENDP
 1180         BCC .2
 1190         RTS
 1200  .2     LDY #3       POINT TO FIRST CHAR
 1210         LDA (SRCP),Y
 1220         BPL .5       NOT TOKEN
 1230         CMP #$C0
 1240         BCS .4       REPEAT TOKEN
 1250  .3     LDY #0       SKIP TO NEXT LINE
 1260         LDA (SRCP),Y LINE LENGTH
 1270         CLC
 1280         ADC SRCP
 1290         STA SRCP
 1300         BCC .1
 1310         INC SRCP+1
 1320         BNE .1       ...ALWAYS
 1330  *--------------------------------
 1340  .4     LDY #5       POINT AT RPTD CHAR
 1350         LDA (SRCP),Y
 1360  .5     CMP #'A'
 1370         BCC .3       NOT LETTER
 1380         CMP #'Z'+1
 1390         BCS .3       NOT LETTER
 1400         JSR LIST.CURRENT.LINE
 1410         JMP .1
 1420  *--------------------------------
```

===========================================================================
DOCUMENT :AAL-8504:DOS3.3:S.PD.NUMOUT.SC.txt
===========================================================================

```
1000  *SAVE S.PRODOS NUMOUT (SC)
1010  *--------------------------------
1020  *--------------------------------
1030  *   CONVERT 00.XX.AA FROM BINARY TO DECIMAL
1040  *   STORE UNITS DIGIT AT $201,Y
1050  *   STORE OTHER DIGITS AT SUCCESSIVE LOWER ADDRESSES
1060  *
1070  *      Note:  it is assumed and required that
1080  *             ACCUM+2 already by zeroed!
1090  *             Either that, or already set to the
1100  *             highest byte of a 24-bit value.
1110  *--------------------------------
1120  CONVERT.TO.DECIMAL
1130         STX ACCUM+1
1140         STA ACCUM
1150  .1     JSR DIVIDE.ACCUM.BY.TEN
1160         LDA REMAINDER
1170         ORA #"0"
1180         STA BUFFER+1,Y
1190         DEY
1200         LDA ACCUM    CHECK IF QUOTIENT ZERO
1210         ORA ACCUM+1
1220         ORA ACCUM+2
1230         BNE .1
1240         RTS
1250  *--------------------------------
1260  DIVIDE.ACCUM.BY.TEN
1270         LDX #24      24 BITS IN DIVIDEND
1280         LDA #0       START WITH REM=0
1290  .1     JSR SHIFT.ACCUM.LEFT
1300         ROL
1310         CMP #10
1320         BCC .2       STILL < 10
1330         SBC #10
1340         INC ACCUM    QUOTIENT BIT
1350  .2     DEX          NEXT BIT
1360         BNE .1
1370         STA REMAINDER
1380         RTS
1390  *--------------------------------
1400  ACCUM      .BS 3
1410  REMAINDER  .BS 1
1420  BUFFER     .EQ $0200
1430  *--------------------------------
1440  *--------------------------------
1450  SHIFT.ACCUM.LEFT
1460         ASL ACCUM
1470         ROL ACCUM+1
1480         ROL ACCUM+2
```

```
1490        RTS
1500 *--------------------------------
1510 T      LDA 0
1520        STA ACCUM+2
1530        LDX 1
1540        LDA 2
1550        LDY #10
1560        JSR CONVERT.TO.DECIMAL
1570 .1     INY
1580        LDA BUFFER+1,Y
1590        JSR $FDED
1600        CPY #10
1610        BCC .1
1620        RTS
1630 *--------------------------------
1640 DIVIDE.ACCUM.BY.TEN.SHORTEST
1650        LDX #24      24 BITS IN DIVIDEND
1660        LDA #0       START WITH REM=0
1670 .1     CMP #5
1680        BCC .2       STILL < 10
1690        SBC #5
1700 .2     ROL ACCUM
1710        ROL ACCUM+1
1720        ROL ACCUM+2
1730        ROL
1740        DEX          NEXT BIT
1750        BNE .1
1760        STA REMAINDER
1770        RTS
1780 *--------------------------------
9999   .LIF
```

=================================================================
DOCUMENT :AAL-8504:DOS3.3:S.ProDOS.NUMOUT.txt
=================================================================

```
 1000  *SAVE S.PRODOS NUMOUT
 1010  *--------------------------------
 1020         .OR $A62F
 1030         .TA $800
 1040  *--------------------------------
 1050  *   CONVERT 00.XX.AA FROM BINARY TO DECIMAL
 1060  *   STORE UNITS DIGIT AT $201,Y
 1070  *   STORE OTHER DIGITS AT SUCCESSIVE LOWER ADDRESSES
 1080  *
 1090  *      Note:  it is assumed and required that
 1100  *             ACCUM+2 already by zeroed!
 1110  *             Either that, or already set to the
 1120  *             highest byte of a 24-bit value.
 1130  *--------------------------------
 1140  CONVERT.TO.DECIMAL
 1150         STX ACCUM+1
 1160         STA ACCUM
 1170  .1     JSR DIVIDE.ACCUM.BY.TEN
 1180         LDA REMAINDER
 1190         ORA #"0"
 1200         STA BUFFER+1,Y
 1210         DEY
 1220         LDA ACCUM    CHECK IF QUOTIENT ZERO
 1230         ORA ACCUM+1
 1240         ORA ACCUM+2
 1250         BNE .1
 1260         RTS
 1270  *--------------------------------
 1280  DIVIDE.ACCUM.BY.TEN
 1290         LDX #24      24 BITS IN DIVIDEND
 1300         LDA #0       START WITH REM=0
 1310         STA REMAINDER
 1320  .1     JSR SHIFT.ACCUM.LEFT
 1330         ROL REMAINDER
 1340         SEC          REDUCE REMAINDER MOD 10
 1350         LDA REMAINDER
 1360         SBC #10
 1370         BCC .2       STILL < 10
 1380         STA REMAINDER
 1390         INC ACCUM    QUOTIENT BIT
 1400  .2     DEX          NEXT BIT
 1410         BNE .1
 1420         RTS
 1430  *--------------------------------
 1440  ACCUM      .EQ $BCAF,BCB0,BCB1
 1450  REMAINDER  .EQ $BCB2
 1460  BUFFER     .EQ $0200
 1470  *--------------------------------
 1480         .OR $AAD7
```

```
1490         .TA $900
1500  *--------------------------------
1510  SHIFT.ACCUM.LEFT
1520         ASL ACCUM
1530         ROL ACCUM+1
1540         ROL ACCUM+2
1550         RTS
1560  *--------------------------------
1570         .LIF
```

=================================================================
DOCUMENT :AAL-8504:DOS3.3:S.WINDOWS.txt
=================================================================

```
 1000  *SAVE S.WINDOWS
 1010  *---------------------------------
 1020  * MOVE WINDOW
 1030  * by Mike Ching, Kula Software
 1040  *    2118 Kula Street, Honolulu, HI 96817
 1050  *---------------------------------
 1060  WNDLFT   .EQ $20
 1070  WNDWDTH  .EQ $21
 1080  WNDTOP   .EQ $22
 1090  WNDBTM   .EQ $23
 1100  BASL     .EQ $28
 1110  BASH     .EQ $29
 1120  A1       .EQ $18,19     MEMORY SOURCE START
 1130  A2       .EQ $1A,1B     MEMORY SOURCE END
 1140  *---------------------------------
 1150  AMPERV   .EQ $3F5
 1160  *---------------------------------
 1170  GETBYT   .EQ $E6F8
 1180  COMBYTE  .EQ $E74C
 1190  BASCALC  .EQ $FBC1
 1200  HOME     .EQ $FC58
 1210  *---------------------------------
 1220           .OR $2F5
 1230           .TF B.WINDOWS
 1240  *---------------------------------
 1250  SETUP  LDA #MOVE.WINDOW     SET UP & VECTOR
 1260         STA AMPERV+1
 1270         LDA /MOVE.WINDOW
 1280         STA AMPERV+2
 1290         RTS
 1300  *---------------------------------
 1310  MOVE.WINDOW
 1320         JSR GETBYT   GET VALUES FROM APPLESOFT
 1330         STX TOP
 1340         STX LINE
 1350         JSR COMBYTE
 1360         STX BOTTOM
 1370         JSR COMBYTE
 1380         STX LEFT
 1390         JSR COMBYTE
 1400         STX RIGHT
 1410         SEC          WIDTH = RIGHT-LEFT
 1420         TXA
 1430         SBC LEFT
 1440         STA WIDTH
 1450         JSR COMBYTE  GET DIRECTION (1 OR 2)
 1460         DEX
 1470         STX TPAGE
 1480  *---------------------------------
```

```
1490  MOVE.LINE
1500         LDA LINE     BASL,H = BASCALC(LINE)
1510         JSR BASCALC
1520         CLC
1530         LDA BASH
1540         LDX TPAGE
1550         BEQ .1       ...SOURCE IS REAL SCREEN
1560         EOR #$0C     ...SOURCE IS SAVED SCREEN
1570  .1     STA A1+1     SOURCE HI BYTE
1580         EOR #$0C     FLIP TEXT PAGE
1590         STA A2+1     DESTINATION HI BYTE
1600         CLC          MEMSTART = BASL,H + LEFT
1610         LDA BASL
1620         ADC LEFT
1630         STA A1       SOURCE LO BYTE
1640         STA A2       DESTINATION LO BYTE
1650  *---MOVE THE LINE SEGMENT--------
1660         LDY WIDTH
1670  .2     LDA (A1),Y
1680         STA (A2),Y
1690         DEY
1700         BPL .2
1710  *---IF CLEARING, DRAW FRAME------
1720         LDY TPAGE
1730         BNE .4       ...NOT CLEAR, DO NOT DRAW FRAME
1740         LDA #$20     INVERSE BLANK
1750         STA (A1),Y   LEFT SIDE
1760         LDY WIDTH
1770         STA (A1),Y   RIGHT SIDE
1780         LDX LINE
1790         CPX TOP
1800         BEQ .3       ...TOP LINE
1810         CPX BOTTOM
1820         BNE .4       ...NEITHER TOP NOR BOTTOM
1830  .3     STA (A1),Y
1840         DEY
1850         BNE .3
1860  *---NEXT LINE-------------------
1870  .4     INC LINE     UNTIL LINE > BOTTOM
1880         LDA BOTTOM
1890         CMP LINE
1900         BCS MOVE.LINE     ANOTHER LINE TO MOVE
1910  *---IF CLEARING, SET WINDOW------
1920         LDA TPAGE
1930         BNE .5
1940         LDX LEFT
1950         INX
1960         STX WNDLFT
1970         LDX WIDTH
1980         DEX
1990         STX WNDWDTH
2000         LDX TOP
2010         INX
2020         STX WNDTOP
```

```
2030        LDX BOTTOM
2040        STX WNDBTM
2050        JSR HOME
2060 .5     RTS
2070 *--------------------------------
2080 TOP    .BS 1        PROGRAM STORAGE
2090 BOTTOM .BS 1
2100 LEFT   .BS 1
2110 RIGHT  .BS 1
2120 WIDTH  .BS 1
2130 LINE   .BS 1
2140 TPAGE  .BS 1
2150 *--------------------------------
```

======================================================================
DOCUMENT :AAL-8504:DOS3.3:WINDOW.DEMO.txt
======================================================================

-

≤ WINDOW DEMO PROGRAM, BASED ON PROGRAMS

≤ BY MIKE SEEDS, NIBBLE, JAN 1985q

-≤ ----------------------´

(P–12:≠,(104)—Pƒπ104,P:πP 256,0:∫Á(4)"RUN WINDOW DEMO"≈

```
2∫Á(4)"BRUN B.WINDOWS"„
```

<≤ ----------------------Î

```
dâ:ó
nÅI–1024¡2047«128:ÅJ–0¡119:πI»J,¤(1) 26»193:ÇJ,IS
x∫"         WINDOW DEMONSTRATION";:å…868:∫:å…868Ç
Ç¢22:å…958:∫:∫"     PRESS ANY KEY TO HALT";£
å≤  ------------------------‘
ñT–10:B–14:L–12:R–21:ØT,B,L,R,1:≤ OPEN WINDOW
†∫" TINY   WINDOW":∞1000:≤  DELAY AND CLOSE WINDOW&™≤------------- -
--------F»T–2:B–7:L–6:R–31:ØT,B,L,R,1Ç“¢T»3:ñ4:∫"NOTICE THE TEXT
IS":ñ4:∫"RESTORED CORRECTLY."å‹∞1000ßÊ≤--------------------¿T –
10:B–19:ØT,B,L,R,1¤ÅJ–1¡25:∫"  ";J,J J:ÇJ¸∫:∫" SCROLLING IS
AUTOMATIC"\"∞1000&J≤------------------------jêW–¤(1) 20»5:H–
¤(1) 10»5:T–¤(1) (24…H):B–T»H:L–¤(1) (40…W):R–
L»WùöØT,B,L,R,1:∫"ABCDEFGHIJKLMNOPQRSTUVWXYZ":∞1000¶§´150∂ËÅD –
1¡1500:Ç”ÚØT,B,L,R,2:≤CLOSE WINDOWË¸≠,(…16384)—
128ƒ±˛\°:π…16368,0:â:ó:Ä
```

```
==================================================================
DOCUMENT :AAL-8505:Articles:Auto.Manual.txt
==================================================================
```

AUTO/MANUAL Toggle Update for ................Robert F. O'Brien
S-C Macro Assembler Version 2.0 Dublin, Ireland

Here is a short routine (23 bytes) which makes use of the ESC-U command option to toggle the Auto-linenumbering mode on and off readily. The routine is relocatable so you can put it anywhere you have sufficient free space - just set the ESC-U vector to point to it, in this case :$C083 C083 D00C:4C 00 03 N C080.

When the cursor is waiting for input at the beginning of the command line, typing ESC-U will generate the command AUTO and then you have the option of entering a line number and/or RETURN. To cancel the AUTO mode just type ESC-U while the cursor is at the beginning of the line (just after the linenumber - 4 or 5 digit line numbers are catered for).

Extended AUTO command:

The second routine, starting at $317, is just 17 bytes long and extends the AUTO command so that you can specify the increment after the starting linenumber. For example, AUTO 3000,1 sets a starting line number of 3000 and an increment of 1. This code is also relocatable but you must patch the first instruction in the main AUTO command so that it uses the new code as a sub- routine. In this case it's :$C083 C083 D392:20 17 03 N C080.

The addresses specified for these new features are for the corrected version of the Assembler - i.e. serial nos. greater than 1251; see note in AAL March '85. Here is a table of what to expect at each of the addresses used, so you can find the equivalent spots in other copies of the assembler:

```
$D198 -- 20 xx D2      (JSR GNC)
         B0 17         (BCS to RTS)
         49 30         (EOR #$30)

$D392 -- 20 xx D1      (JSR GET.VALUE)
         CA            (DEX)
         30 0E         (BMI to SEC)

$D40B -- 41 55 54 CF   (.AT /AUTO/)
         xx D3         (.DA AUTO-1)

$DB9A -- 09 80         (ORA #$80)
         9D 00 02      (STA $0200,X)
         C9 A0         (CMP #$A0)
```

Note that with the Auto/Manual Toggle function installed you won't need the MANUAL command any more, so you have a spare command if you need it!

============================================================================
DOCUMENT :AAL-8505:Articles:Disasm.TechNote.txt
============================================================================

Adapting the Output Format of Rak-Ware DISASM.......Bob Kovacs

This technical note describes the format table used within DISASM 2.2e, which can be modified to adapt the output text file format to other assemblers. Even if you never plan to modify DISASM, or even if you don't own a copy of DISASM, you can learn a lot about the use of configuration tables by studying what follows.

The current version of the disassembler provides three different output formats to support the DOS ToolKit, S-C, and LISA assemblers. The format table contains various attributes which are unique to each assembler. The table begins at location $1331 and is $3F bytes long. Let's first examine the table and then determine how to adapt it to other assembler formats.

```
 Item      ToolKit      S-C          LISA
---------  -----------  -----------  -----------

comment    AA     *     AA     *     BB     ;
firstchr   00     none  89     ^I    00     none
tabchr1    A0     spc   89     ^I    A0     spc
tabchr2    A0     spc   A0     spc   A0     spc
opchr      C1     A     00     none  00     none
pgzchr     C5D1D5 EQU   AEC5D1 .EQ   C5D0DA EPZ
extchr     C5D1D5 EQU   AEC5D1 .EQ   C5D1D5 EQU
hexchr     C4C6C2 DFB   AEC8D3 .HS   C8C5D8 HEX
orgchr     CFD2C7 ORG   AECFD2 .OR   CFD2C7 ORG
prechr     AA0000 *     000000 none  C9CED3 INS
postchr    00     none  98     ^X    85     ^E
```

 comment:  the character used at the beginning of a line to signify a comment line.

firstchr:  the character ouput at the beginning of each line.

 tabchr1:  the character used to tab to the opcode field.

 tabchr2:  the character used to tab to the operand field.

   opchr:  operand for impled accumulator instructions (ASL, LSR, ROR, ROL).

  pgzchr:  directive for page zero declarations.

  extchr:  directive for absolute declarations.

  hexchr:  directive for data tables.

  orgchr:  directive for setting the program origin.

  prechr:  preamble sequence for initialization of the assembler.

 postchr:  postamble character for termination of the assembler's loading operation.

You will find that it is relatively simple to modify the format table for other assemblers.  First, determine which of the three existing formats is to be overwritten (just pick the one you think you'll need the least).  Then determine the format data which is appropriate to your assembler.  BLOAD DISASM, enter the monitor, and stuff the new values into the table.  Finally BSAVE DISASM,A$800,L$D00.

Or, if you have purchased the source code of DISASM 2.2e (or created your own using DISASM!), you can merely edit the table with your assembler and re-assemble the program.

You might also need or want to change some other paramteters, which are not in the format table:

Label Prefix:  located at $132E, the current value is C9DAD8 (the letters "IZX").  These letters are used to indicate internal, pagezero, and external labels in the generated text file.

Menu Table:  located at $1300, this table contains the names of the three assemblers listed in the first menu.  Each name is stored in ASCII, followed by a return ($0D) and a terminator ($00).

Label Name Separator:  A period ($AE) is output as the second character in every generated label name.  This can be changed to any other character by editing the LDA #$AE instruction at location $0EA4.

I would be interested in hearing from any of you who have already modified DISASM.  This kind of feedback can lead to new versions with even more powerful features.

```
==============================================================================
DOCUMENT :AAL-8505:Articles:Front.page.txt
==============================================================================
```

$1.80

Volume 5 -- Issue 8                          May, 1985

In This Issue...



S-C Macro Assembler for ProDOS

At long last, the news you've all been waiting for:  the ProDOS version of the S-C Macro Assembler is almost ready.  We have a working assembler in Beta testing, and it's doing just fine.  We need to spend another month or two shaking on it and developing documentation, so it will be just a little longer 'til we start shipping, but it's on the way!  Watch the front page of AAL for the announcement.

News from Don Lancaster

After nearly a year of delay at the publisher, Enhancing Your Apple II and //e, volume 2 is here!  This followup to his very popular collection of Apple tricks, gimmicks, and techniques contains still more high-quality information on how to get the most out of our favorite computer.  Here Don provides the tricks of microjustification and proportional spacing for Applewriter //e, an absolute "Old Monitor" style RESET for the //e, a software-only video synchronization technique for all Apple II's and //e's, and a just-for-fun guide to mapping and playing Castle Wolfenstein.

I've been saving the best for last:  Tearing Into Applewriter //e. Here is 86 pages of priceless data on the internal workings of the most popular Apple Word Processor, including how to capture source code and customize it to your own taste.  See our ad on page three for price and shipping.

As you will notice from his ad in this issue, Lancaster has been hard at work tearing into Appleworks, and has a set of disks available on that program.  We haven't seen those yet, but I'll bet they're more of the same great inside info we've come to expect from Don.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3

for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8505:Articles:Littles.ProDOS.txt
==============================================================================

Apple ProDOS: Advanced Features for Programmers, a Review.....
.....Bill Morgan

Gary Little, the prolific author of Inside the Apple //e and Inside the Apple //c, has yet another new book out. This one is called Apple ProDOS: Advanced Features for Programmers. In this volume Little covers just about all you need to know to write assembly language programs under ProDOS, from simply passing commands to BASIC.SYSTEM, through great detail on all the MLI calls, to writing your own interrupt handlers and device drivers.

Here's a quick summary of the book's contents:

1 - An Introduction to ProDOS -- Little starts out with the history of Apple's DOS's, a comparison of ProDOS and DOS 3.3, and a summary of important features of ProDOS.

2 - Files and File Management -- Here he covers the directory structures, file structures, disk formatting, and gives us a READ.BLOCK program.

3 - Loading and Installing ProDOS -- This chapter goes into the boot process, ProDOS' memory usage, and the Global Page.

4 - The Machine Language Interface -- This is the information on using the MLI, its error codes, and complete details of all MLI calls.

5 - System Programming Featuring BASIC.SYSTEM -- Here we have a discussion of system programs, the structure and commands of BASIC.SYSTEM, and assembly language programming under BASIC.SYSTEM.

6 - Interrupts -- In this chapter Little covers interrupts in general, ProDOS interrupt handling, and programming the Apple mouse.

7 - Disk Drivers -- Nearing the end, we go into identifying and handling foreign disk drivers, driver commands, the /RAM driver, and adding your own driver.

8 - ProDOS Clock Drivers -- And finally we find out about using the built-in clock support, adding a clock driver, and reading the date and time from Applesoft.

An important strength of this book is the wealth of examples. In the chapter on the Machine Language Interface there is an example of the correct use of EVERY MLI call. The BASIC.SYSTEM chapter includes an ONLINE command, to identify all disk volumes currently on line. The chapter on interrupts contains a couple of examples of mouse programming. The Disk driver section has a listing of a simple /RAM driver for main memory. And this is just a sample of the useful code provided in Little's new book. A companion disk containing all of the book's programs and more is available for $25.00 from the author.

I hear some of you asking: How does Apple ProDOS: Advanced Features (APAF) compare to Beneath Apple ProDOS (BAP)? Well, the two books complement each other quite nicely. With all its examples, treatment of interrupt handlers and device drivers, and overall clarity, I'd say that APAF is the better book on programming under ProDOS. BAP has useful examples as well, and better detail about the internals of diskette formatting and how ProDOS works, especially with its 120+ page supplement describing the code on a line-by-line basis. So if you're concerned with understanding the inner workings of the operating system, or with modifying its behavior, BAP is the book to have. Otherwise, get APAF for the best information on programming using ProDOS. Personally, I'm glad to have both books on the shelf here, along with Apple's ProDOS Technical Reference Manual.

Apple ProDOS: Advanced Features for Programmers, by Gary B. Little. Brady Communications Co., 1985. 266+iv pp., Reference Card. $17.95. Available from S-C Software for $17 + shipping.

```
=====================================================================
DOCUMENT :AAL-8505:Articles:My.Ad.txt
=====================================================================

S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners........................$20
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                  Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains        1980      -         -         -         1
 the source code from      1981      2         3         4         5
 three issues of AAL,      1982      6         7         8         9
 saving you lots of        1983     10        11        12        13
 typing and testing.       1984     14        15        16        17
                           1985     18

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36

Blank Diskettes (Verbatim)......................... package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers...........................  6 cents each

quikLoader EPROM System (SCRG)..............................($179) $170
PROmGRAMER (SCRG).........................................($149.50) $140
D MAnual Controller (SCRG)...................................($90)  $85
Switch-a-Slot (SCRG)........................................($190) $175
Extend-a-Slot (SCRG).........................................($35)  $32

"Apple ProDOS:  Advanced Features for programmers", Little..($17.95)  $17
"Inside the Apple //c", Little..............................($19.95)  $18
"Inside the Apple //e", Little..............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler......................($22.95)  $21
"Understanding the Apple II", Sather........................($22.95)  $21
"Understanding the Apple //e", Sather.......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18
```

```
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
"6502 Assembly Language Programming", Leventhal.............($18.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

=============================================================================
DOCUMENT :AAL-8505:Articles:New.Catalog.txt
=============================================================================

A New Catalog for DOS 3.3...................Robert F. O'Brien
Dublin, Ireland

In AAL March '85 Bob S-C presented re-writes of some loosely coded DOS sections to make space for patches - the Catalog Function Handler is another such loose bit of code, but rather than just free up some bytes I decided to add some useful features which Apple omitted and correct an annoying error at the same time. This new routine adds the following features to the CATALOG command:

1. Displays the free space remaining on the disk.

2. Allows you to terminate the Catalog during the normal pause after a screenful of files have been displayed by pressing the <ESC>-key (or other designated key).

3. Displays the correct number of sectors for each file in the Catalog for even the very large files - where the number of sectors exceeds 255 (which was the limit of the old PRINT.DECIMAL subroutine at $AE42 in DOS 3.3).

4. Optionally displays two filenames on each line of the Catalog - this is an 80-Column card option, also great for double-barrelled CATALOG printouts (for labels etc.).

In addition, the new Catalog retains the principal features of the old routine such as displaying the Volume number, the locked file indicator (*) and the file type abbreviation so that the user is not deprived of any essential information.

All the foregoing was achieved without using any additional DOS RAM space or zero-page locations other than that space already used by the Catalog Function Handler itself. Of course, something of the old routine had to be sacrificed in order to add the new features - it was necessary to omit the message "DISK VOLUME " from the beginning of the display. The 12-byte space where this message resided is now used to house a subroutine to check for locked files.

Even with all these enhancements, there are 17 free bytes left over! You could use some of them to print out an abbreviated form of the "DISK VOLUME " message, like "V=".

An additional constraint I saddled myself with in doing the re-write was that PRINT.DECIMAL (the DOS subroutine used to convert the hexadecimal numbers in locations $44,45 to decimal and print them) should retain its normal entry point ($AE42) so that the new code would be compatible with other programs which might use it.

For those who wish to get double-barrelled Catalog listings on an 80-column card or on a printer just change the "SEC" at line 2010 to "LSR". In other words, $AE12:4A will enable the wide printout, and $AE12:38 will put it back to normal.

To install the new patches just BLOAD the two binary files: NEW CATALOG PART 1, and NEW CATALOG PART 2. You can put the modified DOS onto any normal disk using Bob Perkins' technique (in AAL Aug 1982 p.24) without disturbing any other files present, or INIT a blank disk and the modified DOS will be incorporated on it. If you prefer to terminate long Catalog's with the <RETURN> key as you do for listings with the S-C Macro Assemblers just change byte $AE21:8D.

Also, if you are prepared to restrict yourself to 11 character file-names you can have a double-barrelled Catalog on the 40-Column screen by changing byte $ADF7:0B (POKE 44535,11), but I feel it would be of little value overall.

Now for a more detailed look at the program internals. Due to the requirement to save as many bytes as possible to squeeze in the desired features it was not possible to write the code in as straight-forward a manner as one would like. Even so, the routine was written with 17 byte to spare - after many re-writes to fit in all the features.

Lines 1020-1240 define various subroutines, variables, and data tables inside the rest of DOS.

Lines 1320-1360 use the same code as the original Catalog routine to initialize the File Manager and read the disk Volume Table Of Contents (VTOC).

In lines 1370-1410 we clear LINE.SKIP.FLAG which is used by SKIP.LINE subroutine to determine whether to tab to a second column or print a carriage return. Then we call PRINT.DECIMAL.YA to print the volume number. The volume number itself is passed in the A-register, and a zero high-byte in Y. Since we stripped out the code for printing "DISK VOLUME ", the volume number will be printed immediately to the right of the CATALOG command, on the same line. You will see "CATALOG 254  395", or the like, where the first number is the volume number and the second is the count of free sectors.

By making a special entry above the PRINT.DECIMAL subroutine which is used both here and at line 1830 below, we save several bytes. Of course we have already save a couple dozen bytes by not printing "DISK VOLUME ".

Calculation of the free disk space is made in lines 1420-1530. We make use of a new feature in the corrected PRINT.DECIMAL routine whereby $44 and $45 are reduced to 0 during the conversion - resulting in a saving of 4 bytes by not having to re-zero $45. (In the old routine only $44 was reduced to 0.)

In the VTOC 4 bytes are set aside for each track to indicate sector usage although only 2 are needed for a standard Apple disk. (The extra space allows up to 50 tracks and up to 32 sectors per track to be initialized.) A bit set=1 means that the corresponding sector on the track is available for use. If a bit is set=0 then the sector is already allocated. So it was simply a matter of counting every bit set from offset byte $38 (track 0) to Byte $C3 (for Trk $22) of the VTOC buffer to get a count of the free space. If you want to count all the way to the 50th track, in case the program is working with a hard disk like the Sider or Corvus, or a RANA 320K floppy, change lines 1430-1440:

```
       1430       LDX #$38
       1440       LDA VTOC.BUFFER,X
```

In line 1550 we have another departure from the original code - 2 bytes were saved by entering the tail end of the SKIP.LINE subroutine in order to set the number of lines to place on the screen before pausing during a Catalog. This has the added advantage that you can customize your Catalog more easily in that the line count can be adjusted by modifying a single byte ($AE25).

At lines 1570-1610 we start by clearing the Carry flag so that the first sector of the directory will be read (track $11, sector $0F). Also we set the index (X) to the first filename entry in the sector.

Lines 1620-1660 examine the track number of the Track/Sector list for the current filename entry. Should this number be 0 it indicates that we are at the end of the directory, at which point we would terminate the Catalog by exiting the File Manager routine by a jump to $B37F.

Fortunately, there was a JMP $B37F instruction within relative branching distance of the Catalog Function Handler. We could therefore dispense with the JMP to $B37F in the original code saving a further 3 bytes by branching to FM.EXIT at $AD86 instead. This is an address in the DELETE Function Handler ($AD2B-AD97) which precedes the Catalog routine in RAM. There are three ways we can terminate the Catalog, which all result in a branch to FM.EXIT: here at line 1600 when we find there are no more catalog sectors, at line 1650 when we find there are no more catalog entries, and at line 2090 when the ESC-key is typed during a screen-pause.

At line 1660 if the track number value is negative (bit 7 set) then we have found a deleted file. Deleted files don't show up on the Catalog, so we call on the subroutine at $B230 which sets the X-Register to the value of the entry point offset for the next entry in the sector, if any.

If on return from this subroutine the Carry flag bit is set (=1) then we have reached the end of the current catalog sector and we branch back to READ.SECTOR at line 1580 to read the next directory sector, if any. (Each directory sector accommodates 7 entries.)

At line 1680 we call the SKIP.LINE subroutine, which normally merely prints a carriage return. This routine was called from five different places in the original catalog code, so we have saved a dozen bytes by only calling it from this one place. (Putting it in-line would save four more!)

At line 1700 we call the new subroutine at the site of the DISK VOLUME message space to check for locked files and print the space or asterisk. This routine also leaves the file type code in the Y-register. This code could be placed in-line, rather than making it a subroutine, but then the final two lines could not be used as a short PRINT.SPACE subroutine.

Lines 1710-1790 convert the file type code to a file type character. The file type code is in bits 6-0, and is either zero (meaning type T), or a single bit. The hex values 40, 20, 10, 08, 04, 02, and 01 stand for file types B, A, R, S, B, A, and I. A string at $B4C8 holds "TIABSRAB", so we need to convert the bit position to an index value, and pick up the character out of that string. The ASL at line 1740 elminates the "lock/unlock" bit. The loop in line 1750-1770 shifts bits out until the value is zero, counting up in the Y-register. If the value was already zero, we exit immediately with Y=0, and type is "T". A type value of 1 gives an index of 1, up through $40 giving an index of 7.

By the way, types 40 and 20 are not Binary and Applesoft. They are hardly ever used, except in protection schemes. Types 04 and 02 are Binary and Applesoft.

The original catalog code had a significantly longer loop for converting the file type number to an index. You might want to compare the two.

The number of sectors in the file is picked up and converted in lines 1800-1830 and the decimal value is printed, surrounded by spaces. Lines 1840-1900 print out the file name.

Lines 1920-1950 advance to the next filename entry, and branch either to process it or to read in another catalog sector.

Lines 1970-2130 usually print a carriage return. If you have changed line 2010 to "LSR", to get double column catalogs, the least significant bit of LINE.SKIP.FLAG will determine whether to print a carriage return or not. When line 2010 is "SEC" we will always get a carriage return. If a carriage return is printed, we also count the line. When the line count is complete, we pause and wait for a keystroke. If that key is an ESC-key, the catalog will terminate. If not, the line count is re-initialized and we go back for more file names.

Line 2150 simply reserves 17 bytes, shoving the PRINT.DECIMAL routine down so that it still starts at $AE42 like it used to. These 17 bytes could be used for other code or data, whatever you like.

Lines 2160-2230 store a value to be converted and printed, print a blank, and then fall into the PRINT.DECIMAL subroutine.

The new corrected PRINT.DECIMAL subroutine is actually a little shorter than th buggy original. It left room for a JMP PRINT.SPACE at the end, which saved calling PRINT.SPACE from several other places. It also left room for the LINE.SKIP.FLAG variable.

The PRINT.DECIMAL subroutine (lines 2240-2490) effectively divides the number in $44,45 by subtracting in turn the values 100, 10 and 1 from it - a 16-bit subtraction. The count of the number of subtractions and the low order byte remainder are temporarily stored on the stack to conserve memory usage. We start with 100 and keep subtracting it and incrementing the subtraction-counter until we get borrow, at which point we print the counter value.

Now $44,45 will contain the remainder and so we continue using 10 and then 1 until three decimal digits are printed. This subroutine can accurately convert numbers having values up to 999 decimal.

CHALLENGE. Even though we have already squeezed out 17 bytes, while adding new features, we did lose the "DISK VOLUME " message. Can someone out there squeeze enough more out, without losing any features, to slip the message back in?

CAVEAT. If you decide to put this new CATALOG program on your disks, please be careful. There are some programs which temporarily patch the catalog routine themselves. In particular, ES-CAPE and other commercial programs patch the SKIP.LINES subroutine so that the pause is eliminated. Since SKIP.LINES has been moved and is different, no telling what might happen.

==============================================================================
DOCUMENT :AAL-8505:Articles:Probs32BitValue.txt
==============================================================================

32-bit Values in Version 2.0 -- A Mixed Blessing......Bob S-C

In previous versions of the S-C assemblers, expressions were evaluated in 16 bits, and symbol values were kept in the table in 16-bit form. Version 2.0 works with 32-bit expressions and symbol values. We added this feature for your benefit, but it may sometimes be a mixed blessing.

For example, Bob Bernard had a problem with a program which assembled perfectly under Version 1.0, but gave countless BAD ADDRESS errors in version 2.0. We traced the problem to his origin statement, which was ".OR -31488". In older versions, -31488 is the same as $8500, but in version 2.0 it is $FFFF8500. The following code will not assemble:

```
       .OR -31488
  SSS  JMP SSS
```

Why? Because the value of SSS is also $FFFF8500, and it will not fit in a JMP instruction. In 65816 mode, using a JML instruction, it would be legal.

Two ways to fix come to mind. You normally work in hexadecimal when you are in assembly language, rather than decimal. Therefore, change the origin statement to ".OR $8500". Or, if you really want to use decimal, write ".OR 65536-31488".

Another owner of version 2.0 had a problem with a program that used many macros, and lots of private labels. Private labels are the ones used inside macro definitions, which are written with a colon and a one or two digit number. The private label table normally begins at $FFF and grows downward toward $800. His program assembled with no problems before, but under version 2.0 it got a MEM FULL error. Reason, again, the 32-bit symbol values. Each entry in the private label table now takes two more bytes, so he ran out of space sooner. His solution was to move the beginning of the label table higher.

```
==================================================================
DOCUMENT :AAL-8505:Articles:ProDOS.Date.txt
==================================================================
```

DATE Command for ProDOS.............................Bill Morgan

One of the nice new features in ProDOS is the way the diskette catalog shows the date of creation and last modification for each file, IF you have a clock/calendar card installed in your Apple. Well I don't have such a card in either of the Apples I use regularly, at work or at home. And no //c has a clock! (Yet, at least. I'll bet someone will come up with a way...)

Anyway, I got tired of always seeing <NO DATE> and started figuring out how to set a date without a clock to do it for me. A look at Beneath Apple ProDOS informed me that the current date is transformed into the format YYYYYYYMMMMDDDDD and stored (in the usual 6502 low byte/high byte sequence) at $BF90-BF91 in the ProDOS Global Pages (the fixed locations of all of the accessible system variables). The first thing I did was manually convert the current date into that format and poke it in from the Monitor. That went like this:

```
                         $BF90      $BF91
May =  $5 = 0101       MMM DDDDD  YYYYYYY M
 10 =  $A = 01010      101 01010  1010101 0
'85 = $55 = 1010101       $AA        $AA
```

So, the values to poke into $BF90-91 were $AA and $AA. What better time than a four-A day to start such a project!

That experiment worked just fine: the next file I saved on the disk showed creation and modification dates of 10-MAY-85, just as I had hoped. With that success under my belt the next step had to be to come up with a program to read and/or set those date bytes. And, while I'm at it, why not take advantage of ProDOS' built-in hooks for installing new commands and add a DATE command to the operating system?

How do I go about adding a command? The ProDOS Technical Reference Manual is pretty sketchy on the subject, but two other books, Beneath Apple ProDOS and the new Apple ProDOS: Advanced Features for Programmers, have good descriptions and examples of the procedure. If you're going to do much assembly language programming under ProDOS you should have one or both of those books.

When ProDOS fails to recognize a command it does a JSR EXTRNCMD ($BE06) to find out if an external command processor will claim this one. What I have to do is install the address of DATE in $BE07-08, after moving the address that was already there into a JMP instruction. This way, if DATE doesn't recognize the command it can pass it along to any other processor that might have been there before.

Processing of an external command is normally divided into two phases, a parser and a handler. The parser section will scan the command name at the beginning of the line. If the command is not recognized, the parser should set the carry bit and JMP on to the address found in EXTRNCMD to see if another external processor will claim it.

If the command is recognized, the parser can set certain bits in PBITS ($BE54-55) to signify which parameters are permitted or required on the command line, and store the address of the handler in EXTRNADDR ($BE50-51). After storing the command length minus one in XLEN ($BE52) and a zero in XCNUM ($BE53), to signify that an external processor did claim the command, the parser then returns control to ProDOS to scan the rest of the line. If the line was syntactically correct, ProDOS will return the values of the parameters in a set of standard locations ($BE58-6F) and pass control back to the handler address specified.

Since DATE is a simple processor that uses a nonstandard parameter, I just set PBITS to zero, to indicate no parsing necessary, and store the address of an RTS instruction in EXTRNADDR. I then proceed to do all my processing before returning to ProDOS.

There is one additional wrinkle to using an external command with ProDOS: where do I put my code so ProDOS, Applesoft, and others don't stomp all over it? In the interest of simplicity I have ignored that problem here. The best procedure, as shown in the books mentioned above, is to call ProDOS to assign me a buffer and then relocate my code into that buffer. The examples in the books provide details of this process.

Now, let's take a look at the code:

Lines 1310-1400 install DATE by moving the current External Command address to my exit JMP instruction and storing DATE's address in the vector.

Lines 1440-1540 check the input buffer to see if this is a DATE command. If not we branch on down to that JMP instruction where we earlier put the address found in the External Command vector. This passes control either on to the next external command in the chain, or back to ProDOS for a SYNTAX ERROR.

If the command matched we go on to lines 1560-1650 to do the necessary housekeeping. This involves storing the command length-1 in the Global Page, setting a couple of flags to tell ProDOS not to parse the rest of the command line, and that an external command has taken over. Then we supply a handler address for the second half of ProDOS' processing, which in this case is just an RTS instruction. Finally we reach lines 1670-1690, where we check to see if the character following DATE is a Carriage Return. If so we branch forward to RETURN.DATE to display the existing date.

If there is more than just DATE on the command line, we must want to set a new date, so we fall into SET.DATE at line 1710. This routine makes heavy use of ACCUMULATE.DIGITS at line 2400, so we'll examine that code first. The first step is to zero the byte where we'll be accumulating the value typed in. Then we scan forward in the input buffer, looking for a nonblank character. When we find one we first check to see if it is a slash, which marks the end of a number, or a Carriage Return, which marks the end of the line. If it was either of those we exit, setting the Carry bit to indicate which one we found.

If the character found was not a delimiter we next check to see if it is a number. If not, we have a SYNTAX ERROR. When we do get a number, we strip off the high bits to convert the ASCII code to a binary value, and save that value. We then multiply the previous value in ACCUM by 10 and add in the new value. Then it's back to line 2440 to get another character. Lines 2710-2730 load the A-register with the value found and branch to the error exit if that value was zero.

Now, back to SET.DATE. That routine begins at line 1720 with a DEY to get ready for the INY at the beginning of ACCUMULATE.DIGITS. We then get the month, check for a legal value, and store it. Next we get the day, save the status, and check and save that value. Then it's time to check the status to see if the day was followed by a slash, or by a Carriage Return. If it was a slash then a year was specified, so we go get that value. If it was a Return no year was present, so we use 1985. (I guess that means we'll have to reassemble or patch this program every year. I think I can handle that.)

The last step in SET.DATE is to fold the year, month, and day together as described above and store the results in the Global Page. The comments in the listing illustrate how the bits are shuffled around to the correct format. After setting the date we fall into RETURN.DATE to display the result.

RETURN.DATE, at lines 2080-2290, is quite straightforward. It just gets the bytes from the Global Page, unfolds them, and calls DEC.OUT to translate them to decimal numbers and display those numbers. Again, the comments illustrate the bit manipulations involved in the unfolding process.

The final section of code is DEC.OUT, at lines 2750-2910. In lines 2760-2810 we use the Y-register to count how many times we can subtract 10 from the number passed in the A-register. Then lines 2830-2910 restore and save the A-register, make sure the tens count is non-zero, convert it to a character and print it. We then recover the units value and print that out.

```
==============================================================================
DOCUMENT :AAL-8505:Articles:Windows80Column.txt
==============================================================================
```

80-Column Window Utility for //e and //c............Bill Reed
New Orleans, LA

I throughly enjoyed "Fast Text Windows" by Michael Ching. However, I prefer not to use the area at $800-BFF as a text buffer; I much prefer to use the first bank of the language card, which is not normally used by Applesoft programs running under DOS 3.3.

I modified Mike's program by changing the immediate values in lines 1560 and 1580 from #$0C to #$D4 and adding lines 1644, 1646 and 1905. The first two lines enable the bank of RAM to be read or written to. The last re-enables the Applesoft ROMS.

```
     1644       LDA $C08B
     1646       LDA $C08B
     1905       LDA $C082
```

I further modified the program to function in 80 columns on a //e or //c. The big problem was to mimic the text card, which uses bank switching to store adjacent characters in the same address, but different locations (main RAM and aux RAM). This was solved by using one buffer for the "even" characters and another for the "odd".

Additional code was required to determine the even/odd condition, so I (being lazy) removed the border portion of the program to conserve room. The border routines could certainly be retained if part of the program was also moved to bank one of the language card area. (Be careful if you try this, because you must avoid calling the monitor or Applesoft ROMs when the ROMs are switched off. You can possibly get away with calling the monitor with the ROMs switched off, but only if you first make a copy of the monitor in the F800-FFFF area of RAM.)

I moved the data storage to the zero page, mostly because it was available and slightly faster.

```
=============================================================================
DOCUMENT :AAL-8505:DOS3.3:S.AUTO.MAN.txt
=============================================================================

 1000  *SAVE S.AUTO/MAN
 1010  *---------------------------------
 1020         .OR $300
 1030  *      .TF AUTO/MAN TOGGLE
 1040  *---------------------------------
 1050  INCREMENT    .EQ $5A,5B
 1060  SYM.VALUE    .EQ $B8,B9
 1070  AUTO.FLAG    .EQ $E3
 1080
 1090  WARM.START   .EQ $D028
 1100  GET.VALUE    .EQ $D198
 1110  AUTO.CMD     .EQ $D40B
 1120  INSTALL.CHAR .EQ $DB9A
 1130  *---------------------------------
 1140  AUTO.MAN.CODE
 1150         TXA               check cursor posn.
 1160         BEQ .1            OK to output cmd.
 1170         CPX #7            line start?
 1180         BGE .2            ignore ESC-U.
 1190         LSR AUTO.FLAG     cancel auto-mode.
 1200         JMP WARM.START
 1210
 1220  .1     LDA AUTO.CMD,X    output cmd. name
 1230         JSR INSTALL.CHAR  put in buffer+scrn.
 1240         CPX #4            4 chars. output?
 1250         BNE .1            no.
 1260  .2     RTS               exit ESC-U routine
 1270  *------------------------------
 1280  *  Point start of AUTO cmd. handler
 1290  *  to here for extended function.
 1300  *---------------------------------
 1310  NEW.AUTO.EXT
 1320         JSR GET.VALUE     get linenum if any.
 1330         JSR GET.VALUE     get inc. if any.
 1340         CPX #3            increment?
 1350         BLT .1            no
 1360         DEX               adjust for inc.
 1370         DEX                   do.
 1380         LDA SYM.VALUE     set inc. low byte
 1390         STA INCREMENT
 1400  *  (following 2 lines only needed
 1410  *   if you use increments of 255+!)
 1420  *      LDA SYM.VALUE+1   set inc. high byte
 1430  *      STA INCREMENT+1
 1440  .1     RTS               finish AUTO cmd.
```

```
==============================================================================
DOCUMENT :AAL-8505:DOS3.3:S.DATE.txt
==============================================================================

 1000  *SAVE S.DATE
 1010  *--------------------------------
 1020  *
 1030  *      Program to read or set the
 1040  *      date bytes in the Global Page
 1050  *
 1060  *            by Bill Morgan
 1070  *
 1080  *--------------------------------
 1090  POINTER    .EQ $40,41
 1100  ACCUM      .EQ $42
 1110  MONTH      .EQ $43
 1120  DAY        .EQ $44
 1130  TEMP       .EQ $45
 1140
 1150  WBUF       .EQ $200
 1160
 1170  EXTRNCMD   .EQ $BE07
 1180  EXTRNADDR  .EQ $BE50,51
 1190  XLEN       .EQ $BE52
 1200  XCNUM      .EQ $BE53
 1210  PBITS      .EQ $BE54
 1220  GP.DATE    .EQ $BF90
 1230
 1240  PRAX       .EQ $F941
 1250  CROUT      .EQ $FD8E
 1260  COUT       .EQ $FDED
 1270  *--------------------------------
 1280         .OR $803
 1290  *      .TF B.DATE
 1300  *--------------------------------
 1310  INSTALL
 1320         LDA EXTRNCMD+1    exit to old
 1330         STA EXIT+2        user command
 1340         LDA EXTRNCMD
 1350         STA EXIT+1
 1360         LDA /DATE         become new
 1370         STA EXTRNCMD+1    user command
 1380         LDA #DATE
 1390         STA EXTRNCMD
 1400         RTS
 1410  *--------------------------------
 1420  COMMAND .AS /DATE/
 1430  *--------------------------------
 1440  DATE   LDY #0
 1450         STY POINTER       point to input buffer
 1460         LDA /WBUF
 1470         STA POINTER+1
 1480  .1     LDA (POINTER),Y   scan command
```

```
1490        AND #%01111111
1500        CMP COMMAND,Y
1510        BNE ERR.BRIDGE    not mine
1520        INY
1530        CPY #4
1540        BCC .1
1550 *--- ProDOS bookkeeping ---------
1560        DEY
1570        STY XLEN          command length - 1
1580        INY
1590        LDA #0
1600        STA PBITS         don't parse parms
1610        STA XCNUM         external command
1620        LDA #RTS1
1630        STA EXTRNADDR     no execution after
1640        LDA /RTS1         command parsing
1650        STA EXTRNADDR+1
1660 *--- set or display date? -------
1670        LDA (POINTER),Y
1680        CMP #$8D          DATE only?
1690        BEQ RETURN.DATE   yes, return old date
1700 *--------------------------------
1710 SET.DATE
1720        DEY
1730        JSR ACCUMULATE.DIGITS  get month
1740        CMP #13
1750        BCS ERROR         >12 no good
1760        STA MONTH
1770        JSR ACCUMULATE.DIGITS  get day
1780        PHP               save status
1790        CMP #32
1800        BCC GO.ON         <=31 ok
1810
1820        PLP
1830 ERR.BRIDGE
1840        BNE ERROR      ...always
1850
1860 GO.ON  STA DAY
1870        PLP               recover status
1880        BCC .1            .CC. if "/"
1890        LDA #85           year defaults to '85
1900        BNE .2         ...always
1910 .1     JSR ACCUMULATE.DIGITS  get year
1920        CMP #100
1930        BCS ERROR         >99 no good
1940 .2     PHA               save year
1950        LDA MONTH         X 0000MMMM
1960        LSR               M 00000MMM
1970        ROR               M M00000MM
1980        ROR               M MM00000M
1990        ROR               M MMM00000
2000        STA MONTH
2010        PLA               M 0YYYYYYY
2020        ROL               0 YYYYYYYM
```

```
2030        STA GP.DATE+1
2040        LDA MONTH          MMM00000
2050        ORA DAY            MMMDDDDD
2060        STA GP.DATE
2070 *--------------------------------
2080 RETURN.DATE
2090        JSR CROUT
2100        LDA GP.DATE+1     X YYYYYYYM
2110        LSR               M 0YYYYYYY
2120        PHA
2130        LDA GP.DATE       M MMMDDDDD
2140        PHA
2150        ROR               X MMMMDDDD
2160        LSR               X 0MMMMDDD
2170        LSR               X 00MMMMDD
2180        LSR               X 000MMMMD
2190        LSR               X 0000MMMM
2200        JSR DEC.OUT       display month
2210        LDA #"/"          /
2220        JSR COUT
2230        PLA               X MMMDDDDD
2240        AND #%00011111    X 000DDDDD
2250        JSR DEC.OUT       display day
2260        LDA #"/"          /
2270        JSR COUT
2280        PLA               X 0YYYYYYY
2290        JSR DEC.OUT       display year
2300 *--------------------------------
2310 GOOD.EXIT
2320        CLC               signal no error
2330 RTS1   RTS
2340 *--------------------------------
2350 ERROR1 PLA               clean up
2360        PLA               return addresses
2370 ERROR  SEC               signal error
2380 EXIT   JMP RTS1          INSTALL makes address
2390 *--------------------------------
2400 ACCUMULATE.DIGITS
2410        LDA #0
2420        STA ACCUM         zero accumulator
2430
2440 .1     INY               next character
2450        LDA (POINTER),Y
2460        AND #%01111111    hi-bit off
2470        CMP #' '          space?
2480        BEQ .1            back for another
2490        CMP #'/'          slash?
2500        BEQ .2            yes, exit .CC.
2510        CMP #$0D          <CR>?
2520        BEQ .3            yes, exit .CS.
2530        CMP #'0'          too small?
2540        BCC ERROR1        not digit
2550        CMP #'9'+1        too big?
2560        BCS ERROR1        not digit
```

```
2570
2580          AND #%00001111    isolate value
2590          STA TEMP          stash it
2600          LDA ACCUM
2610          ASL               X 2
2620          ASL               X 4
2630          ADC ACCUM         X 5
2640          ASL               X 10
2650          ADC TEMP          add new digit
2660          BCS ERROR1        too big
2670          STA ACCUM
2680          BCC .1         ...always
2690
2700   .2     CLC               .CC. if /
2710   .3     LDA ACCUM         return value
2720          BEQ ERROR1        0 no good
2730          RTS
2740   *--------------------------------
2750   DEC.OUT
2760          LDY #0            zero counter
2770          SEC               get ready
2780   .1     SBC #10           subtract 10
2790          BCC .2            borrow?
2800          INY               count a 10
2810          BPL .1         ...always
2820
2830   .2     ADC #10           restore borrow
2840          PHA               save units
2850          TYA               print 10's count
2860          BEQ .3            no leading zero
2870          ORA #$B0          make character
2880          JSR COUT          print it
2890   .3     PLA               recover units
2900          ORA #$B0          make character
2910          JMP COUT          return through COUT
```

==============================================================================
DOCUMENT :AAL-8505:DOS3.3:S.NEW.CATALOG.txt
==============================================================================

```
*SAVE S.NEW CATALOG
*--------------------------------
DOS.ARITH.REG          .EQ $44,45
*--------------------------------
ADV.NEXT.DIR.ENTRY     .EQ $B230
DOS.INIT.FM            .EQ $ABDC
EXIT.FM                .EQ $AD86
MON.COUT               .EQ $FDED
MON.CROUT              .EQ $FD8E
MON.RDKEY              .EQ $FD0C
READ.DIRECTORY.SECTOR  .EQ $B011
READ.VTOC              .EQ $AFF7
*--------------------------------
DEC.CONVERSION.TABLE   .EQ $B3A4
FILE.TYPE.NAME.TABLE   .EQ $B3A7
*--------------------------------
CATALOG.LINE.COUNT     .EQ $B39D
DIRECTORY.ENTRY        .EQ $B4C6
DIRECTORY.INDEX        .EQ $B39C
DISK.VOL.NUMBER        .EQ $B7F6
FILE.NAME              .EQ $B4C9
FILE.SIZE              .EQ $B4E7
FILE.TYPE              .EQ $B4C8
FM.VOL.NUMBER          .EQ $B5F9
VTOC.BUFFER            .EQ $B3BB
*--------------------------------
*     New Catalog for DOS 3.3
*     by  Robert F.O'Brien
*--------------------------------
       .OR $AD98
       .TF NEW CATALOG PART 1
*--------------------------------
CATALOG
       JSR DOS.INIT.FM    Init file manager.
       LDA #$FF           Set Volume = 0
       STA FM.VOL.NUMBER    (matches any volume)
       JSR READ.VTOC      Load in VTOC into buffer.
*---Print Volume Number----------
       LDY #0             High byte = 0
       STY LINE.SKIP.FLAG     (signal to skip)
       LDA DISK.VOL.NUMBER    low byte
       JSR PRINT.DECIMAL.YA
*---Calculate Free Space---------
       LDX #$74           Trk 0 VTOC offset
.1     LDA VTOC.BUFFER+$38-$74,X      Bit Map Byte
.2     BPL .3             This sector in use
       INC DOS.ARITH.REG    Count a free sector.
       BNE .3
       INC DOS.ARITH.REG+1
```

```
.3     ASL                 Check next bit
       BNE .2              Still more in this byte
.4     INX                 Next byte of bit map
       BNE .1              ...still more
       JSR PRINT.DECIMAL    print number free
*---Start Line count-------------
       JSR SET.LINE.COUNT        lines to print.
*---Start reading directory------
       CLC                 Get first sector.
READ.SECTOR
       JSR READ.DIRECTORY.SECTOR
       BCS EXIT.FM         No more sectors
       LDX #0              Index to 1st file in sector
SET.ENTRY.INDEX
       STX DIRECTORY.INDEX   Set entry offset
       LDA DIRECTORY.ENTRY,X See if valid filename
EXIT   BEQ EXIT.FM         ...end of directory
       BMI NEXT.ENTRY      ...ignore deleted file.
*---Start next file display------
       JSR SKIP.LINE       Next line or Tab
*---Locked or Unlocked-----------
       JSR LOCKED.FILE.CHECK     "*" if locked file.
*---File Type--------------------
       TYA                 Get file type byte
       LDY #-1             Index to type table
       ASL                 Ignore Bit 7
.1     INY                 Next file type code
       LSR                 Check bit of type byte
       BNE .1              ...not yet
.2     LDA FILE.TYPE.NAME.TABLE,Y Get file type
       JSR MON.COUT        ...and print it
*---File Size--------------------
       LDY FILE.SIZE+1,X         high order byte
       LDA FILE.SIZE,X           low order byte
       JSR PRINT.DECIMAL.YA      print total sect.
*---File Name--------------------
       LDY #30
.3     LDA FILE.NAME,X           char. no. in Y.
       JSR MON.COUT          print file name.
       INX                       next char.
       DEY
       BNE .3                    not done yet!
*---Next File in Directory-------
NEXT.ENTRY
       JSR ADV.NEXT.DIR.ENTRY  Set X-Reg for next file
       BCC SET.ENTRY.INDEX     more in sector
       BCS READ.SECTOR         get next sector
*--------------------------------
SKIP.LINE
       JSR PRINT.SPACE   Separate 2nd line entry
       INC LINE.SKIP.FLAG  Toggle lsbit
       LDA LINE.SKIP.FLAG  Check Odd or Even
       SEC        <<<Change to "LSR" for double
*                    column CATALOG >>>
```

```
2030         BCC RETURN
2040         JSR MON.CROUT     Start a new line
2050         DEC CATALOG.LINE.COUNT    continue countdown
2060         BNE RETURN                not full screen yet
2070         JSR MON.RDKEY     Pause for keypress!
2080         CMP #$9B          Is it  ESC-key?
2090         BEQ EXIT          ...yes, exit file manager
2100  SET.LINE.COUNT
2110         LDA #21           lines per screenful
2120         STA CATALOG.LINE.COUNT
2130  RETURN RTS               Continue catalog
2140  *--------------------------------
2150         .BS 17            17 FREE BYTES!
2160  *--------------------------------
2170  *   Print (YA) with leading and
2180  *      trailing blanks.
2190  *--------------------------------
2200  PRINT.DECIMAL.YA
2210         STY DOS.ARITH.REG+1
2220         STA DOS.ARITH.REG
2230         JSR PRINT.SPACE
2240  *--------------------------------
2250  *   Print ($44,45) with trailing blank
2260  *--------------------------------
2270  PRINT.DECIMAL
2280         LDY #2            Set for 3 divisors
2290  .1     LDA #$B0          ASCII zero
2300  .2     PHA               save digit on stack
2310         SEC               Subtract 100, 10, or 1
2320         LDA DOS.ARITH.REG    from remainder
2330         SBC DEC.CONVERSION.TABLE,Y
2340         PHA               save remainder on stack
2350         LDA DOS.ARITH.REG+1
2360         SBC #0            (divisor high byte = 0)
2370         BCC .3            ...far enough
2380         STA DOS.ARITH.REG+1   Update remainder
2390         PLA
2400         STA DOS.ARITH.REG
2410         PLA               get current digit
2420         ADC #0            and count the subtraction
2430         BNE .2            ...continue subtracting
2440  .3     PLA               Discard stacked remainder byte
2450         PLA               Get quotient digit
2460         JSR MON.COUT      and print it!
2470         DEY               Next divisor
2480         BPL .1            ...not finished yet
2490         JMP PRINT.SPACE   Trailing space
2500  *--------------------------------
2510  LINE.SKIP.FLAG .DA #0    LEAST SIGNIFICANT BIT IS FLAG
2520  *--------------------------------
2530         .OR $B3AF
2540         .TF NEW CATALOG PART 2
2550  *--------------------------------
2560  *   OVERLAYS "DISK VOLUME " MESSAGE
```

```
2570  *--------------------------------
2580  LOCKED.FILE.CHECK
2590         LDA #"*"
2600         LDY FILE.TYPE,X   file type code.
2610         BMI CAT.COUT      ...the file is locked
2620  PRINT.SPACE
2630         LDA #" "
2640  CAT.COUT
2650         JMP MON.COUT
2660  *--------------------------------
2670         .LIF
```

```
=============================================================================
DOCUMENT :AAL-8505:DOS3.3:S.WINDOWS.80.txt
=============================================================================

 1000  ; SAVES.WINDOWS.80
 1010  *-AAL APRIL 85 P 16-------------
 1020        .OR $2F5
 1030        .TF B.WINDOWS.80
 1040  *-------------------------------
 1050  TOP        .EQ $00  $0-$6 DATA
 1060  BOTTOM     .EQ $01
 1070  LEFT       .EQ $02
 1080  RIGHT      .EQ $03
 1090  WIDTH      .EQ $04
 1100  LINE       .EQ $05
 1110  DIREC      .EQ $06
 1120  *
 1130  B1         .EQ $18,19   TEXT PNTR
 1140  B2         .EQ $1A,1B   BUFR PNTR
 1150  B3         .EQ $1C,1D   BUFR PNTR
 1160  WNDLFT     .EQ $20
 1170  WNDWDTH    .EQ $21
 1180  WNDTOP     .EQ $22
 1190  WNDBTM     .EQ $23
 1200  BASL       .EQ $28
 1210  BASH       .EQ $29
 1220  *-------------------------------
 1230  AMPERV     .EQ $3F5
 1240  PAG2OFF    .EQ $C054  READ MRBRD
 1250  PAG2ONN    .EQ $C055  READ AUXBRD
 1260  LCROM      .EQ $C082
 1270  LCRAM1     .EQ $C08B
 1280  GETBYTE    .EQ $E6F8
 1290  COMBYTE    .EQ $E74C
 1300  BASCALC    .EQ $FBC1
 1310  HOME       .EQ $FC58
 1320  *-------------------------------
 1330  SETUP
 1340        LDA #MOVE.WINDOW
 1350        STA AMPERV+1
 1360        LDA /MOVE.WINDOW
 1370        STA AMPERV+2
 1380        RTS
 1390  *-------------------------------
 1400  MOVE.WINDOW
 1410        JSR GETBYTE
 1420        STX TOP
 1430        STX LINE
 1440        JSR COMBYTE
 1450        STX BOTTOM
 1460        JSR COMBYTE
 1470        STX LEFT
 1480        JSR COMBYTE
```

```
1490        STX RIGHT
1500        INX
1510        SEC
1520        TXA
1530        SBC LEFT
1540        STA WIDTH
1550        JSR COMBYTE
1560        DEX
1570        STX DIREC
1580  *--------------------------------
1590  *--------------------------------
1600  MOVE.LINE
1610        LDA LINE
1620        JSR BASCALC
1630        LDA BASH
1640        STA B1+1
1650        EOR #$D4
1660        STA B2+1
1670        CLC
1680        ADC #$04       2ND BUFR
1690        STA B3+1
1700        LDA BASL
1710        STA B1
1720        STA B2
1730        STA B3
1740        LDA LCRAM1     ENABLE LANG
1750        LDA LCRAM1     CARD R/W
1760  *--MOVE THE LINE SEGMENT--------
1770        LDA RIGHT
1780        LSR            A/2 + EVN/ODD
1790        TAY            TXT SCRN PNTR
1800        LDX DIREC
1810        BNE .3
1820  *--MOVE IT UP-------------------
1830        LDX WIDTH      DOWN COUNTER
1840        BCC .2
1850  .1    LDA (B1),Y     DO ODD COLS
1860        STA (B2),Y
1870        DEX
1880        BMI .6
1890  .2    LDA PAG2ONN    DO EVN COLS
1900        LDA (B1),Y
1910        STA (B3),Y
1920        LDA PAG2OFF
1930        DEY
1940        DEX
1950        BPL .1
1960        BMI .6
1970  *--MOVE IT DOWN-----------------
1980  .3    LDX WIDTH
1990        BCC .5
2000  .4    LDA (B2),Y     DO ODD COLS
2010        STA (B1),Y
2020        DEX
```

```
2030        BMI .6
2040  .5    LDA PAG2ONN   DO EVN COLS
2050        LDA (B3),Y
2060        STA (B1),Y
2070        LDA PAG2OFF
2080        DEY
2090        DEX
2100        BPL .4
2110  *--NEXT LINE--------------------
2120  .6    INC LINE
2130        LDA LCROM     RESTORE ROM
2140        LDA BOTTOM
2150        CMP LINE
2160        BCS MOVE.LINE
2170  *--IF CLEARING, SET WINDOW------
2180        LDA DIREC
2190        BNE .7
2200        LDX LEFT
2210        STX WNDLFT
2220        LDX WIDTH
2230        DEX
2240        STX WNDWDTH
2250        LDX TOP
2260        INX
2270        STX WNDTOP
2280        LDX BOTTOM
2290        STX WNDBTM
2300        JSR HOME
2310  .7    RTS
2320  *-------------------------------
2330  ZZEND .EQ *
```

=====================================================================
DOCUMENT :AAL-8506:Articles:Ads.txt
=====================================================================

==============================================================================
DOCUMENT :AAL-8506:Articles:Alliance.Note.txt
==============================================================================

Note About Alliance Computers

Back in January Alliance Computers advertised 65802's for $50.00, but couldn't fill the orders because the chips didn't exist yet. Some of their formerly unhappy customers tell me that their orders have now arrived, so Alliance is taking care of their customers. You can reach Alliance Computers at P.O. Box 408, Corona, NY 11368.

```
=============================================================================
DOCUMENT :AAL-8506:Articles:AppleVisions.txt
=============================================================================
```

AppleVisions, a Glimpse

Here is a very elementary introduction to Apple Assembly Language programming by that old master Bob Bishop, along with Linda Gross-berger and Harry Vertelney. This 150+ page book and its companion diskette gently and humorously guide the beginning programmer into the realm of machine code. A "Cardboard Computer" introduces the concepts of registers, machine instructions, addressing, and branching. This background is then applied to the Apple's 6502 and the surrounding computer. AppleVisions is a nice place for the absolute beginner to start, especially the younger programmers interested in finding out what assembly language is.

AppleVisions. Addison-Wesley, 1985. $39.95 including diskette.

=============================================================================
DOCUMENT :AAL-8506:Articles:BernardsHexSrch.txt
=============================================================================

The Boyer-Morris String Search Algorithm..........Bob Bernard
Westport, CT

For years now, I have been working on a debugger for the Apple. Lately I have been adding a hex string search capability to it.

I needed one so I could look through the Apple IIc (ProDOS) utilites to see how it squirrels away in the alternate page screen holes user specified default settings for the serial ports. These are used at PR#1 or 2 time to simulate the dip switches on the Super Serial Card in a IIe. Without setting them you always get 9600 bps, etc. (Imagewriter settings, that is). I (and I assume other AAL readers) want a little routine for DOS 3.3 hello that will allow the user's defaults to be put away the same as the IIc utility does.

Well, that routine is not ready yet. However, the search utility is rather interesting in its own right.

I was just going to code up a straight hex search, but then I mentioned it to my computer science graduate son, David. He was horrified that I would waste my time on anything so crude. That's what I get for bringing up a programmer! David insisted that I should instead code an implementation of Boyer and Moore's algorithm, which appeared in the October 1977 issue of the Communications of the ACM. [A more recent reference is in the book "Algorithms", by Robert Sedgewick, (Addison-Wesley Publishing Co., 1983, 551 pages) on pages 249-252.]

Well, I read the article and it seemed like a challenge. Besides it looked like a real time saver, and could also be used for character string searches. The code here has been excerpted from my debugger, and then worked over by Bob S-C.

The "conventional" or "brute-force" search technique aligns the search pattern with the left end of the string to be searched through and compares one byte at a time, from left to right, until either the entire pattern is compared successfully or a mismatch occurs. In the latter case the search window is moved one byte to the right, and the comparing process is repeated.

Without any knowledge about the contents of the search pattern, the most the window can be moved is one place to the right. Boyer-Moore owes its speed advantage to the fact that it uses context (i.e. knowledge about the contents of the pattern to be searched for) to increase the distance that the search window can be advanced when a mismatch occurs. Thus efficiency increases as the length of the pattern increases, which does not happen in a conventional search.

The cost of this benefit (there always is a cost) is that a table (called DELTA1 in the CACM article and DELTA.TABLE in my program) is required to store this context information, 256 bytes in this implementation. One byte is needed in the table for every possible value of the characters in the string to be searched.

If a particular byte appears in the search pattern, then the corresponding DELTA table entry contains the distance that the rightmost occurrence of that byte is from the left end of the pattern. All other entries contain the value -1. When a mismatch occurs, the DELTA table entry corresponding to that byte from the text being searched is used to compute how far to advance the search window. If that byte does not appear anywhere in the pattern, then the search window can be advanced by the full length of the pattern.

Since moving the search window, and the associated testing for finished, take most of the time in any searching technique, saving time here can be extremely beneficial, and explains why Boyer and Moore should be complimented.

My program uses the control-Y monitor command, in the form

```
     adr1.adr2^Y <hexstring>
```

The two addresses specify the start and end of the area to be searched. "^Y" stands for "control-Y". The hex string may be separated from the control-Y by one or more spaces, if you desire. Since the control-Y doesn't show up on the screen, I usually type at least one space before the hex string. The hex string itself is a continuous string of hex digits, with no imbedded spaces. Here is an example that will search from $800 to $BFFF for "BERNARD":

```
     800.BFFF^Y 4245524E415244
```

The program will list the starting addresses of any and all complete matches that are found.

The maximum length of the hex string is limited by the monitor input buffer. Since the longest command you can type is less than 256, and you have to use around ten characters for the addresses and control-Y, that puts an upper limit of less than 246 hex digits in your command. Each byte of the search pattern (or "key") is made up of two hex digits, so the maximum hex string will be less than 123 bytes long.

I assigned DELTA.TABLE to the area $02D0.03CF. Since I scan and collect the search pattern right in the monitor keyboard buffer at $0200, after converting to hex bytes it will run no higher than $027F.

Actually, I only implemented a simplified version of Boyer and Moore's procedure. The CACM article also discusses a second table, DELTA2, which is filled with additional context information regarding "terminating substrings" of the search pattern. In cases where a partial mismatch occurs, it may be possible to advance the search window farther than the DELTA1 table would indicate. However, since such situations occur in less than 20% of the cases, David allowed that the potential additional speed did not justify the time and effort and the additional table and code space that would have been required, and he gave me a passing grade on my effort without it. The incorporation of this additional capability, and changes to make the program an ASCII search, are left "as a exercise for the reader."

My program must go through several steps. First it has to find and pack up the search key. Next it must build the DELTA table. And finally the search can be performed.

Lines 1290-1360 will be executed when you BRUN the program. They install the control-Y vector and jump into the monitor, just as though you entered with CALL-151.

When you enter the search command, the Apple monitor parses the command line up to and including the control-Y, and then branches to my code at line 1380. The two addresses will have been converted and stuffed into A1 ($3C,3D) and A2 ($3E,3F). A variable named YSAV (at $34) contains the index to the next character following the control-Y.

Lines 1400-1440 skip over any blanks you may have typed between the control-Y and the first hex digit. Actually, the Y-register gets incremented once too often, so lines 1460-1470 decrement Y and save it; now YSAV points to the first hex digit in the search key.

The next problem I had to solve was to differentiate odd from even length strings and arrange them properly, adding a leading zero when an odd number of hex digits is input. Lines 1490-1530 search for the end of the hex string; if there are no digits at all, we are finished and line 1530 returns for the next monitor command.

This is a nice place to insert a brief description of the NXTCHAR subroutine, found in lines 2460-2590. NXTCHAR picks up the next character from the input buffer, and tests to see if it is a hex digit. If so, it returns either $00-09 or $FA-FF in the A-register, and carry will be clear. If not a hex digit, it returns with carry set. If we got a digit, the Y-register indexing the input buffer will have been advanced.

Lines 1550-1590 compute the key length. Since two digits make a byte, the number of digits in the hex string divided by two gives the number of bytes. But I actually want to use the byte-count-minus-one. Also I need to adjust for odd or even length strings. Lines 1600-1650 take care of these details. If the count was odd, I jump into the middle of the packing loop so that a leading zero gets inserted.

Lines 1670-1800 comprise the packing loop. NXTCHAR will return with carry set when we try to get a digit beyond the end of the key, so line 1680 is the only test in the loop. Lines 1670-1730 retrieve a left-hand digit and store it in the buffer. Lines 1740-1800 do the same for right-hand digits. Key bytes are stored starting at $0200, so they never catch up to the advancing retrieval of digits.

Line 1810 sets YSAV to point to the first character past the end of the hex string. This will usually be a carriage return, or another monitor command. Unless it is beyond $2CF, the monitor will correctly continue parsing whatever is in the buffer when we are through searching. At $2D0 and beyond, the DELTA table will clobber any further characters.

Now we come to the Boyer-Moore part. Lines 1820-1870 initialize the DELTA table to all -1 values, which is what we want for any bytes not present in the key. When the loop finishes, X=0 again.

Lines 1880-1970 scan through the search key from left to right, and store into DELTA the index of the rightmost occurrence of each value in the key. For example, if the key is "4245524E415244" ("BERNARD" again), the DELTA values will be:

```
     DELTA+$41:  4
     DELTA+$42:  0
     DELTA+$44:  6
     DELTA+$45:  1
     DELTA+$4E:  3
     DELTA+$52:  5 (also at 2, but 5 is rightmost)
    all others:  -1
```

We'll continue with this example after a brief look at the rest of the code.

Lines 1980-2040 back up the end pointer, which has been patiently waiting all this time in A2L and A2H. We subtract the key length (in bytes, not digits) from the end pointer, so that we will not try to match the key any further than necessary. We could do this inside the search loop, but it will run faster if we do it once before the loop.

Lines 2050-2440 perform the search. I inserted lines 2070-2110 inside the loop to printout the search window start address each time through the loop. This helps me to make sure it is working, and to explain how. Of course you should remove these five lines before using the routine for real problems. Notice they are all marked "<<<DEBUG>>>".

Lines 2120-2170 check whether the beginning of the search window has moved past the end of the area to be searched. If so, we are finished.

Lines 2180-2240 compare bytes from the key and the search window. If the entire key matches, we fall out of the loop into lines 2250-2300, where the address of the match will be printed. After a successful match the search window will be moved one byte to the right by lines 2370-2430, and we will begin the SEARCH.LOOP again.

Notice that the key is compared from right-to-left, not left- to-right. This is a critical part of the Boyer-Moore method. If a key byte does not match a search-window byte, we branch to line 2320. The byte from the search window is in the A-register. Lines 2320-2370 compute how far we can advance the search window, based on just what character we DID find in the search window, and how far into the key we had already matched.

To see how this works, let's continue the "BERNARD" example. Suppose the text we are searching is "THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO." The key will be BERNARD, entered in hex as shown above. We first try to match BERNARD at the beginning of the text. We start at the right end, matching the "D" of the key with "A" of the text. The match fails, so we look up the "A" value in the DELTA table, which is 4. We subtract the delta value (4) from the current key index (6) and add the result (6-4=2) to the search window address. Note that this has the result of aligning the "A" of BERNARD with the "A" in the text.

Back to the top, and we now try to match the "D" of BERNARD to the "E" at the end of "ARE". Failure again! This time the DELTA value is 1, and we are still at position 6 in the key: index-delta is 5, so we advance the window by 5. This lines up the "E" of BERNARD with the E of the text. The next attempted match will find a blank in the text, which does not occur in the key at all. The delta value for blank is -1: 6-(-1)=7, so we will advance the window by 7. Now the window is up to "ST. BER" in the text.

When we compare "D" of BERNARD to "R" in the text, we fail again. The delta value for R is 5. There are two R's in BERNARD, but the rightmost one is at index 5. We can move the search window by 6-5=1. Next we try "D" against "N". The delta value of "N" is 3, so we can move the window 6-3=3 bytes. This time we have found "BERNARD"!

If you count it all up, we have compared the "D" of BERNARD with only six characters, and already we are at the first occurrence of the whole key in the text. A conventional search would have tried to match the first character of the key ("B") with all 18 characters in the text which precede the first "B" of the text. We have saved 13 times around the main loop! Of course, our loop is a tiny bit longer, but the end result is faster.

Here is a step-by-step picture of the entire search, which finds BERNARD twice:

```
     THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO.
     BERNARD
       BERNARD
            BERNARD
                   BERNARD
                    BERNARD
                       BERNARD  (success!)
                        BERNARD
                               BERNARD
                                  BERNARD
                                       BERNARD  (success!)
                                        BERNARD
                                               BER... (end)
```

I have tacked two more examples onto the end of the source code, at lines 2620-2690. You can play with them. The five <<<DEBUG>>> lines will print out the window address at each step, so you can see how the search progresses. Remember to take those lines out before you make a production version of the program.

If you decide to include this search algorithm in your own private debugger program, like I am, you might want to add the ability to use an ASCII string for the key. You could use a quotation mark after the control-Y to signal the packer loop that an ASCII string follows. You might also want to add single-byte wildcard characters, and/or the ability to ignore the high-order bit of each byte matched.

Perhaps the Boyer-Moore algorithm would be even more useful in a data base program, a word processor, or other context in which you are searching through huge quantities of text for relatively interesting keys. My example should get you started, and my son will be proud of you!
1

=============================================================================
DOCUMENT :AAL-8506:Articles:DP18.Leftovers.txt
=============================================================================

Some Final DP18 Subroutines................Bob Sander-Cederlof

Gerald Ferrier (Princeton, Minnesota) wrote to point out that we somewhow omitted a double-handful of subroutines from our lengthy series on 18-digit arithmetic for Applesoft.  With apologies to you all, and thanks to Gerald, here they are:

<<<double column listing>>>

```
=============================================================================
DOCUMENT :AAL-8506:Articles:Firmware.27128.txt
=============================================================================
```

Two ROM Sets in One Apple //e..............Bob Sander-Cederlof

If and when you decide to upgrade to the new enhanced //e ROMs (which Apple sells for $70 along with a 65C02), you will probably have to turn your old ROMs over to the store that makes the switch. Reportedly, Apple is binding the stores with a contract that forces them to collect all the old chips.

That is VERY unfortunate.  It could lead to wild shouting and panic, when you discover some of your favorite old software no longer works.

The upgrade consists of three parts:

*  the new processor chip (65C02), which is nice but not especially useful until software which uses its new features becomes available;

*  a new character generator ROM which includes special characters for icons and line drawing in text mode (called the "mouse" characters).

*  new CD and EF ROMs which upgrade the firmware.

The new firmware does NOT use any of the new features in the 65C02, so you could use it without the new cpu chip.  Furthermore, there is no absolute requirement to have the new character generator installed. The new firmware is much better than the old, having lost some bugs and speeded up the 80-column scrolling and added lower-case support to Applesoft (among other things).  It is compatible with the 6502, the 65C02, and the new 65802.

I personally do not yet have any use for the mouse characters, and do not expect to.  Don Lancaster, in the June 1985 issue of "Computer Shopper", tells how to connect a 2764 EPROM in the character generator socket.  The 2764 can hold two complete character sets, because it has twice the capacity of the 2732 normally in that socket.  However, the socket has only 24 holes and the 2764 has 28 pins!  Don shows how to wire this up with a socket adapter, and use a toggle switch to select either half.

And now Apple has "sort of" released an even more enhanced set of firmware, with debugging stuff built in.  You may not see them on the open market for some time, but I like them even better than the standard enhanced ROMs.  The "debug" ROMs add an absolute RESET (ctrl-RESET with solid apple), 16-byte hex display in the monitor when in 80-column mode, display of both hex and ASCII values of each byte in a memory dump, and the ability to use all monitor commands on both main and auxiliary memory.  The disassembler and miniassembler are both present, and enhanced to include the 65C02 extensions.

The CD and EF ROM sockets are compatible with 2764 EPROMs. You can also use 27128s, which have twice the space. Pin 26 on the 2764 is always tied to +5 volts. On a 27128, pin 26 selects the top or bottom half of the 16K bytes inside. You can burn one set of firmware in one half, and the other set in the other half. Then bend out pin 26 a little, so that it does not go into the socket when you insert the chip. Attach a clip lead to the bent-out pin, and connect the other end to either +5 volts or ground, to select the half you want at any given time.

You can connect it to a toggle switch, or just stick the bare end of a wire into the game paddle connector. If you use the game socket on the motherboard, pin 1 is +5 volts and pin 8 is ground. Or stick a wire into one of the annunciator outputs (pins 12, 13, 14, and 15) so you can flip back and forth between firmware sets by software control.

It can be a little tricky to make a copy of the ROM firmware and get it into RAM or on a disk, so that you can later burn it in your own EPROM. Especially in the Cxxx part. My approach, since I have more than one Apple, is to put my SCRG PromGramer card in a different machine. Then one by one I can read the //e ROMs and burn them into the appropriate 27128s. This a lot faster than trying to figure out how to flip all the //e soft switches so as to get at the different banks of Cx ROM code.

I have recently seen 27128s priced as low as $5 and as high as $20, in the back of Byte magazine. It is well worth it to invest in a PromGramer, at $140, and an EPROM eraser ($50 to $100 from Logical Devices in Florida, see Byte ads). You can keep your Apple standard for commercial software, and still have your own private firmware on the motherboard at the flip of a switch!

==========================================================================
DOCUMENT :AAL-8506:Articles:Front.Page.txt
==========================================================================

$1.80

Volume 5 -- Issue 9                     June, 1985

In This Issue...



S-C Macro Assembler ProDOS

We will begin shipping the ProDOS version of the S-C Macro Assembler in July, so we are now accepting advance orders.  There is more to the ProDOS version than just a change of operating systems.  The new upgrade includes a couple of major new features:

.INB (INclude Blocked) directive -- This works just like .IN, except that only one disk block at a time is overlaid into memory.  Allows assembly of much larger files, with only a minor speed penalty.

.AC  (Ascii Compressed) directive -- Generates compressed strings from a string between delimiters, according to rule tables.  Very complex, but worth the effort if you have a lot of messages and need to save memory.

The price of the ProDOS version alone will be $100.  The up- grade from DOS Version 2.0 to ProDOS will be $30.  The upgrade from DOS Version 1.0 or 1.1 to ProDOS will be $50, and will include DOS Version 2.0.  The initial purchase price of the DOS 3.3 and ProDOS versions together will be $120.  These are introductory prices which may well be raised in a few months.

65802's Are Here!

After many months of manufacturing delays, Western Design Center is shipping 65802 and 65816 microprocessors.  We recently received a final production '802, and it's now happily processing away in Bob's oldest Apple II (#219).  You can order the chips from WDC for $95.00. Call (602) 962-4545.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050.

Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

```
===============================================================================
DOCUMENT :AAL-8506:Articles:Johnsons.Call.txt
===============================================================================
```

A CALL Utility for Applesoft...................David C. Johnson
Applied Engineering

Anyone who has ever used Applesoft eventually realizes that the most powerful statement in the language is CALL.  It allows you to get to the Monitor for instance (however, Extended Debugging Monitor users have a better way).  When writing a program in BASIC, you invariably will want to do something that is at best difficult and often impossible to code using the other Applesoft statements.

The solution to this type of situation is to speak to your Apple in its native tongue.  There are several way this can be done.  Ampersand (&) routines are a popular technique.  The USR( function even has its uses.  The most logical way, for me, is the CALL statement.

Using CALL neatly transfers control from the Applesoft interpreter to whatever you want to do in machine language.  The one disadvantage to CALL is that the processor's registers do not contain useful data when your machine code gets control.

The CALL utility presented in this article will allow you to specify, as part of the CALL statement, the contents of any or all of the registers upon entry of your machine language subroutine.  You assign the register contents with LET-like structures.  Obviously you can only fit an 8 bit value into the 8 bit registers and the program counter value will probably be a 16 bit number.  Here's how the CALL statement should be written:

```
         CALL 768,PC=word,A=byte,X=byte,Y=byte,P=byte
```

The expressions "word" and "byte" may be any valid Applesoft numeric expression.  This is because the utility calls routines internal to Applesoft to evaluate the expressions.  If an expression results in a value larger than the register to which it is being assigned, or isn't numeric, or is invalid, you will get one of the usual errors.  The commas shown separating the register assignments are required (syntax error if comma missing).  The equals characters ("=") are also required.  The register names ( PC, A, X, Y, & P) must be upper case on older Apples, while the newer firmware will convert lower case for you (or in spite of you).  The register assignments may appear, after the first comma, in any order and need not all be specified. Unspecified registers will be loaded with their last used value. Previously unused registers default to zero, except the P-register which defaults to $04 in order to set the interrupt disable flag.

The program is well commented, but I'll add one more note of caution. Readers with Apples containing reqular 6502s (not 65C02s or 65802s) should avoid re-assembling the code with the label PC.Sav's bytes falling across a page boundary ($XXFF).

The program was written using the ProDOS version of the S-C Macro Assembler 2.0, while I was beta testing it for Bob.  It works GREAT!

```
=================================================================
DOCUMENT :AAL-8506:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners........................$20
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                 Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains        1980     -         -         -         1
 the source code from      1981     2         3         4         5
 three issues of AAL,      1982     6         7         8         9
 saving you lots of        1983    10        11        12        13
 typing and testing.       1984    14        15        16        17
                           1985    18        19

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100

Blank Diskettes (Verbatim)........................ package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers............................  6 cents each

quikLoader EPROM System (SCRG)................................($179) $170
PROmGRAMER (SCRG).........................................($149.50) $140
D MAnual Controller (SCRG).....................................($90)  $85
Switch-a-Slot (SCRG)..........................................($190) $175
Extend-a-Slot (SCRG)...........................................($35)  $32

"Apple ProDOS:  Advanced Features for programmers", Little..($17.95)  $17
"Inside the Apple //c", Little..............................($19.95)  $18
"Inside the Apple //e", Little..............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler......................($22.95)  $21
"Understanding the Apple II", Sather........................($22.95)  $21
"Understanding the Apple //e", Sather.......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
```

```
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
"6502 Assembly Language Programming", Leventhal.............($18.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18
"AppleVisions", Bishop & Grossberger........................($39.95)  $36
```

Add $1.50 per book for US shipping.  Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

```
==============================================================================
DOCUMENT :AAL-8506:Articles:Note.65802.txt
==============================================================================
```

Note on the TXS instruction in the 65802...Bob Sander-Cederlof

Sandy Greenfarb wrote the other day that he had received a 65802 and plugged it into his Basis 108 with success.

He has been trying various permutations of the new opcodes and modes, and discovered some stones are better left unturned:

"The following programs should both print the letter "A" on the screen. However, the one on the left works, while the one on the right hangs up the computer."

```
       Works           Hangs Up
       --------        --------
       CLC             CLC
       XCE             XCE
       LDA #"A         LDA #"A
       JSR $FDF0       JSR $FDED
       SEC             SEC
       XCE             XCE
       RTS             RTS
```

The only difference in the two programs is that the unsuccessful one weaves its way through DOS. I looked at the DOS code it goes through, and at first glance it appears there should be NO PROBLEMS associated with executing all this code in 65802 mode, since both 16-bit modes are off.

However, for some reason it still hangs up. Actually, it might not always hang: it depends on what is in page zero at the corresponding position as the stack pointer in page one.

I do not know why, but the TXS instruction transfers the entire 16-bit value of X to S when you are in the 65802 mode, regardless of the status of the M and X bits. Since M and X are both 1, the high byte of the X-register is 00. Therefore the TXS instruction at $9FB9 in DOS clears the high byte of the S-register. The RTS at $9FC4 then uses a return address from page zero, rather than page one.

I tried various experiments to see how TXS and TSX worked, and also examined TXA and TAX. In my humble opinion, the 65802 is inconsistent here. If you are in 65802 mode with M and X = 1, TXA does not modify the high byte of the A-register. This is what I expect and what I want. But TXS does modify the high byte of the S-register, contrary to my expectations.

Of course, as long as you know exactly how the chip works it really doesn't matter a lot. The problems come when we ASSUME we know how it works, but are wrong. The best antidote for these kind of assumptions, at least until a definitive reference manual for the chip is published, is trial and error.

I have had my 65802 for about six months now, and still have had no problems whatsoever with compatibility as long as I stay in normal 6502 mode.  If I leave it in 65802 and go charging through a program written for the 6502 mode, I expect I will run into trouble.

```
=============================================================================
DOCUMENT :AAL-8506:Articles:Putney.IRQTrace.txt
=============================================================================
```

Interrupt Trace...............................Charles H. Putney
Dublin, Ireland

Have you ever wondered what's happening when the Apple goes off into nothingness? If your answer is yes, then this short utility will help you find out.

I was recently debugging an assembly language program and ran into this problem. The program seemed to work for almost all the input data, but occasionally would hang. After several frustrating hours trying to simulate the event, I decided that an interrupt trace utility would solve my problems. Later when I had this utility working, it was easy to see why the program was hanging.

This utility consists of a pushbutton addition to the Apple which connects to the interrupt request line (IRQ) of the 6502 and an interrupt service routine which is in page three. When the interrupt pushbutton is depressed, the interrupt service routine displays the program counter and all the registers on the bottom line of the screen. It also displays a flashing cursor and waits for an "S" or "G" from the keyboard to stop or resume execution.

I have mounted a pushbutton switch at the upper right hand side of the keyboard in the center of the styling surface. For a temporary installation I suggest leaving the pushbutton on a flexible lead. The wiring is easily done with 30 gage wire wrap wire. One side of the pushbutton is connected to ground. You may solder a wire to any convenient ground point on the top of the circuit board. Or, for a temporary installation, you could stick a wire into pin 8 of the game I/O connector.

The other side of the pushbutton is connected to the IRQ signal. I found that signal at pin 4 of the 6502. Remove the 6502 from the socket and strip the insulation from the end of the 30 gage wire. Insert it in the socket for the 6502 in pin 4 and replace the 6502 to retain the wire. Route the wire along the chips for a neat installation.

For a temporary hookup, Bob S-C suggests folding a 3-by-5 card in half, and triming it so that the folded edge just fits into an empty slot. Then, while power to the Apple is off, slip one wire into the space between the card and pin 26 (ground) and the other wire between the card and pin 30 (IRQ). Both of these wires will be on the power-supply side of the card: pin 26 is at the back edge, and pin 30 is the fifth from the back. Once the wires are inserted, you may wish to tape them down.

Enter the routine at address $300 and BSAVE it. When you want to debug a hanging program, first BRUN the INTERRUPT TRACE utility. This installs the utility at address $300 to $3CA.  Pressing the pushbutton will cause an immediate display of the current program counter and registers.  The utility will wait with a blinking cursor for a "G" or "S" from the keyboard to continue or enter monitor.

Sometimes the program you're investigating may not respond to the pushbutton.  This is because somewhere in the program interrupts have been disabled with the SEI command ($78).  You must search through the entire program and replace these with a CLI instruction ($58).  Make sure that each $78 found is not data in the program and is a valid instruction before you replace it.

The next time that you have a problem with your Apple "hanging" for no apparent reason, use this  utility to see where the 6502 "is".  It may help solve those "hard to debug programs".

When you run the program, the SETUP routine (from $300 to $30B) sets the interrupt vector location and then enables interrupts.  When the pushbutton is depressed, the IRQ line (pin 4 on the 6502) is pulled low.  At the completion of the current instruction, the program counter high, program counter low, and processor status are pushed on the stack.  Interrupt disable is automatically set and the program counter is loaded with the contents of $FFFE and $FFFF.  In the Autostart monitor ROM the program counter is set to $FA40 where the monitor interrupt service routine is located. (In the old monitor the identical routine is at $FA86)  This routine saves the accumulator in $45 and examines the processor status register to see if the interrupt was caused by a BRK command.  Remember, the BRK command shares the same vector location with the interrupt for software simulation of interrupts.  If the interrupt was not caused by BREAK then a JMP indirect to location ($3FE) is performed.

Lines 1280-1290 save the X- and Y-registers.  The accumulator has already been saved by the monitor interrupt routine.

Lines 1300-1350 copy the register display titles to the bottom of the screen.  Of course, if your program happened to be in one of the full-screen graphics modes, this line will not be visible.  If you have a //e, you can add code to sense the graphics mode, save it, switch to text mode; then you will have to restore it all when you type "G" to continue after the interrupt.  The new enhanced //e ROMs automatically handle saving and restoring all the bank switched memory, but they still leave the graphics modes up to the programmer.

Lines 1360-1510 convert the values of the five registers and store them into the bottom line.  I add 3 to the S-register value before displaying it, so you see the value before the IRQ code pushed PC and S onto the stack.  I start with the Y-register pointing at the point on the bottom line where the A-register should be displayed.  The DISPLAY.HEX subroutine advances the Y-register by 5, so it is always ready for displaying the next register.

Lines 1520-1590 display the PC-register.  This value is taken from the stack, where the IRQ automatically saved it.

Lines 1600-1750 wait for you to type "G" or "S". While waiting, the last character on the bottom line is flashed to remind you to type. If you type "G", lines 1760-1800 restore the registers and return to the interrupted program. If you type "S", line 1820 takes you to the monitor.

==============================================================================
DOCUMENT :AAL-8506:Articles:SQRT16.txt
==============================================================================

Short Integer Square Root Subroutine.......Bob Sander-Cederlof

In some graphics situations you need a square root subroutine (it is probably the fault of Pythagoras).   Since the screen coordinates are integers, a short and fast integer square root subroutine can be handy.

The following program is probably not in the "fast" category, but it is indeed short.  It can produce the integer value of the square root of any integer from 0 through 65535.  The program uses the method of subtracting successive odd numbers.

Every perfect square (N*N, where N is an integer) is the sum of a series of odd numbers from 1 through 2*N-1.  Thus 4=1+3, 25=1+3+5+7+9, etc.

The program starts by subtracting 1, then 3, then 5, and so on until the remainder is negative.  When the remainder goes negative, the last odd number subtracted was 2*N+1, so we can get the square root by dividing that odd number by 2.

I set up the routine so I could test it with an Applesoft pro- gram. You can POKE the low 8-bits of a number at 768 ($300), the high 8-bits at 769, and CALL 772.  Upon return, PEEK(770)+ 256*PEEK(771) gives you the integer value of the square root.

I used a couple of tricks in the code.  For one, the variable ODD is always an even number.  Since I preface the subtraction with CLC, a "borrow" is assumed, so it has the effect of sub- tracting the odd number which is one larger than the even number in ODD.  This save a LDA #1 instruction after line 1090.

In lines 1190-1230, I add 2 to the even number in ODD.  But you can see that line 1200 is ADC #1.  This adds 2 because carry happens to be set.

```
==============================================================================
DOCUMENT :AAL-8506:DOS3.3:DIGITS.3.txt
==============================================================================

 1000  *SAVE DIGITS.3
 1010  *--------------------------------
 1020         .LIST OFF
 1030  *--------------------------------
 1040  BYTE   .EQ $00
 1050  COUT   .EQ $FDED
 1060  CROUT  .EQ $FD8E
 1070  PRBYTE .EQ $FDDA
 1080  *--------------------------------
 1090  *      COMMAND
 1100  *--------------------------------
 1110  P      LDA #0
 1120         STA BYTE
 1130  .1     JSR WRITE
 1140         JSR CROUT
 1150         INC BYTE
 1160         LDA BYTE
 1170         BNE .1
 1180         RTS
 1190  *--------------------------------
 1200  *      WRITE
 1210  *--------------------------------
 1220  WRITE  LDY #0
 1230         SEC
 1240  .1     SBC #10
 1250         PHP
 1260         PHA
 1270         TYA
 1280         SED
 1290         ADC #0
 1300         TAY
 1310         PLA
 1320         PLP
 1330         BCS .1
 1340         ADC #"0+10
 1350         PHA
 1360         TYA
 1370         JSR PRBYTE
 1380         PLA
 1390         JMP COUT
 1400  *--------------------------------
 1410  SC
 1420         LDY #"0"
 1430         TAX
 1440         BEQ .3
 1450         LDA #0
 1470         SED
 1475  .2     CLC
 1480         ADC #1
```

```
1490        BCC .25
1500        INY
1510 .25    DEX
1520        BNE .2
1530        CLD
1540 .3     PHA
1550        TYA
1560        JSR COUT
1570        PLA
1580        JMP $FDDA
1590 *--------------------------------
1600 T      LDA #0
1610 .1     STA BYTE
1620        JSR SC
1630        LDA BYTE
1640        JSR $FDDA
1650        JSR CROUT
1651        LDA $C000
1652        BPL .2
1653        STA $C010
1654 .3     LDA $C000
1655        BPL .3
1656 .2     STA $C010
1660        LDA BYTE
1670        CLC
1680        ADC #1
1690        BNE .1
1700        RTS
1710 *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8506:DOS3.3:DP18.MOVE.SUBS.txt
==============================================================================

```
 1000  *SAVE DP18.MOVE.SUBS
 2020  *--------------------------------
 2030  *      MOVE (Y,A) INTO DAC. YA IS UNPACKED
 2040  *--------------------------------
 2050  MOVE.YA.DAC.1
 2060         STA PNTR
 2070         STY PNTR+1
 2080         LDY #10      MOVE 11 BYTES
 2090  .1     LDA (PNTR),Y
 2100         STA DAC,Y
 2110         DEY
 2120         BPL .1
 2130         LDA DAC.EXPONENT
 2140         STA DAC.SIGN
 2150         AND #$7F
 2160         STA DAC.EXPONENT
 2170         RTS
 2180  *--------------------------------
 2190  *      MOVE (Y,A) INTO ARG. YA IS UNPACKED
 2200  *--------------------------------
 2210  MOVE.YA.ARG.1
 2220         STA PNTR
 2230         STY PNTR+1
 2240         LDY #10      MOVE 11 BYTES
 2250  .1     LDA (PNTR),Y
 2260         STA ARG,Y
 2270         DEY
 2280         BPL .1
 2290         LDA ARG.EXPONENT
 2300         STA ARG.SIGN
 2310         AND #$7F
 2320         STA ARG.EXPONENT
 2330         RTS
 2340  *--------------------------------
 2350  *      MOVE DAC TO (Y,A) WITHOUT PACKING
 2360  *--------------------------------
 2370  MOVE.DAC.YA.1
 2380         STA PNTR
 2390         STY PNTR+1
 2400         LDA DAC.EXPONENT
 2410         BPL .0
 2420         JMP DAC.YA.O.U  OVER- OR UNDER-FLOW
 2430  .0     BIT DAC.SIGN
 2440         BPL .1       POSITIVE
 2450         ORA #$80     NEGATIVE
 2460  .1     LDY #0
 2470  .2     STA (PNTR),Y
 2480         INY
 2490         LDA DAC,Y
```

```
2500        CPY #11      11 BYTES
2510        BCC .2
2520        RTS
2530 *--------------------------------
2540 MOVE.DAC.TEMP1
2550        LDA #DP.TEMP1
2560        LDY /DP.TEMP1
2570        JMP MOVE.DAC.YA.1
2580 *--------------------------------
2590 MOVE.TEMP1.ARG
2600        LDA #DP.TEMP1
2610        LDY /DP.TEMP1
2620        JMP MOVE.YA.ARG.1
2630 *--------------------------------
2640 MOVE.TEMP1.DAC
2650        LDA #DP.TEMP1
2660        LDY /DP.TEMP1
2670        JMP MOVE.YA.DAC.1
2680 *--------------------------------
2690 MOVE.DAC.TEMP2
2700        LDA #DP.TEMP2
2710        LDY /DP.TEMP2
2720        JMP MOVE.DAC.YA.1
2730 *--------------------------------
2740 MOVE.TEMP2.DAC
2750        LDA #DP.TEMP2
2760        LDY /DP.TEMP2
2770        JMP MOVE.YA.DAC.1
2780 *--------------------------------
2790 MOVE.TEMP2.ARG
2800        LDA #DP.TEMP2
2810        LDY /DP.TEMP2
2820        JMP MOVE.YA.ARG.1
2830 *--------------------------------
2840 MOVE.TEMP3.DAC
2850        LDA #DP.TEMP3
2860        LDY /DP.TEMP3
2870        JMP MOVE.YA.DAC.1
2880 *--------------------------------
2890 MOVE.TEMP3.ARG
2900        LDA #DP.TEMP3
2910        LDY /DP.TEMP3
2920        JMP MOVE.YA.ARG.1
2930 *--------------------------------
2940 MOVE.DAC.TEMP3
2950        LDA #DP.TEMP3
2960        LDY /DP.TEMP3
2970        JMP MOVE.DAC.YA.1
2980 *--------------------------------
```

```
=====================================================================
DOCUMENT :AAL-8506:DOS3.3:S.CALL.UTIL.txt
=====================================================================

 1000  *SAVE S.CALL.UTIL
 1010
 1020  * 6/13/85 dcj
 1030
 1040  * CALL 768{,pc=word,a=byte,x=byte,y=byte,p=byte}
 1050
 1060  *--------------------------------
 1070
 1080         .OR $300
 1090         .TF CU
 1100
 1110  EQ.TOK .EQ $D0      Applesoft '=' token
 1120
 1130  CHRGET .EQ $B1 -$C8 advance TXTPTR & fetch chr
 1140  CHRGOT .EQ $B7      just fetch chr
 1150
 1160  FRMNUM .EQ $DD67    evaluate FP expression (FAC)
 1170  SYNCHR .EQ $DEC0    require chr in Acc syntax @ TXTPTR
 1180  SYNERR .EQ $DEC9    syntax error
 1190  GETBYT .EQ $E6F8    evaluate 8 bits @ TXTPTR (X-reg)
 1200  GETADR .EQ $E752    convert FAC to 16 bits in Acc & Y-reg
(hi/lo)
 1210
 1220  *--------------------------------
 1230
 1240  CALL.UTIL
 1250
 1260         JSR CHRGOT   get chr after call adr expression
 1270         CMP #','     comma indicates more stuff follows
 1280         BEQ .1       =>go continue parsing
 1290         LDA P.SAV    load registers
 1300         PHA           (P-reg via stack)
 1310         LDA ACC.SAV
 1320         LDX X.SAV
 1330         LDY Y.SAV
 1340         PLP
 1350         JMP (PC.SAV) go 4 it!
 1360
 1370  * we got something to parse
 1380
 1390  .1     JSR CHRGET   get chr after comma
 1400         CMP #'A'     (as in 'Acc')
 1410         BEQ .2       =>go get '=' & byte for Acc
 1420         CMP #'X'     (as in 'X-reg')
 1430         BEQ .3       =>go get '=' & byte for X-reg
 1440         CMP #'Y'     (as in 'Y-reg')
 1450         BEQ .4       =>go get '=' & byte for Y-reg
 1460         CMP #'P'     (as in P-reg or Program Counter)
 1470         BEQ .5       =>go get '=' or 'C='...
```

```
 1480         JMP SYNERR   razz
 1490
 1500  * pickup Acc byte
 1510
 1520  .2     JSR .7       require '=' (@ next) & fetch byte exp
 1530         STX ACC.SAV  stuff it
 1540         BVC CALL.UTIL ...always
 1550
 1560  * pickup X-reg byte
 1570
 1580  .3     JSR .7       require '=' (@ next) & fetch byte exp
 1590         STX X.SAV    stuff it
 1600         BVC CALL.UTIL ...always
 1610
 1620  * pickup Y-reg byte
 1630
 1640  .4     JSR .7       require '=' (@ next) & fetch byte exp
 1650         STX Y.SAV    stuff it
 1660         BVC CALL.UTIL ...always
 1670
 1680  * Finish parsing 'P=' or 'PC='
 1690
 1700  .5     JSR CHRGET   advance to next chr position & fetch it
 1710         CMP #'C'     (as in 'Program Counter')
 1720         BEQ .6       =>go get '=' & 16 bits for PC
 1730
 1740  * pickup P-reg byte
 1750
 1760         JSR .10      require '=' @ current chr position
 1770         JSR .8       fetch byte expression
 1780         STX P.SAV    stuff it
 1790         BVC CALL.UTIL ...always
 1800
 1810  * pickup PC word
 1820
 1830  .6     JSR .9       require '=' @ next chr position
 1840         JSR FRMNUM   fletch FP expression
 1850         JSR GETADR   convert FP expression to Acc & Y-reg
(hi/lo)
 1860         STY PC.SAV   stuff 'em
 1870         STA PC.SAV+1
 1880         JMP CALL.UTIL no flag known...
 1890
 1900  .7     JSR .9       require '=' @ next chr position
 1910
 1920  .8     JSR GETBYT   fetch byte expression (2 X-reg)
 1930         CLV          to force branch
 1940         RTS
 1950
 1960  .9     JSR CHRGET   1st advance to next chr position
 1970
 1980  .10    LDA #EQ.TOK  require '=' before register expressions
 1990         JMP SYNCHR    (SYNTAX ERROR IF '=' NOT FOUND)
 2000
```

```
2010  *--------------------------------
2020
2030  ACC.SAV    .DA #$00
2040  X.SAV      .DA #$00
2050  Y.SAV      .DA #$00
2060  P.SAV      .DA #$04
2070  PC.SAV     .DA $0000
2080
2090  *--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8506:DOS3.3:S.HEX.SEARCH.txt
==================================================================

 1000  *SAVE S.HEX.SEARCH
 1010  *--------------------------------
 1020  *   MEMORY SEARCH FOR HEX STRING
 1030  *      BY BOB BERNARD, MAY 17, 1985
 1040  *      MODIFIED BY BOB S-C, MAY 27TH
 1050  *   ADR1.ADR2^YXXXXXXXXXXXX
 1060  *      ("^Y" MEANS CONTROL-Y)
 1070  *
 1080  *   SEARCH MEMORY FROM ADR1 THRU ADR2
 1090  *   LOOKING FOR REFERENCES TO
 1100  *   THE HEX STRING, XXXXXXXXXX
 1110  *
 1120  *--------------------------------
 1130  YSAV        .EQ $34
 1140  A1L         .EQ $3C,3D   START OF SEARCH AREA
 1150  A2L         .EQ $3E,3F   END OF SEARCH AREA
 1160  KEY.LENGTH  .EQ $40      (MONITOR'S A3L)
 1170  *--------------------------------
 1180  KBDBUF      .EQ $0200 THRU $2CF
 1190  DELTA.TABLE .EQ $02D0 THRU $3CF
 1200  USRADR      .EQ $03F8   CTL-Y JUMPS HERE
 1210  *--------------------------------
 1220  PRINTAX     .EQ $F941
 1230  CROUT       .EQ $FD8E   NEW LINE
 1240  MONZ        .EQ $FF69   MONITOR, NO BELL
 1250  *--------------------------------
 1260          .OR $0800
 1270          .TF B.HEX.SEARCH
 1280  *--------------------------------
 1290  HEX.SEARCH
 1300          LDA #$4C       JMP OPCODE
 1310          STA USRADR       STUFF INTO CNTL-Y EXIT LOC
 1320          LDA #SEARCH      LO ADR
 1330          STA USRADR+1
 1340          LDA /SEARCH      HI ADR
 1350          STA USRADR+2
 1360          JMP MONZ         MONITOR, NO BELL
 1370  *--------------------------------
 1380  SEARCH
 1390  *---SKIP LEADING BLANKS----------
 1400          LDY YSAV         NEXT VALID KBDBUF CHAR
 1410  .1      LDA KBDBUF,Y     GET CHAR FROM
 1420          INY              KEYBOARD BUFFER
 1430          CMP #" "         SKIP LEADING BLANKS
 1440          BEQ .1
 1450  *---MARK KEY START---------------
 1460          DEY
 1470          STY YSAV         WHERE SCAN STARTS
 1480  *---FIND END OF KEY--------------
```

```
1490  .2       JSR NXTCHAR
1500           BCC .2           ...HEX DIGIT
1510           CPY YSAV         CHECK FOR NULL KEY
1520           BNE .3           ...NOT NULL
1530           RTS              NULL KEY
1540  *---COMPUTE KEY LENGTH-----------
1550  .3       TYA
1560           SBC YSAV
1570
1580           LSR
1590           STA KEY.LENGTH
1600           LDY YSAV
1610           LDX #0
1620           STX KBDBUF       (IN CASE ODD COUNT)
1630           BCS .5           ...ODD NUMBER OF BYTES
1640  *---ADJUST FOR EVEN LENGTH-------
1650           DEC KEY.LENGTH MAKE EVEN LENGTH ONE LESS
1660  *---LEFT NYBBLE-----------------
1670  .4       JSR NXTCHAR
1680           BCS .6           END OF KEY
1690           ASL
1700           ASL
1710           ASL
1720           ASL
1730           STA KBDBUF,X     LEFT HALF DEST CHAR
1740  *---RIGHT NYBBLE----------------
1750  .5       JSR NXTCHAR
1760           AND #$0F
1770           ORA KBDBUF,X     MERGE HI NIBBLE
1780           STA KBDBUF,X
1790           INX
1800           BNE .4           ...ALWAYS
1810  .6       STY YSAV
1820  *---INIT ALL DELTAS=-1 ----------
1830           LDX #0
1840           LDA #-1
1850  .7       STA DELTA.TABLE,X
1860           INX
1870           BNE .7     ...256 OF THEM
1880  *---DELTA(KEY(I))=I-------------
1890           LDY #0           FOR I=0 TO KEYLEN
1900  .8       LDA KBDBUF,Y    DELTA(K) = DISTANCE FROM LEFT END
1910           TAX                  OF RIGHT-MOST OCCURENCE OF
1920           TYA                  8-BIT VALUE "K" IN KEY.
1930           STA DELTA.TABLE,X
1940           INY              NEXT I
1950           CPY KEY.LENGTH
1960           BCC .8
1970           BEQ .8
1980  *---ADJUST END OF SEARCH---------
1990           SEC
2000           LDA A2L
2010           SBC KEY.LENGTH
2020           STA A2L
```

```
2030          BCS SEARCH.LOOP
2040          DEC A2L+1
2050 *---------------------------------
2060 SEARCH.LOOP
2070        LDA A1L+1    <<<DEBUG>>>
2080        LDX A1L      <<<DEBUG>>>
2090        JSR PRINTAX  <<<DEBUG>>>
2100        LDA #"-"     <<<DEBUG>>>
2110        JSR $FDED    <<<DEBUG>>>
2120          LDA A2L           CHECK AGAINST
2130          CMP A1L            UPPER BOUND
2140          LDA A2L+1          FOR SEARCH
2150          SBC A1L+1
2160          BCS .1            A1<=A2, NOT FINISHED
2170          RTS               A1>A2, FINISHED
2180 *---COMPARE IN THIS POSITION-----
2190 .1       LDY KEY.LENGTH   FOR I=KEYLEN TO 0
2200 .2       LDA (A1L),Y      CHECK BYTES FROM
2210          CMP KBDBUF,Y        RIGHT TO LEFT
2220          BNE .3            ...DID NOT MATCH
2230          DEY               NEXT I
2240          BPL .2
2250 *---MATCH FOUND------------------
2260          LDX A1L           PRINT ADR
2270          LDA A1L+1         WHERE MATCH
2280          JSR PRINTAX          WAS FOUND
2290          JSR CROUT         NEW LINE
2300          JMP .4
2310 *---ADVANCE SEARCH POINTER-------
2320 .3       TAX               STRING CHAR JUST LOOKED AT
2330          TYA               I
2340          CLC
2350          SBC DELTA.TABLE,X
2360          BPL .5            ...VALUE IS POSITIVE
2370 .4       LDA #0            ...ADVANCE BY 1
2380 .5       SEC               COMPENSATE
2390          ADC A1L
2400          STA A1L
2410          BCC SEARCH.LOOP
2420          INC A1L+1
2430          BNE SEARCH.LOOP  ...ALWAYS, UNLESS WE
2440          RTS               ...RAN OFF THE END OF MEMORY
2450 *---------------------------------
2460 NXTCHAR
2470          LDA KBDBUF,Y     NEXT ACTIVE CHAR
2480          INY
2490          EOR #$B0          CONVERT ASCII TO DIGIT
2500          CMP #10           0..9?
2510          BCC .1            YES
2520          ADC #$88          SHIFT RANGE FOR A-F TEST
2530          CMP #$FA          A..F?
2540          BCS .1            YES. EXIT CC
2550          SEC               NOT HEX CHAR
2560          DEY               BACK UP INDEX
```

```
 2570           RTS
 2580  .1       CLC               SIGNAL HEX CHAR
 2590           RTS
 2600  *--------------------------------
 2610  END    .BS $A00-*
 2620  TEST.STRING
 2630         .AS /XXXXXXXCOCACACACACACACACACACAC/
 2640  * TRY A00.A1F^Y  43414341434143
 2650  * SHOULD GET A09-A0B-A0D-A0F-A11-A13-A15-A17-A19
 2660  *--------------------------------
 2670  TS2    .AS /A STRING SEARCHING EXAMPLE CONSISTING OF SIMPLE
TEXT/
 2680  * TRY A20.A53^Y48494E47
 2690  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8506:DOS3.3:S.IRQ.TRAPPER.txt
==============================================================================

 1000  *SAVE S.IRQ TRAPPER
 1010  *--------------------------------
 1020  *       INTERRUPT TRACE UTILITY
 1030  *
 1040  *       BY: CHARLES H. PUTNEY
 1050  *           18 QUINNS ROAD
 1060  *           SHANKILL
 1070  *           COUNTY DUBLIN
 1080  *           IRELAND
 1090  *--------------------------------
 1100  A.REG       .EQ $45      A-REG SAVE AREA USED BY MONITOR
 1110  STACK       .EQ $100     STACK PAGE
 1120  INTVEC      .EQ $3FE     INTERRUPT VECTOR
 1130  BOTTOM.LINE .EQ $7D0     LINE 24 OF TEXT SCREEN
 1140  *--------------------------------
 1150  KEYBD  .EQ $C000    KEYBOARD DATA
 1160  KEYSTB .EQ $C010    KEYBOARD STROBE
 1170  MNTR   .EQ $FF69    MONITOR ENTRY POINT (CALL -151)
 1180  *--------------------------------
 1190         .OR $300     PAGE THREE
 1200  *--------------------------------
 1210  SETUP  LDA #INT     LOAD IRQ VECTOR
 1220         STA INTVEC   LOW BYTE
 1230         LDA /INT
 1240         STA INTVEC+1 HIGH BYTE
 1250         CLI          ALLOW IRQ'S
 1260         RTS
 1270  *--------------------------------
 1280  INT    STX XREG     SAVE X (A-REG SAVED BY MONITOR)
 1290         STY YREG     SAVE Y
 1300  *---DISPLAY REG TITLES-----------
 1310         LDX #39      PUT UP MESSAGE LINE
 1320  .1     LDA TITLES,X  GET MESSAGE CHAR
 1330         STA BOTTOM.LINE,X   PUT ON SCREEN
 1340         DEX
 1350         BPL .1       DONE ?
 1360  *---DISPLAY REG VALUES-----------
 1370         LDY #10      START OF REG DISPLAY AREA
 1380         LDA A.REG    ...A-REG
 1390         JSR DISPLAY.HEX
 1400         LDA XREG     ...X-REG
 1410         JSR DISPLAY.HEX
 1420         LDA YREG     ...Y-REG
 1430         JSR DISPLAY.HEX
 1440         TSX          GET STACK POINTER
 1450         INX          POINT AT PROCESSOR STATUS
 1460         LDA STACK,X  ...P-REG
 1470         JSR DISPLAY.HEX
 1480         INX          ADJUST S-REG
```

```
       INX
       TXA          ...S-REG AS WAS BEFORE INTERRUPT
       JSR DISPLAY.HEX
*---DISPLAY PC-REG---------------
       LDY #0       START OF PC-REG DISPLAY
       LDA STACK,X  GET PC HIBYTE
       JSR DISPLAY.HEX
       DEX
       LDY #2
       LDA STACK,X  GET PC LOBYTE
       JSR DISPLAY.HEX
*---WAIT FOR "S" OR "G"----------
.2     LDA KEYBD    KEY PRESSED ?
       BPL .3       NO
       STA KEYSTB   CLEAR THE KEY
       CMP #"G"     GO AHEAD ?
       BEQ .4       YES
       CMP #"S"     STOP ?
       BEQ .5       YES
.3     DEX          BLINK CURSOR
       BNE .3
       DEY          LONGER DELAY
       BNE .3
       LDA BOTTOM.LINE+39  LAST CHAR ON SCREEN
       EOR #$80     INVERT IT
       STA BOTTOM.LINE+39  REPLACE IT
       BNE .2       BRANCH ALWAYS
*---"G" TYPED, RETURN------------
.4     LDA A.REG      RESTORE THE REGISTERS
       LDX XREG
       LDY YREG
       RTI          BACK TO WORK
*---"S" TYPED, SO STOP-----------
.5     JMP MNTR     ENTER THE MONITOR
*--------------------------------
DISPLAY.HEX
       PHA          SAVE THE A-REG
       LSR          SHIFT INTO LOWER NIBBLE
       LSR
       LSR
       LSR
       JSR DIGIT    MAKE IT A DIGIT
       STA BOTTOM.LINE,Y   SHOW HIGH NIBBLE
       PLA          GET A-REG AGAIN
       AND #$0F     MASK IT
       JSR DIGIT    MAKE IT A DIGIT
       STA BOTTOM.LINE,Y   SHOW LOWER NIBBLE
       INY
       INY
       INY
       RTS
*--------------------------------
DIGIT  INY
       ORA #$B0     ADD NUMBER ZERO
```

```
2030        CMP #$BA     IS IT A LETTER
2040        BCC .1       NO - DONE
2050        ADC #$6      6 PLUS CARRY MAKES A
2060 .1     RTS
2070 *--------------------------------
2080 TITLES .AS -/     -   A=   X=   Y=   P=   S=         /
2090 *--------------------------------
2100 XREG   .DA #*-*     X REGISTER SAVE AREA
2110 YREG   .DA #*-*     Y REGISTER SAVE AREA
2120 *--------------------------------
```

```
=========================================================================
DOCUMENT :AAL-8506:DOS3.3:S.LovesConvers.txt
=========================================================================

 1000  *SAVE S.LOVE'S CONVERSION
 1010  *--------------------------------
 1020         .LIST OFF
 1030  *--------------------------------
 1040  BYTE   .EQ $00
 1050  COUT   .EQ $FDED
 1060  CROUT  .EQ $FD8E
 1070  PRBYTE .EQ $FDDA
 1080  *--------------------------------
 1090  *      COMMAND
 1100  *--------------------------------
 1110  P      LDA #0
 1120         STA BYTE
 1130  .1     JSR PRINT.000.255
 1140         JSR CROUT
 1150         INC BYTE
 1160         LDA BYTE
 1170         BNE .1
 1180         RTS
 1190  *--------------------------------
 1200  *      PRINT.000.255
 1210  *--------------------------------
 1220  PRINT.000.255
 1230         LDY #0
 1240         SEC
 1250  .1     SBC #10
 1260         PHP
 1270         PHA
 1280         TYA
 1290         SED
 1300         ADC #0
 1310         TAY
 1320         PLA
 1330         PLP
 1340         BCS .1
 1350         ADC #"0+10
 1360         PHA
 1370         TYA
 1380         JSR PRBYTE
 1390         PLA
 1400         JMP COUT
 1410  *--------------------------------
 1420  SC
 1430         LDY #"0"
 1440         TAX
 1450         BEQ .3
 1460         LDA #0
 1470         SED
 1480  .2     CLC
```

```
1490        ADC #1
1500        BCC .25
1510        INY
1520 .25    DEX
1530        BNE .2
1540        CLD
1550 .3     PHA
1560        TYA
1570        JSR COUT
1580        PLA
1590        JMP $FDDA
1600 *--------------------------------
1610 T      LDA #0
1620 .1     STA BYTE
1630        JSR SC
1640        LDA #" "
1650        JSR $FDED
1660        LDA BYTE
1670        JSR $FDDA
1680        JSR CROUT
1690        LDA $C000
1700        BPL .2
1710        STA $C010
1720 .3     LDA $C000
1730        BPL .3
1740 .2     STA $C010
1750        LDA BYTE
1760        CLC
1770        ADC #1
1780        BNE .1
1790        RTS
1800 *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8506:DOS3.3:S.SQRT16.txt
==========================================================================

```
 1000  *SAVE S.SQRT16
 1010  *--------------------------------
 1020         .OR $300
 1030  ARG    .BS 2
 1040  ODD    .BS 2
 1050  *--------------------------------
 1060  SQRT   LDX ARG+1    X = HI BYTE   HI
 1070         LDY ARG      Y = LO BYTE   LO
 1080         LDA #0       START ODD=0
 1090         STA ODD+1
 1100  .1     STA ODD
 1110         CLC          BORROW ON, SUBTRACT (ODD+1)
 1120         TYA          LO
 1130         SBC ODD
 1140         TAY
 1150         TXA          HI
 1160         SBC ODD+1
 1170         TAX
 1180         BCC .2       ...ODD>REMAINDER, FINISHED
 1190         LDA ODD      CARRY SET, ADD 2 TO ODD
 1200         ADC #1
 1210         BCC .1       ...NEXT
 1220         INC ODD+1
 1230         BNE .1       ...ALWAYS     ...ALWAYS
 1240  .2     LSR ODD+1    SQRT IS (ODD/2)
 1250         ROR ODD
 1260         RTS
 1270  *--------------------------------
```

```
===============================================================================
DOCUMENT :AAL-8506:DOS3.3:TEST.SQRT16.txt
===============================================================================

Ë≤TEST SQRT16:ÌW–256:A–768:B–769:C–770:D–771:E–772ZÚÅXH–
0¡255:∫XH"-";:ÅXL–0¡255m \πA,XL:πB,XH:åE Y–,(D) W»,(C)°≠Y—
œ”(⁄(XH W»XL))ƒ∫:∫XH W»L,Y©$Ç:Ç
```

================================================================
DOCUMENT :AAL-8507:Articles:BSave2NewFile.txt
================================================================

Allow BSAVE to New Non-Binary Files in BASIC.SYSTEM 1.1
................Mark Jackson
Chicago, Illinois

I consider it a bug:  BASIC.SYSTEM doesn't allow BSAVEing to a new file unless the type is binary.  Yet it is equally desirable to be able to BSAVE to non-binary files without first CREATEing them.

I discovered this problem while implementing FIG-FORTH in ProDOS when I wanted to save the data blocks using as little code as possible, and at the same time allow use of standard text-file word processors.

BSAVEing would solve the code length problem, but to make a text file I would have had to CREATE the file first, thus decreasing speed and increasing code length.  Therefore I looked for the BSAVE code inside BASIC.SYSTEM to fix the bug.

As it comes from Apple, BASIC.SYSTEM's parser puts the specified type in $BE6A and then the BSAVE processor places it there again.  I used the space this redundant code took for my patch.

There seems to be no good reason for Apple to purposely prevent BSAVEing to new non-binary files, so I think my patch is both worthwhile and safe.

The following applies only to Apple's BASIC.SYSTEM version 1.1, which is the latest as far as I know.  The addresses shown are the actual running position.  If you want to patch the SYS file by BLOADing at A$2000, then addresses $ADxx will be at $37xx and addresses $AExx will be at $38xx.

The following is in the CREATE code.

Now is:

```
     AD41- A9 0F    LDA #$0F     DEFAULT SYS FILE
     AD43- 8D 6A BE STA $BE6A    PUT IN GLOBAL PAGE
```

Change to:

```
     AD41- A2 0F    LDX #$0F
     AD43- 8E 6A BE STX $BE6A
```

The following is in the BSAVE code, and is only reached if it is a new file:

Now is:

```
     ADF5- A9 06    LDA #$06     ASSUME TYPE IS BIN
```

```
     ADF7- 8D 6A BE STA $BE6A    PUT IN GLOBAL PAGE
     ADFA- 8D B8 BE STA $BEB8    SET-FILE-INFO LIST
     ADFD- AD 56 BE LDA $BE56    CHECK IF TYPE GIVEN
     AE00- 29 04    AND #$04
     AE02- D0 0E    BNE $AE12    IF YES, THEN ERROR
     AE04- 20 46 AD JSR $AD46    CREATE NEW FILE
```

Change to:

```
     ADF5- AE 6A BE LDX $BE6A    FILE TYPE FROM PARSING
     ADF8- AD 56 BE LDA $BE56    CHECK IF TYPE GIVEN
     ADFB- 29 04    AND #$04
     ADFD- D0 02    BNE $AE01    IF YES SKIP DEFAULT
     ADFF- A2 06    LDX #$06     DEFAULT BIN FILE
     AE01- 8E B8 BE STX $BEB8    SET-FILE-INFO LIST
     AE04- 20 43 AD JSR $AD43    GO CREATE FILE
```

Thanks to Don Worth and Pieter Lechner for their help in disassembling, through their book "Supplement to Beneath Apple ProDOS." (This is the book you get by sending in $10 and a coupon from Beneath Apple ProDOS.)

==============================================================================
DOCUMENT :AAL-8507:Articles:Front.Page.txt
==============================================================================

$1.80

Volume 5 -- Issue 10                    July, 1985

In This Issue...



ProDOS Macro Assembler

We are now shipping the ProDOS version of the S-C Macro Assembler.  As reported last month, the ProDOS version alone is $100 and the DOS and ProDOS versions together are $120.  The ProDOS update for owners of the DOS Version 2.0 is $30, and for owners of DOS Version 1.x is $50.

The S-C Cross Reference Utility and the Laumer Research Full Screen Editor have been updated to ProDOS versions.  The ProDOS code will be included on the back of the disk in all new shipments, and current owners can return their original disks to be updated at a cost of only $5 per program.

65802 Chips

Good News!  We have arranged a quantity price on 65802 processors, so we will be able to sell them to our readers for only $50 + shipping. That's only $51.50 in the US for this powerful new 16-bit processor that plugs right into your Apple II, II+, //e, or //c.  Combine this chip with the S-C Macro Assembler Version 2.0 and you can start writing faster, more compact code.  Order yours today!

Updated VideoTerm Driver

We recently revised the Videx VideoTerm driver in the S-C Macro Assembler Version 2.0 to make it firmware-independent and ViewMaster-compatible.  This revision is effective with Serial Number T-1483, so owners of earlier copies can send in their original disks and $5 for an updated copy.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

```
=================================================================
DOCUMENT :AAL-8507:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0 (DOS or ProDOS)......................$100
S-C Macro Assembler Version 2.0 (DOS and ProDOS).....................$120
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners........................$20
Source Code for Version 1.1 (on two disk sides)......................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................each $15, or any four for $45
                                  Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains       1980       -         -         -         1
 the source code from     1981       2         3         4         5
 three issues of AAL,     1982       6         7         8         9
 saving you lots of       1983      10        11        12        13
 typing and testing.      1984      14        15        16        17
                          1985      18        19

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
ES-CAPE:  Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100

Blank Diskettes (Verbatim)......................... package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50
quikLoader EPROM System (SCRG)...............................($179) $170
PROmGRAMER (SCRG).........................................($149.50) $140
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32

"Apple ProDOS:  Advanced Features for programmers", Little..($17.95)  $17
"Inside the Apple //c", Little..............................($19.95)  $18
"Inside the Apple //e", Little..............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler......................($22.95)  $21
"Understanding the Apple II", Sather........................($22.95)  $21
"Understanding the Apple //e", Sather.......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
```

```
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
"6502 Assembly Language Programming", Leventhal.............($18.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18
"AppleVisions", Bishop & Grossberger........................($39.95)  $36

 Add $1.50 per book for US shipping.  Foreign orders add postage needed.
      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

```
==================================================================
DOCUMENT :AAL-8507:Articles:New.Cat.Revisit.txt
==================================================================
```

New Catalog Revisited........................Robert F. O'Brien
Dublin, Ireland

In the May issue of AAL Bob S-C published my article "A New Catalog for Dos 3.3" - he failed to mention or take credit for the fact that he modified my routine and managed to leave a whopping 17 spare bytes - which is 16 more than I left.  I was happy enough to have added the new features.

At the end of that article Bob S-C set the challenge to add the Disk Volume message back.  However, I have another possible use for those 17 spare bytes - well at least 14 of them!

How about a single-key format control feature for the Catalog command? The user issues the CATALOG command normally; then one more keypress will select either a normal or double- barrelled Catalog display.

Once you install the following additional code, when you issue the CATALOG command the routine waits for a keypress.  If you press "D" you get a double-barrelled Catalog listing for your 80-column card or printer.  Any other keypress will result in the normal 40-column version.

The line numbers on the 14-byte routine which follows make the code fit into the listing from the May article.

```
               1320 CATALOG
AD98- 20 0C FD 1321        JSR MON.RDKEY     await keypress
AD9B- 49 8E    1322        EOR #$8E   "D" ($C4) eor LSR ($4A)
AD9D- C9 4A    1323        CMP #$4A   if was "D", now LSR
AD9F- F0 02    1324        BEQ .0     ...it was "D"
ADA1- A9 38    1325        LDA #$38   SEC opcode
ADA3- 8D 21 AE 1326 .0     STA DBL.SWITCH    set option
               .
               .
               .
               2010 DBL.SWITCH SEC
               .
               .
               .
               2150        .BS 3         three free bytes.
```

The code above is of the deadly self-modifying variety, so beware.

Note that if you have version 2.0 of the S-C Macro Assembler, you can write line 1322 as EOR #"D"^$4A if you wish.

==============================================================================
DOCUMENT :AAL-8507:Articles:ProDOS.DOS.Load.txt
==============================================================================

Reading DOS 3.3 Disks With ProDOS..........Bob Sander-Cederlof

At the track and sector level, DOS 3.3 disks are identical to ProDOS disks.  They both have 35 tracks, 16 sectors, and the sectors are laid out on the tracks the same way in both systems.  You can use DOS's COPYA program to copy ProDOS disks, and you can use some ProDOS utilities on DOS disks.

The structure of the files is of course entirely different between the two systems.  Hence the need for the CONVERT program found on ProDOS system master disks, and the System Utilities Disk that comes with the //c.  Unfortunately both of the above programs have bugs that get in the way nearly every time I want to move a file from DOS to ProDOS. The one that bites me the most is the way CONVERT dies when it encounters a DOS filename which does not start with a letter.  We routinely use such "illegal" filenames on our disks to separate and identify sections of long catalogs, but CONVERT goes absolutely crazy when it finds one.

Therefore, I decided to write a program which could "LOAD" assembler source files from a DOS 3.3 disk while I am running the ProDOS version of the S-C Macro Assembler.  Even with error messages and other fancy features, the program turns out to be only a little over $280 bytes long, and it works.

It is based on the fact that the Block Read MLI call does not care whether the disk being read is a DOS or a ProDOS disk.  The Block Read MLI call reads 512 bytes, or two sectors, at a time.  The call looks like this:

```
       JSR $BF00       (MLI link in global page)
       .DA #$80        (block read code)
       .DA PARMLIST    (address of parameters)
```

MLI returns with carry clear if there was no error, or carry set if there was an error.  The error code will be in the A-register if there was an error.

The PARMLIST for Block Read looks like this:

```
    PARMLIST .DA #3         (3 parameters)
             .DA #$60       (1-byte unit number)
             .DA BUFFER     (address of 512-byte buffer)
             .DA 2          (2-byte block number)
```

Page 3-17 of "Beneath Apple ProDOS" contains a table which converts block numbers to physical track/sector, and vice versa.  The latest printing of the book also includes a line which correlates the physical sector values to the DOS 3.3 logical sector.  Boiling it down, you can derive a ProDOS block number from the DOS 3.3 logical sector by multiplying the track number by 8 and adding a value according to the sector number from the following table:

```
    DOS sector #:  0 1 2 3 4 5 6 7 8 9 A B C D E F
                   0 7 6 6 5 5 4 4 3 3 2 2 1 1 0 F
```

For example, track 0 sector 2 is in ProDOS block 6. The only problem is, so is DOS track 0 sector 3. We also need to remember whether a given sector is in the upper or lower half of a 512-byte block.

I developed the following subroutine, which will translate the DOS logical track and sector numbers into the appropriate block number, read the block, and return with the address of the buffer page in which the sector data has been read. Call the routine with the track number in the A-register and the sector number in the X-register. The high-byte of the buffer address will return in the X-register. If MLI detects an error, the subroutine will return with carry set.

```
RTS    LDY #0  ASSUME BLOCK # < $100
       ASL     FORM TRACK*8
       ASL
       ASL
       BCC .1  ...BLOCK < $100
       INY     ...BLOCK > $0FF
.1     ASL     *2, MAKE ROOM FOR H/L FLAG BIT
       ORA BLKTBL,X   MERGE FROM SECTOR TRANSLATION
       ROR     H/L FLAG BIT TO CARRY
       STA BLOCK
       STY BLOCK+1
       LDX /BLOCK.BUFFER   HIGH BYTE OF BUFFER ADDRESS
       BCC .2  ...LOWER HALF OF BUFFER
       INX     ...UPPER HALF OF BUFFER
.2     JSR $BF00
       .DA #$80,PARMLIST
       RTS

BLKTBL .HS 00.0E.0D.0C.0B.0A.09.08
       .HS 07.06.05.04.03.02.01.0F

PARMLIST
       .DA #3
       .DA #$60        SLOT 6, DRIVE 1
       .DA BLOCK.BUFFER
BLOCK  .DA 0           <FILLED IN>
```

After playing with the subroutine a while, I proceeded to write the load program. Using a well-worn copy of "Beneath Apple DOS", I figured out once more how to work through a DOS catalog. I decided to display a menu of files on the screen, and allow a single keystroke to select a file to be loaded.

The program that follows is designed to work with the ProDOS version of the S-C Macro Assembler. Assuming it has been assembled and is in a ProDOS binary file as DOS.LOAD, and assuming you have booted the ProDOS version of the S-C Macro Assembler, you can start up the load program by typing "-DOS.LOAD". It will load source files from DOS disks, which are DOS type I files, and place them in the assembler's edit area. After selecting the slot and drive, the program reads the DOS catalog and displays 20 filenames at a time. Only type I filenames are displayed, any others are skipped over. If there are more than 20 files, you can page through them. If you change your mind about loading a file, you can abort. If you see the file you want to load, you type a single letter to select it. A few seconds later it has been loaded, and you are returned to the assembler.

The assembler's soft entry point is at $8003, and the load program jumps there after finishing a load or after encountering an error. Three pointer locations in page zero which the assembler uses are used by the load program: HIMEM ($73,74) points one byte higher than the program can be loaded; PP ($CA,CB) will point to the beginning of the program, if it is successfully loaded; LOMEM ($67,68) points to the lowest address the program can occupy. HIMEM is normally at $7400, and LOMEM at $1000, but these can be changed with the HIMEM and LOMEM commands. LOMEM could be set as low as $0800.

With these limitations on the program extent ($0800...73FF), you can see that the maximum size assembler source file that can be loaded from a DOS disk is $6C00 bytes, or 108 sectors. Or, if you prefer to leave LOMEM at $1000, you can load $6400 bytes or 100 sectors. Most likely you do not have any source files which are bigger than that anyway. If you do, you need to load the DOS version of the assembler and split the files before they can be transferred to ProDOS. The maximum size file of 108 data sectors would only have one track/sector list, so I did not include any logic to chain to a second track/sector list. You may be wondering where the load program itself loads....

The command interpreter I developed for the ProDOS version of the S-C Macro Assembler has three 1024-byte buffers permanently allocated between $7400 and $7FFF. None of them will be in use while the load program is executing, so I borrowed some of that space for the load program. The load program itself loads inside the buffer space allocated to the EXEC command, at $7400-77FF. The blocks read by MLI will be stored at $7C00-7DFF, and I will save a copy of the track/sector list for the file being loaded at $7E00-7EFF.

Now for a description of the actual code. Lines 1270-1410 ask you to type in the slot and drive numbers of the floppy drive the DOS disk is in. ProDOS uses a "unit number", which is a coded form of the slot and drive all in one byte. The slot number is in bits 4-6, and the drive number (0 or 1, corresponding to drives 1 or 2 respectively) in bit 7. My subroutine GETNUM prints a prompt message (selected by the Y-register), inputs a single character from the keyboard, and checks it for legal range. GETNUM is designed to accept only digits, starting with "1", and up to but not including the value in the A-register when GETNUM is called.

Once the unit number has been established, we fall into the LOAD.MENU code. This code is somewhat convoluted, enough to disgust even me. Interlocking loops? Multiple entries and exits? Ouch! Maybe it really IS structured code, but just not in Euclidean space. I think maybe it could be diagrammed on the surface of a Klein bottle (recursive torus?).

Anyway, let's walk through it. Line 1440-1500 set up a fresh menu display and read in the DOS VTOC page so we can start reading the catalog. The second and third bytes in the VTOC page give the track and sector of the first catalog sector. This is almost always track $11, sector $0F; however, by starting at VTOC, we are a little more general. We are still assuming we know where the VTOC is, which is track $11, sector 0. Some non-standard software sets up disks with the VTOC somewhere else, but you are very unlikely to find any S-C source code on such a disk. Each sector of the catalog also contains the track/sector of the next catalog sector in the 2nd and 3rd bytes.

Lines 1530-1550 read in the next catalog sector and set the pointer to the first file entry in that sector. Each file entry is 35 bytes long, and the first one starts at $0B within the sector. The subroutine READ.NEXT.CATALOG.SECTOR will return with carry set if there are no more catalog sectors. The first time through this code, when we fall in from the code above, we will read the first catalog sector.

Lines 1570-1960 pick up filenames out of the catalog sectors and write them on the screen. Not all file names are used: line 1610 filters out deleted files; lines 1660-1700 filter out files which are not type I. The track and sector of the active type-I files are saved in an array, indexed by the menu letter. These values are first picked up in lines 1620-1650, and added to the array in lines 1870-1940. Lines 1720-1770 print the menu letter and two dashes, and then lines 1780-1850 print the filename.

Lines 1950-1960 decrement the line count and test if the screen is full yet. I arbitrarily call a screen full if it has 20 filenames, leaving room for my three-line prompt message. We jump to MENU.SELECTION when we reach 20 lines or when we reach the end of the catalog, whichever comes first.

If we are not yet at the end of catalog and have not yet filled the screen, or if the file was one that got filtered out of the menu, we come to GET.NEXT.FILE at line 1980. Lines 1990-2040 update the pointer into the catalog sector so that it points at the next file, if there is another one. If so, we branch back to NEXT.FILE.NAME, to try the next one in the current sector. If no more names in this sector, we go back to NEXT.CAT.SECTOR to get the next catalog sector (if any).

When we reach the end of catalog, lines 2070,2080 set a flag. We need a flag to tell whether it was screen-full or catalog- end which caused us to come to MENU.SELECTION, so we can either continue through the catalog or wrap-around to the beginning should you wish to see another screenful of filenames.

The MENU.SELECTION section prints a three-line prompt message and waits for you to type a character. If you type a space, you seethe next screenful of filenames. (Of course, if there are fewer than 21 type I files on the disk you will see the same ones over again.) If you type the RETURN or ESCAPE keys, the load program will abort, returning directly to the assembler without loading a file. If you type a letter in the range of the menu, that file will be loaded. Any other key is ignored.

Lines 2260-2370 convert the menu letter you typed into an index to get the track and sector for the track/sector list of the selected file. The track/sector list contains the track and sector for every data sector in the file. Line 2310 reads the track/sector list, and lines 2330-2370 copy it into a special buffer.

The first two bytes of the first data sector of a type-I file contain the length of the file. We need to know the length so we can figure out where to read the data. Lines 2390-2510 read in the first data sector and get the file size.

Lines 2520-2630 figure out where PP should be set so that the file exactly fits between PP and HIMEM, and checks to make sure that it does not go below LOMEM.

Lines 2650-2670 copy the rest of that first sector into the load area, starting at PP. If the file is so short it doesn't fill the first data sector, the LOAD.FROM.SECTOR subroutine will return with carry set and we will return to the assembler, all finished. Otherwise, we fall into the code below, to load the succeeding data sectors. Eventually we will bump into HIMEM, and we are finished.

Now that this program is working I can see neat ways to extend it. Why restrict it to type-I files? It could also BLOAD type-B files, as long as an appropiate load address was set up. It could do the equivalent of a BLOAD on a type-T file, which then could be BSAVE as type TXT in ProDOS. Seems like we might be able to do away with the need for CONVERT, at least in the direction of moving from DOS to ProDOS.

=============================================================================
DOCUMENT :AAL-8507:Articles:Recursive.Cat.txt
=============================================================================

Multi-Level ProDOS Catalog.................Bob Sander-Cederlof

Last week I looked through some old piles of papers and came across a program by Greg Seitz, dated Dec 20, 1983. It was attached to a set of ProDOS Tech Notes, and Greg apparently worked at Apple at that time.

Greg's program lists the filenames of an entire ProDOS directory, showing the whole tree. It shows directory files by printing a slash in front of the filename, and shows the level by indenting. For example, a typical listing might look like this:

```
       PRODOS
       BASIC.SYSTEM
       /UTILITIES
        HELPER
        DOER
        /MORE
         WHATEVER
         AND.ANOTHER
        TEXT.FILE
       ANOTHER
```

A listing like this can be a big help in finding things on a large hard disk. The program can also be extended in many ways. One that comes to mind immediately is to print the rest of the CATALOG information as well as the file names. Another is to create a complete CATALOG MANAGER utility, which would permit re-arranging the filenames, promoting and demoting files, and so on.

I typed in Greg's program, and then I rewrote it. The listing that follows bears very little resemblance to his code, but I do thank him for the help in getting started.

The program assumes a prefix has been set. If there is no prefix, you will get a beep and no listing. If there is a prefix, and the directory named is online, the listing will begin with that directory. Another enhancement would be to display the current prefix, and allow accepting it or changing it before starting the filename listing.

If we were always starting with the volume directory, it would be a little easier. The volume directory always starts in block 2. However, since we are able to start with any directory, we do not know where it starts. ProDOS allows you to read a directory, and we can get the first block of any directory by using MLI to open the directory file.

Lines 1100-1120 read the current prefix into a buffer. The lines 1130-1150 open that file. Although I have never seen it in the books, apparently OPEN also reads the first block. After the OPEN call, BUFFER.ONE contains the first block of the directory file. Unless we are willing to do a complete search without ProDOS's help, this is the only way I know of to find the first block of a directory file (other than the volume directory).

Since the only reason to OPEN the directory file was to read the first block, lines 1180-1200 close it again. If any of these MLI calls don't go through, line 1210 will ring the alarm and stop.

Lines 1230-1260 start up the directory listing. The first block ONLY will be in BUFFER.ONE. All subsequent blocks will be read into BUFFER.TWO. In order to make the LIST.DIRECTORY program completely recursive, it is called with the buffer address in a zero-page pointer. SETUP.NEXT.BLOCK also gets the next block pointer from the buffer and saves it in NEXT.BLOCK.

LIST.DIRECTORY is really quite simple, in spite of its size. Its main function is to print a list of filenames. Each filename is preceded by a number of blanks, determined by NEST.LEVEL. NEST.LEVEL is incremented at line 1290, each time LIST.DIRECTORY is called. If a file listed happens to be a directory file, LIST.IDRECTORY saves all the pointers and counters on the stack and then calls itself. When the subdirectory's files have all been listed, that recursive call of LIST.DIRECTORY will return, the pointers and counters can be unstacked, and the listing can continue.

The format of the information in a directory is detailed quite well in both "Beneath Apple ProDOS" and "Apple ProDOS Advanced Features". (We recommend and sell both books.) The first four bytes of each block are two block numbers: that of the previous block, and that of the next block, in the same directory. This allows scanning in both forward and reverse directions through a directory. We will only use the next-block pointers in our program. After the block numbers there are 13 descriptors of 39 bytes each. The first descriptor in a directory describes the directory itself, and the rest describe files.

For some reason Apple was not quite sure that it would always use 13 39-byte descriptors, so they stored these two numbers in the directory descriptor. Anyone who access a directory is supposed to look up these two numbers and use them, just in case Apple decides to change them someday. The directory descriptor also contains an active file count. When a file is deleted this count is decremented, but the file descriptor remains. We use the active file count to determine when we reach the end of a directory. Lines 1300-1360 pick up the bytes per descriptor, descriptors per block, and active file count and save them.

Lines 1370-1450 set up PNTR to point at the first file descriptor, which follows the directory header. CURRENT.ENTRY.NUMBER will count up to 13, so we will know when it is time to read another block. We start at 2, because the first block uses the first descriptor for the header. We also clear the file count.

Lines 1460-1500 check for the special case of an empty directory. If there are no active files, we are finished.

Lines 1510-1750 print out the file name from the current file descriptor. The first byte of a descriptor contains a code for the type of file in the first nybble, and the length of the file name in the second nybble. If both are zero, the file has been deleted. The other legal values are $1x, $2x, and $3x to signify a seedling, sapling, or tree file, respectively; and $Dx to signify a directory file. All we care about is whether is a directory file or not, and how long the file name is.

If it is a directory file, lines 1760-2100 will be executed. Lines 1760-1860 push the counters and pointers on the stack. Lines 1870-1930 read in the first block of the sub-directory. Line 1950 calls LIST.DIRECTORY to list the subdirectory. After it is finished, line 1960 will decrement the nesting level. Lines 1970-2060 unstack the pointers and counters. If we were still in the first block of the highest level directory (where we started), we do not need to read the block again: it is still in BUFFER.ONE. Otherwise, lines 2070-2100 read the block back in. If we did not care how much memory we used, we could make this program a lot faster by using more buffers. We could have a different buffer for each level, so that blocks would never have to be re-read.

Lines 2110-2210 count the file just listed, and then check to see if our count is the same as the active file count from the directory header. If so, we are finished.

If we are not finished, lines 2220-2290 bump the pointer into the directory block by the size of a descriptor entry. If we are still in the same block, that is all that we need to do. If not, lines 2350-2420 read in the next block and set things up for it. Then it's back to the top again for the next file name!

We hope some time in the not-so-distant future to be able to write a complete catalog manager program like I started to describe back at the beginning of this article. Some of you are using Bill Morgan's CATALOG ARRANGER for DOS 3.3, and this would be an equivalent utility for ProDOS. We're not quite ready yet, but this program is a step in the right direction.

==================================================================
DOCUMENT :AAL-8507:Articles:SpeedDemon.txt
==================================================================

Review of M-c-T SpeedDemon.................Bob Sander-Cederlof

Is the Apple II a slow machine? Hey, it MUST be! After all, it is over 8 years old! It only has an 8-bit microprocessor! It only has a 1-MHz clock! It must be many times slower than today's PC clones, etc. Isn't it?

No.

The 6502 is inherently faster than most other microprocessors. An old rule of thumb had it that a 4-MHz Z-80 ran roughly the same speed as a 1-MHz 6502. Other factors, such as memory speeds, overhead for screen and keyboard, and disk I/O also influence the overall speed, often in favor of the venerable Apple.

Some comparisons come to mind with machines from the past. Anyone remember MIT's "Whirlwind"? A long time ago, its speed was considered super. I'll bet it wasn't as fast as an Apple. According to the book, it had an upper limit of 2048 16-bit words of "high-speed" memory, and had a design limit of 50,000 instructions per second. In actual implementation, it only ever achieved 20,000 operations per second. And that was with a 1 MHz clock! The 6502 with a 1 MHz clock runs from 500,000 to 142,000 operations per second, depending on which ones you are doing. Probably an average of 250,000.

How about the Bendix G-15? It was the "personal" computer of the 1950's, roughly the size of a large refrigerator (much warmer though) and selling for only $50,000. Engineering firms bought them eagerly for their friendly features, amazing flexibility, capacity, and speed. Let's see.... G-15 had 2183 words of RAM, on a magnetic drum, 29 bits per word. Most operations were measured in milliseconds. A floating point interpretive package, called Intercom 500 (or 1000 for double precision), could almost keep up with the typewriter (an IBM Executive, the primary user I/O device). Paper tape cassettes served as handy off-line storage devices.

Some other popular systems were considered fast with memory cycle times over ten microseconds per byte. Fast enough to support several users in a timesharing environment, compile large Fortran programs, and manage large businesses. And usually with smaller than 128K bytes of RAM. Or "core", as we called it in those days.

Nevertheless, Apples often seem slow. Because we ask them to do a lot, and don't want to wait around while it is done. And tolerable waiting times one day seem intolerable the next, because we get used to it. Remember when a trip around the world in 80 days seemed impossibly fast?

Perceived necessity being a prime motivator for innovation, several methods for dramatically accelerating Apples have been developed. Titan Technologies markets the Accelerator, and Microcomputer Technologies (McT) the SpeedDemon. These both promise "up to" 3.5 times faster running speed, and actually deliver an average of over 2 times faster.

We have wanted to try one of these boards for years. The price was too high and our faith too low, so we never bought one. Recently the price has dropped considerably, and reports from friends using them have increased our faith. When McT offered to loan us one for a month, we had no more resistance at all.

Imagine this scenario: the card arrives by UPS at noon. Thirty seconds later we have it in our hands, and are trying to find an Apple with at least one empty slot. Despairing of that, we take out a card and make room for the SpeedDemon in our //e. We turn on the //e, load up the S-C Macro Assembler, and proceed to assemble the biggest program we have. Wow! That's fast!

We promptly ran a lot of speed tests, timing various programs we commonly use around here:

| | | | |
|---|---|---|---|
| S-C Word Processor | | | |
| Load 89 sectors | 6.8 | 5.5 | 1.2 |
| Search /###/ | 10.4 | 3.3 | 3.2 |
| Replace /85/##/ | 8.3 | 2.8 | 3.0 |
| Mail Label System (primarily Applesoft) | | | |
| Load 48 sectors | 23.7 | 13.8 | 1.7 |
| Sort - last name | 140.6 | 49.1 | 2.9 |
| Sort - zip code | 56.0 | 20.0 | 2.8 |
| S-C Macro Assembler | | | |
| Assemble 771 lines | 7.2 | 3.0 | 2.4 |
| AppleWorks Data Base | | | |
| Load 47K | 25.7 | 25.0 | 1.0+ |
| Sort - last name | 2.2 | 1.0 | 2.2 |
| Sort - zip code | 5.0 | 2.0 | 2.5 |
| AppleWorks Spreadsheet | | | |
| Load 35K | 20.3 | 19.3 | 1.1 |
| Recalculate | 14.9 | 6.6 | 2.3 |
| Insert 9 rows | 4.9 | 1.8 | 2.7 |

In a review by Lee The, Personal Computing, Jan 85, the Apple with SpeedDemon was compared to a Compaq PC. Lee compared the systems using word processors on the two machines. The accelerated Apple ran faster in most cases, except when disk I/O was involved. In one case, even an un-accelerated Apple ran faster; the SpeedDemon to Compaq ratio was 4.4!

To summarize, the SpeedDemon really does make your software run faster. The absolute maximum speedup factor is 3.5, but no "real" program would achieve it. The two things that keep you from reaching 3.5 are I/O and memory.

Some I/O cards, notably the disk interface, use software timing. If you speed up the processor while trying to read or write the disk, you are in trouble. SpeedDemon automatically slows down to normal Apple speed when you access slot 6. Jumpers on the card allow you to do the same for slots 4 and 5. I have a disk controller in slot 7 in one of my Apples; I cannot read or write to disks using that controller when the SpeedDemon is active.

Old Apple serial interface cards used software timing loops to convert a byte to a bit stream at a given baud rate. These cards normally were placed in slots 1 or 2, and thus would not be compatible with the SpeedDemon. Modem cards sometimes use software timing for dialing, and they would not work right if accelerated. Any sound effects created through the Apple speaker will be raised way up in pitch. Music cards which depend on timing loops will make a whole new kind of sound.

The card can be turned off in two ways, so the above problem areas can be circumvented. During the power up cycle you have about two seconds during which you may tap the ESCAPE key. If you do, the card will be turned off. Then you hit ctrl-RESET to go into a normal boot. Another way to turn off the card is to store anything into $C05B (POKE 49243,0). After the POKE the Apple will lock up; when you hit ctrl-RESET it will come back in normal speed. There is no way to turn the card back on without turning off the Apple. (Some of you can probably find a way to re-wire it so it could be turned back on.)

The other way the card slows down is during memory access. Apple memory can only be accessed at a 1 MHz rate, so the processor can spend time waiting for memory. SpeedDemon has a 4096-byte cache memory which can run at a full 3.58 MHz rate. The cache is implemented with 4 static RAM chips, providing 8192 bytes of RAM. These are paired so that you get 4096 data bytes and 4096 address bytes. Whenever you read a byte from RAM or ROM, the low-order 12 bits of the address select one of thes 4096 byte pairs. The high 4 bits of the address are compared to the 4 bits in the cache; if they are the same then the data in the cache is presumed to be the data you want. If not, the processor will wait for Apple's memory to read, and then update the cache with the result. Something like that, anyway. Stores into memory always slow down to a 1 MHz rate, because the stores MUST be performed in real RAM, not just cache RAM.

I might have been talking through my hat in the above paragraph. There is no technical documentation available on the SpeedDemon, so I am just deducing the way it works from external appearances.

The Titan Accelerator card has a full 64K RAM, rather than a cache. It is therefore a little bit faster. Reports from those who have tried both indicate Titan is only about 10 percent faster, if that much. Of course you could design artificial situations in which the difference would be much more dramatic. Personally I think I would rather have the cache. And also the cash, since SpeedDemon costs about $25 less.

Titan's card draws about 300 ma at 5 volts, SpeedDemon draws about 600 ma. Titan's card uses more CMOS, and is more sensitive to static electricity.

SpeedDemon uses a 65C02, so you have the additional opcodes and address modes of this enhanced 6502 chip available. I believe you could romove the 65C02 plug a 65802 into the socket and gain even greater enhancements. You would have to have a 65802 rated at 4MHz, but the ones I have are only 2 MHz chips.

There are five PLA's on the SpeedDemon. At least some of these are used to keep track of whatever bank switching you do with Apple's RAM and ROM. Somehow they are able to keep track of the RAMWORKS card too, so the cache doesn't get confused even with a megabyte of RAM. I worry about using it with my STB128 card, or the other cards of the type. Boards which store into Apple RAM using DMA transfer will possible give trouble. I don't know for certain because I don't have any.

I also worried about compatibility with QuikLoader. Both QL and SD want to take control of the bus on power up or reset. Both substitute their own firmware for whatever is plugged into the mother board. Sure enough, when I tried them both in the same machine they did not work. On power up both cpu's began to operate. SD drew its hi-res graphic logo, and then died. QL died too. Take either card out, and all is well.

Speaking of firmware, I should mention that there is a 2716 with 2K of firmware on the SpeedDemon. When you power up or hit ctrl-RESET the firmware on the card takes control. It sets a bunch of //e soft switches, in case it is in a //e, and then looks at the power-up bytes to see whther this is a RESET or power up. (Remember the power up bytes at $3F3 and $3F4? These bytes will be random when you first turn on your Apple, but during initialization they are set so that the exclusive-or of the two bytes is $A5.) If SpeedDemon thinks you have pressed ctrl-RESET, it copies a short (21-byte) program from its own ROM down to $1D0 and jumps to it. The program turns off the SpeedDemon ROM (by storing at $C800) and then uses a loop to make sure the cache doesn't contain misleading information (I call this action TRASHING the CACHE). Then it jumps to Apple's normal reset code.

If SpeedDemon thinks it is power-up time, because the "eor" the bytes at $3F3 and $3F4 is not $A5, it trashes the cache and copies a large program down to RAM at $1000 through $17FF. Then it trashes the cache again, clears the text screen, and jumps to $1000. The copied code at $1000 turns off the firmware ROM, clears the hi-res screen, switches on hi-res graphics, and draws the SpeedDemon logo. This all takes about two seconds. Then it reads the keyboard to see whether you have typed an ESCAPE, a "1", or a "T". ESCAPE signals SpeedDemon you want to run at normal Apple speed, so it shuts itself off. The other codes cause self-testing code to be executed.

I had a lot of fun figuring out the firmware. It so happens they purposely arranged all the bits in the EPROM in reverse order, so that I had to write a program to flip the bytes around before disassembling the code. I guess it was an attempt to frustrate reverse engineering. I think they should have re-arranged the address lines too, if they really are worried about it.

If all the above makes you want to rush right out and buy one, the price is $295 from Microcomputer Technologies (McT), at 1745 21st St., Santa Monica, CA 90404. Their phone number is (213) 829-3641. If you are a member of Call APPLE, they are selling the SpeedDemon card for only $199. The name on the card has been changed to "Mach 3.5", but it is the same as SpeedDemon. Call them at (206) 251-5222. Since the Call APPLE price is as close to wholesale price as we can get, we will not be trying to sell this board at S-C Software.

By the way, Call APPLE's ad contains a warning: "Mach 3.5 is not compatible in speedup mode with Saturn, Legend, Prometheus expansion memory cards with programs that make use of the extra banks on these cards. A compatible version of Mach 3.5 may be specially ordered."

```
==============================================================================
DOCUMENT :AAL-8507:ProDOS:S.DOS.LOAD.txt
==============================================================================

 1000  *SAVE S.DOS.LOAD
 1010  *--------------------------------
 1020         .OR $7400
 1030         .TF DOS.LOAD
 1040  *--------------------------------
 1050  PNTR          .EQ $00,01
 1060  CAT.INDEX     .EQ $02
 1070  MENU.LETTER   .EQ $03
 1080  LINE.COUNT    .EQ $04
 1090  TRACK         .EQ $05
 1100  SECTOR        .EQ $06
 1110  DONE.FLAG     .EQ $07
 1120  SIZE          .EQ $08,09
 1130  LIMIT         .EQ $0A
 1140  *--------------------------------
 1150  LOMEM         .EQ $67,68
 1160  HIMEM         .EQ $73,74
 1170  PP            .EQ $CA,CB
 1180  *--------------------------------
 1190  BLOCK.BUFFER  .EQ $7C00
 1200  TS.LIST       .EQ $7E00
 1210  *--------------------------------
 1220  MON.RDKEY     .EQ $FD0C
 1230  MON.CROUT     .EQ $FD8E
 1240  MON.PRHEX     .EQ $FDDA
 1250  MON.COUT      .EQ $FDED
 1260  *--------------------------------
 1270  DOS.LOAD
 1280         LDY #EM3     "SLOT:"
 1290         LDA #"8"     1...7
 1300         JSR GETNUM   00000SSS
 1310         LSR          000000SS S
 1320         ROR          S000000S S
 1330         ROR          SS000000 S
 1340         ROR          SSS00000
 1350         STA UNIT
 1360         LDY #EM4     "DRIVE:"
 1370         LDA #"3"     1...2
 1380         JSR GETNUM
 1390         LSR
 1400         LSR
 1410         ROR UNIT     DSSS0000
 1420  *--------------------------------
 1430  LOAD.MENU
 1440         JSR SETUP.SCREEN
 1450         LDA #17           TRACK 17
 1460         LDX #0            SECTOR 0
 1470         STX DONE.FLAG
 1480         STX PNTR
```

```
1490        JSR RTS           READ DOS 3.3 VTOC
1500        STX PNTR+1        SET POINTER
1510 *--------------------------------
1520 NEXT.CAT.SECTOR
1530        JSR READ.NEXT.CATALOG.SECTOR
1540        BCS END.OF.CATALOG
1550        LDY #$0B
1560 *--------------------------------
1570 NEXT.FILE.NAME
1580        STY CAT.INDEX
1590        LDA (PNTR),Y      TRACK
1600        BEQ END.OF.CATALOG
1610        BMI GET.NEXT.FILE ...DELETED FILE
1620        STA TRACK
1630        INY
1640        LDA (PNTR),Y
1650        STA SECTOR
1660        INY
1670        LDA (PNTR),Y      FILE TYPE
1680        ASL               INGORE LOCK BIT
1690        CMP #2            MUST BE TYPE I
1700        BNE GET.NEXT.FILE ...NOT I, SKIP OVER IT
1710 *---DISPLAY MENU LINE------------
1720        LDA MENU.LETTER
1730        JSR MON.COUT      DISPLAY MENU LETTER,
1740        INC MENU.LETTER
1750        LDA #"-"
1760        JSR MON.COUT      ...TWO DASHES
1770        JSR MON.COUT
1780        LDX #30
1790 .1     INY
1800        LDA (PNTR),Y
1810        ORA #$80
1820        JSR MON.COUT      ...AND FILENAME
1830        DEX
1840        BNE .1
1850        JSR MON.CROUT
1860 *---SAVE T/S OF TS-LIST----------
1870        LDA MENU.LETTER
1880        AND #$1F          CONVERT TO INDEX
1890        TAX
1900        DEX               ...SINCE LETTER INC'ED ALREADY
1910        LDA TRACK
1920        STA TRACKS,X
1930        LDA SECTOR
1940        STA SECTORS,X
1950        DEC LINE.COUNT
1960        BEQ MENU.SELECTION  BRANCH IF SCREEN FULL
1970 *--------------------------------
1980 GET.NEXT.FILE
1990        CLC
2000        LDA CAT.INDEX
2010        ADC #35
2020        TAY               BUMP INDEX
```

```
2030        BCC NEXT.FILE.NAME
2040        BCS NEXT.CAT.SECTOR
2050 *---------------------------------
2060 END.OF.CATALOG
2070        LDA #1
2080        STA DONE.FLAG
2090 MENU.SELECTION
2100        LDY #EM0          3-LINE PROMPT
2110        JSR PRINT.MSG
2120 .2     JSR MON.RDKEY
2130        CMP #$E0          LOWER CASE?
2140        BCC .3
2150        AND #$DF          STRIP CASE
2160 .3     CMP #" "          SPACE?
2170        BEQ MENU.NEXT.SCREEN
2180        CMP #$8D          RETURN?
2190        BEQ ABORT
2200        CMP #$9B          ESCAPE?
2210        BEQ ABORT
2220        CMP #"A"
2230        BCC .2            NOT A-Z, SO IGNORE
2240        CMP MENU.LETTER
2250        BCS .2            BEYOND VALID VALUES
2260 *---GET T/S LIST-----------------
2270        AND #$1F          CONVERT LETTER TO INDEX
2280        TAY
2290        LDX SECTORS,Y
2300        LDA TRACKS,Y
2310        JSR RTS           READ TRACK/SECTOR LIST
2320        STX PNTR+1        SET POINTER
2330        LDY #0
2340 .4     LDA (PNTR),Y      MOVE T/S LIST TO ITS BUFFER
2350        STA TS.LIST,Y
2360        INY
2370        BNE .4
2380 *---GET THE FILE SIZE------------
2390        LDY #$0C          POINT AT FIRST T/S
2400        STY CAT.INDEX
2410        LDA TS.LIST,Y     TRACK
2420        BEQ ERR.EMPTY.FILE
2430        LDX TS.LIST+1,Y   SECTOR
2440        JSR RTS           READ FIRST SECTOR
2450        STX PNTR+1
2460        LDY #0
2470        LDA (PNTR),Y      GET FILE SIZE
2480        STA SIZE
2490        INY
2500        LDA (PNTR),Y
2510        STA SIZE+1
2520 *---MAKE ROOM FOR FILE-----------
2530        SEC
2540        LDA HIMEM
2550        SBC SIZE
2560        STA PP            SET ASSEMBLER'S POINTER
```

```
2570        STA LPTR+1        AND OUR LOAD POINTER
2580        LDA HIMEM+1
2590        SBC SIZE+1
2600        STA PP+1
2610        STA LPTR+2
2620        CMP LOMEM+1
2630        BCC ERR.TOO.BIG   ...TOO LOW
2640 *---LOAD FROM 1ST SECTOR---------
2650        INY               POINT AT FIRST PROGRAM BYTE
2660 .5     JSR LOAD.FROM.SECTOR
2670        BCS ABORT         ...END OF LOAD
2680 *---LOAD REST OF FILE------------
2690        LDY CAT.INDEX
2700        INY
2710        INY
2720        BEQ ABORT
2730        STY CAT.INDEX     NEXT TRACK/SECTOR
2740        LDA TS.LIST,Y     TRACK
2750        BEQ ABORT         ...END OF FILE
2760        LDX TS.LIST+1,Y   SECTOR
2770        JSR RTS           READ IT
2780        STX PNTR+1        SET POINTER
2790        LDY #0
2800        BEQ .5            ...ALWAYS
2810 *--------------------------------
2820 ABORT  JMP $8003         WARMSTART ASSEMBLER
2830 *--------------------------------
2840 MENU.NEXT.SCREEN
2850        LDA DONE.FLAG
2860        BEQ .1
2870        JMP LOAD.MENU     START ALL OVER
2880 .1     JSR SETUP.SCREEN
2890        JMP GET.NEXT.FILE
2900 *--------------------------------
2910 ERR.EMPTY.FILE
2920        LDY #EM1
2930        .HS 2C
2940 ERR.TOO.BIG
2950        LDY #EM2
2960        JSR PRINT.MSG
2970        JMP $8003
2980 *--------------------------------
2990 PRINT.MSG
3000 .1     LDA EMS,Y
3010        BEQ .2            00 IS END OF MESSAGE
3020        JSR MON.COUT
3030        INY
3040        BNE .1            ...ALWAYS
3050 .2     RTS
3060 *--------------------------------
3070 GETNUM
3080        STA LIMIT
3090        JSR PRINT.MSG     PROMPT
3100 .1     JSR MON.RDKEY
```

```
3110        CMP #"1"
3120        BCC .1           GO BACK IF TOO SMALL
3130        CMP LIMIT
3140        BCS .1           ...OR TOO LARGE
3150        JSR MON.COUT     ECHO CHARACTER
3160        EOR #"0"         EXTRACT VALUE
3170        RTS
3180 *--------------------------------
3190 READ.NEXT.CATALOG.SECTOR
3200        LDA #$0B         RESTART INDEX
3210        STA CAT.INDEX
3220        SEC              IN CASE NO MORE SECTORS
3230        LDY #2
3240        LDA (PNTR),Y
3250        TAX              SECTOR
3260        DEY
3270        LDA (PNTR),Y     TRACK
3280        BEQ .1           END OF CATALOG
3290        JSR RTS          READ IT
3300        STX PNTR+1       PAGE IN BUFFER
3310        CLC              SIGNAL WE GOT A SECTOR
3320 .1     RTS
3330 *--------------------------------
3340 *   READ TRACK/SECTOR
3350 *      (A)=TRACK, (X)=SECTOR
3360 *      RETURNS (X)=PAGE OF BUFFER CONTAINING SECTOR
3370 *              CARRY SET IF ERROR
3380 *      CLOBBERS (A) AND (Y)
3390 *--------------------------------
3400 RTS
3410        LDY #0
3420        ASL              TRACK*8
3430        ASL
3440        ASL
3450        BCC .1           BLOCK < $100
3460        INY              BLOCK > $0FF
3470 .1     ASL              *2, MAKE ROOM FOR H/L FLAG BIT
3480        ORA BLKTBL,X
3490        ROR              H/L BIT TO CARRY
3500        STA BLOCK
3510        STY BLOCK+1
3520        LDX /BLOCK.BUFFER
3530        BCC .2           LOWER HALF OF BLOCK
3540        INX              UPPER HALF OF BLOCK
3550 .2     JSR $BF00
3560        .DA #$80,PARMLIST
3570        BCS .3           ...ERROR
3580        RTS
3590 .3     PHA              SAVE ERROR CODE
3600        LDY #EM5         "ERROR"
3610        JSR PRINT.MSG
3620        PLA
3630        JSR MON.PRHEX    DISPLAY CODE
3640        JMP $8003        SOFTLY BACK TO S-C MACRO
```

```
3650 *--------------------------------
3660 SETUP.SCREEN
3670        LDA #20          LINES PER SCREEN
3680        STA LINE.COUNT
3690        LDA #"A"         START MENU WITH LETTER "A"
3700        STA MENU.LETTER
3710        JSR MON.CROUT    THREE BLANK LINES
3720        JSR MON.CROUT
3730        JMP MON.CROUT    RETURN THROUGH CROUT
3740 *--------------------------------
3750 *      RETURN .CS. IF END OF LOAD
3760 *--------------------------------
3770 LOAD.FROM.SECTOR
3780        LDA LPTR+1       IS THERE ROOM FOR
3790        CMP HIMEM        ANOTHER BYTE?
3800        LDA LPTR+2
3810        SBC HIMEM+1
3820        BCS LFS2         NO, END OF LOAD
3830        LDA (PNTR),Y
3840 LPTR   STA $5555
3850        INC LPTR+1
3860        BNE .1
3870        INC LPTR+2
3880 .1     INY
3890        BNE LOAD.FROM.SECTOR
3900 LFS2   RTS
3910 *--------------------------------
3920 EMS
3930 EM0    .EQ *-EMS
3940        .HS 8D
3950        .AS -/TYPE LETTER TO LOAD A FILE,/
3960        .HS 8D
3970        .AS -/OR <SPACE> FOR MORE FILES,/
3980        .HS 8D
3990        .AS -/OR <RET> OR <ESC> TO ABORT:  /
4000        .HS 00
4010 EM1    .EQ *-EMS
4020        .HS 8D
4030        .AS -/FILE IS EMPTY/
4040        .HS 00
4050 EM2    .EQ *-EMS
4060        .HS 8D
4070        .AS -/FILE IS TOO BIG/
4080        .HS 00
4090 EM3    .EQ *-EMS
4100        .AS -/ SLOT: /
4110        .HS 00
4120 EM4    .EQ *-EMS
4130        .HS 8D
4140        .AS -/DRIVE: /
4150        .HS 00
4160 EM5    .EQ *-EMS
4170        .HS 8D
4180        .AS -/ERROR /
```

```
4190         .HS 00
4200  *--------------------------------
4210  BLKTBL .HS 00.0E.0D.0C.0B.0A.09.08
4220         .HS 07.06.05.04.03.02.01.0F
4230  *--------------------------------
4240  PARMLIST
4250         .DA #3
4260  UNIT   .HS 60            DRIVE-1*8+SLOT*16
4270         .DA BLOCK.BUFFER
4280  BLOCK  .DA 2
4290  *--------------------------------
4300  TRACKS  .BS 21
4310  SECTORS .BS 21
```

```
==============================================================================
DOCUMENT :AAL-8507:ProDOS:S.RECURCAT.txt
==============================================================================

 1000  *SAVE S.RECURCAT
 1010  *--------------------------------
 1020  MLI    .EQ $BF00
 1030  DEVNUM .EQ $BF30
 1040  BELL   .EQ $FBDD
 1050  CROUT  .EQ $FD8E
 1060  COUT   .EQ $FDED
 1070  PNTR   .EQ $EB AND EC
 1080  *--------------------------------
 1090  CAT
 1100         JSR MLI           GET CURRENT PREFIX
 1110         .DA #$C7,P.PREFIX
 1120         BCS .1            ...ERROR
 1130         JSR MLI           OPEN THE DIRECTORY
 1140         .DA #$C8,P.OPEN     AND READ FIRST BLOCK
 1150         BCS .1            ...ERROR
 1160         LDA DEVNUM        SET UP READ MLI BLOCK
 1170         STA R.DEVNUM
 1180         JSR MLI           CLOSE THE DIRECTORY
 1190         .DA #$CC,P.CLOSE
 1200         BCC .2            ...NO ERROR
 1210  .1     JSR BELL          INDICATE ERROR
 1220         RTS
 1230  .2     LDA #0       BUFFERS ON PAGE BOUNDARIES
 1240         STA NEST.LEVEL    START AT TOP LEVEL
 1250         LDY /BUFFER.ONE   POINT TO NEXT BLOCK
 1260         JSR SETUP.NEXT.BLOCK
 1270  *--------------------------------
 1280  LIST.DIRECTORY
 1290         INC NEST.LEVEL    DROP TO NEXT LEVEL
 1300  *---GET DIR DATA-----------------
 1310         LDY #38
 1320  .1     LDA (PNTR),Y           GET: BYTES.PER.ENTRY....35
 1330         STA BYTES.PER.ENTRY-35,Y    ENTRIES.PER.BLOCK..36
 1340         DEY                         FILE.COUNT......37,38
 1350         CPY #35
 1360         BCS .1
 1370  *---POINT TO FIRST FILE NAME-----
 1380         LDA #2            SKIP OVER DIR HEADER
 1390         STA CURRENT.ENTRY.NUMBER
 1400         ASL               A=4, CLEAR CARRY
 1410         ADC BYTES.PER.ENTRY
 1420         STA PNTR          POINT AT FIRST NAME
 1430         LDA #0            START FILE COUNT
 1440         STA CURRENT.FILE.COUNT
 1450         STA CURRENT.FILE.COUNT+1
 1460  *---STOP IF NO ACTIVE FILES------
 1470         LDA ACTIVE.FILE.COUNT
 1480         ORA ACTIVE.FILE.COUNT+1
```

```
1490        BNE .2       ...AT LEAST ONE FILE
1500        RTS          ...END OF DIRECTORY
1510 *---PRINT FILE NAME-------------
1520 .2     LDY #0            POINT TO TYPE/LENGTH
1530        LDA (PNTR),Y
1540        BEQ .8            0 = DELETED FILE
1550        AND #$0F          ISOLATE NAME LENGTH
1560        TAX               X = #CHARS IN NAME
1570        LDY NEST.LEVEL    NUMBER OF LEADING BLANKS
1580        LDA #" "
1590 .3     JSR COUT          INDENT BY DIRECTORY LEVEL
1600        DEY
1610        BNE .3
1620        LDA (PNTR),Y      GET TYPE/LENGTH
1630        PHA               1L, 2L, 3L, OR DL
1640        BPL .4            ...NOT DIR FILE
1650        LDA #"/"          DIR FILE, PRINT A SLASH
1660        JSR COUT
1670 .4     INY               PRINT THE FILE'S NAME
1680        LDA (PNTR),Y
1690        ORA #$80
1700        JSR COUT
1710        DEX
1720        BNE .4
1730        JSR CROUT
1740        PLA               GET TYPE/LENGTH AGAIN
1750        BPL .7            ...NOT DIR FILE
1760 *---PUSH DATA ON STACK-----------
1770        LDA PNTR+1        SAVE POINTER IN CURRENT BLOCK
1780        PHA
1790        LDA PNTR
1800        PHA          SAVE:  R.BLOCK
1810        LDX #0              BYTES.PER.ENTRY
1820 .5     LDA PUSH.VARS,X     ENTRIES.PER.BLOCK
1830        PHA                 ACTIVE.FILE.COUNT
1840        INX                 CURRENT.FILE.COUNT
1850        CPX #PUSH.COUNT     CURRENT.ENTRY.NUMBER
1860        BNE .5              NEXT.BLOCK
1870 *---READ HEADER OF SUBDIR--------
1880        LDY #$12          POINT AT KEYBLOCK POINTER
1890        LDA (PNTR),Y      GET HIGH BYTE
1900        TAX
1910        DEY
1920        LDA (PNTR),Y      GET LOW BYTE
1930        JSR READ.NEXT.BLOCK
1940 *---RECURSIVE CALL---------------
1950        JSR LIST.DIRECTORY
1960        DEC NEST.LEVEL    POP TO HIGHER LEVEL
1970 *---POP DATA OFF STACK-----------
1980        LDX #PUSH.COUNT   GET BLOCK OF VARS
1990 .6     PLA
2000        STA PUSH.VARS-1,X
2010        DEX
2020        BNE .6
```

```
2030        PLA
2040        STA PNTR          GET KEYBLOCK POINTER
2050        PLA
2060        STA PNTR+1
2070        CMP /BUFFER.TWO   IS BLOCK IN BUFFER.TWO?
2080        BCC .7            ...NO, DON'T NEED TO READ
2090        JSR MLI           ...YES, MUST READ THE BLOCK
2100        .DA #$80,P.READ
2110 *---COUNT THE FILE---------------
2120 .7     INC CURRENT.FILE.COUNT
2130        BNE .8
2140        INC CURRENT.FILE.COUNT+1
2150 *---SEE IF THAT WAS LAST FILE----
2160 .8     LDA CURRENT.FILE.COUNT
2170        CMP ACTIVE.FILE.COUNT
2180        LDA CURRENT.FILE.COUNT+1
2190        SBC ACTIVE.FILE.COUNT+1
2200        BCC .9            ...NOT LAST FILE
2210        RTS               ...END OF DIRECTORY
2220 *---ADVANCE PNTR TO NEXT ENTRY---
2230 .9     CLC
2240        LDA PNTR          GET RESULT IN Y,X
2250        ADC BYTES.PER.ENTRY
2260        TAX
2270        LDA PNTR+1
2280        ADC #0
2290        TAY
2300 *---ARE WE STILL INSIDE BLOCK?---
2310        LDA CURRENT.ENTRY.NUMBER
2320        INC CURRENT.ENTRY.NUMBER
2330        CMP ENTRIES.PER.BLOCK
2340        BCC .10           ...INSIDE SAME BLOCK
2350 *---READ NEXT BLOCK--------------
2360        LDA NEXT.BLOCK
2370        LDX NEXT.BLOCK+1
2380        JSR READ.NEXT.BLOCK
2390        LDA #1            START WITH FIRST ENTRY
2400        STA CURRENT.ENTRY.NUMBER    IN NEW BLOCK
2410        LDX #4            SKIP OVER BLOCK NUMBERS
2420        LDY /BUFFER.TWO
2430 .10    STX PNTR          NEW PNTR VALUE
2440        STY PNTR+1
2450        JMP .2            ...TO LIST NEXT FILENAME
2460 *--------------------------------
2470 READ.NEXT.BLOCK
2480        STA R.BLOCK       BLOCK # IN X,A
2490        STX R.BLOCK+1
2500        JSR MLI           READ THE BLOCK
2510        .DA #$80,P.READ
2520        LDA #BUFFER.TWO   WE USED BUFFER.TWO
2530        LDY /BUFFER.TWO
2540 SETUP.NEXT.BLOCK
2550        STA PNTR          PNTR FROM Y,A
2560        STY PNTR+1
```

```
2570        LDY #2            GET NEXT BLOCK #
2580        LDA (PNTR),Y
2590        STA NEXT.BLOCK
2600        INY
2610        LDA (PNTR),Y
2620        STA NEXT.BLOCK+1
2630        RTS               RETURN
2640  *---------------------------------
2650  P.PREFIX   .DA #1
2660             .DA BUFFER.TWO
2670  *---------------------------------
2680  P.OPEN     .DA #3
2690             .DA BUFFER.TWO
2700  OPENBUF    .DA BUFFER.ONE
2710             .DA #0
2720  *---------------------------------
2730  P.CLOSE    .DA #1
2740             .DA #0
2750  *---------------------------------
2760  P.READ     .DA #3
2770  R.DEVNUM   .DA #$60
2780             .DA BUFFER.TWO
2790  PUSH.VARS .EQ *
2800  R.BLOCK    .DA 0
2810  *---------------------------------
2820  BYTES.PER.ENTRY          .BS 1
2830  ENTRIES.PER.BLOCK        .BS 1
2840  ACTIVE.FILE.COUNT        .BS 2
2850  CURRENT.FILE.COUNT       .BS 2
2860  CURRENT.ENTRY.NUMBER     .BS 1
2870  NEXT.BLOCK               .BS 2
2880  PUSH.COUNT               .EQ *-PUSH.VARS
2890  *---------------------------------
2900  NEST.LEVEL               .BS 1
2910  *---------------------------------
2920  WASTED .EQ *+255/256*256-*
2930        .BS WASTED
2940  *---------------------------------
2950  BUFFER.ONE  .BS 512
2960  BUFFER.TWO  .BS 512
2970  *---------------------------------
```

=============================================================================
DOCUMENT :AAL-8508:Articles:Conversions.txt
=============================================================================

Generic Conversion Routines................Bob Sander-Cederlof

I may have written hundreds of different versions of the elementary I/O conversion routines. The first few would have been for the IBM 704, back in college days. Then there was the G-15, the 1620, the 3100, the 3300, the 6600, the 1700, the 8090, the 960, the 980, the 990, and so on. Don't worry of those numbers don't mean anything to you. They are the "names" of computers out of the past, not micro chips.

What I am talking about is writing programs which convert input decimal characters representing decimal numbers into internal binary form, and the converse operation of converting binary numbers into decimal form. We have published several variations of both in previous newsletters, but I have some special ones to present here.

There are many variations of the basic routines, and that is one reason I have written so many. Thinking just of the output conversions (binary to decimal):

* Convert to a string in memory, or print it out.
* Number of bytes in binary number.
* Supply leading zeroes or blanks or neither.
* Integer, fraction, floating point, or fixed point.
* Signed or unsigned.

The routine I set out to write today works with unsigned integers, prints out the resulting characters rather than storing them in a string, and does not print any leading zeroes or blanks. I wrote it to work with two-byte values, between 0 adn 65535. As an added feature, I indicated in the comments how to expand it to work with larger values.

Lines 1800-2080 in the listing comprise the output conversion routine. I divide the number by ten, saving the remainder as the least significant digit; the quotient becomes the new number, so I repeat the process until the quotient is zero. Then the digits, which were all saved on the 6502 stack, are popped back off and printed.

Line 1810 starts the digit counter at 0, and line 1950 increments the counter each time a new digit is pushed onto the stack. Lines 2020-2060 pull the digits off the stack and print them in reverse order.

Lines 1970-2000 test the quotient: if it is non-zero, another division is performed; if not, we are ready to print the result. This is one place where you need to add code if your input values are larger than two bytes, as I indicated in line 1980. By the way, since we do one division before testing, an input value of zero will print as "0".

Lines 1830-1930 divide the input value by ten. It may look like I am dividing by five, but remember 5 = 10/2. I did more fiddling than analyzing in this loop, but it really does work. Line 1840 sets the loop count to 16, the number of bits in two bytes. If you want to convert three-byte values, change the 16 to 24. The loop needs to be executed once for each bit in the input value. If you are going to have values longer than two bytes, you also need to add more ROL instructions between lines 1880 and 1900, as indicated in my comment line 1890. If you were to need a three byte conversion routine, you could just remove the "*--" from the front of lines 1890 and 1980, and chane line 1840 to LDY #24.

Notice that this subroutine is very short, and fairly fast. I have an idea that some of you will think of ways to make it shorter and faster; if you do, try to keep it easily modifiable for the number of bytes in values.

Next I wrote a program to convert from a decimal string into binary, lines 1290-1720. It is also set up for unsigned two-byte integer values, with comments indicating how to modify it for longer values. I have written shorter routines before, but this one makes extension to longer values easy and tests for overflow.

The string is assumed to be in ASCII, with high bits = 1, starting at $0200, and terminated by any non-digit. It just so happens that these are just the conditions you usually find in an Apple, because almost all input routines use the buffer at $0200. Woz started it, and we all followed Woz.

Lines 1300-1330 clear the value, as well as starting the buffer index at zero. The rest of the routine scans through the digits. Each time the current value is multiplied by ten, and the next digit added. If at any point an overflow is detected (a value too large for the number of bytes) the routine rings the bell and quits. You can use some other error indication, and probably should, such as printing "NUMBER TOO LARGE".

In order to multiply by ten, I set aside another storage area equal in length to the value accumulator. At line 1380 the new digit is saved in the Y-register. The accumulated value at this point is in XH and XL. Lines 1390-1480 form the value*4 in SH and XL, leaving the original value in XH and SL. (Yes, they are criss-crossed.) Lines 1410-1420 show how you would extend this portion to longer values.

Lines 1490-1610 add value*4 to value to get value*5, and then double the result to get value*10. Again, lines 1530-1550 show how to extend the value. Lines 1630-1700 add in the new digit, and the comments show how to extend to longer values.

The top level routine in lines 1130-1270 is just a test routine. It calls the monitor line input routine. If you type an empty line, it will stop. Otherwise it calles the input conversion routine, prints the resulting value in hexadecimal, and converts it back to decimal with the output conversion routine.

```
=====================================================================
DOCUMENT :AAL-8508:Articles:Davids.IIc.Buff.txt
=====================================================================
```

Another Auxiliary Memory Program...............David C. Johnson
Applied Engineering

What has 640K of memory and is as cute as a button? My Apple //c! It didn't come with all that memory, "only" 128K of it. Before I even powered it up for the first time, I installed a 512K Z-RAM. Ready to take on Blue's 640K machine? Maybe.

I've had quite a few Apple Computers, my first had Integer ROMs and a serial number in the thirty one thousands, and my current workhorse is an Apple //e with the works. So why a //c? Well, for one it's cute, and secondly its firmware was written by Ernie Beernink and Rich Williams, the same guys that wrote the //e Enhanced ROMs and Extended Debugging Monitor. These guys write slick code. Finally, I can type control-reset with one hand.

Well, what to do after getting it home? I tried my mouse out on it, but moved it back to the //e. My paddles and joysticks all have 16-pin plugs, so I couldn't use them. I don't have an RGB interface for the //c yet, so the color monitor has to stay put. That leaves my Imagewriter printer to play with.

Having two computers and only one printer is an old problem. One usually solved with a rotary switch. I figured that I could do a little better. What I did is connect the Imagewriter to the //c's Printer port, and the //e's Super Serial Card (SSC) to the //c's Modem port. I then wrote the program that follows this article. It implements a 576K buffer for the //e, in the //c. Now I can use the printer from the //c just by typing pr#1. When I want to print from the //e, I just boot a disk on the //c, then type pr#1 on the //e. However, the printing, for the //e, goes MUCH faster. I've setup the link between the //e and the //c to transmit at 19200 baud! Assembling a listing of the buffering program takes about 7 seconds (and half of that is writing the target file)!

The SSC is in slot 1, it is configured as follows:

```
     SW1: off off off off off  on  on
     SW2:  on off off  on  on off off
     The jumper block is installed pointing towards modem
```

The Imagewriter's swiches are set:

```
     SW1:   open   open   open   open closed closed open open
     SW2: closed closed   open   open.
```

The pieces are connected with two DIN 5-Pin(m) to DB-25(m) cables, Apple Model Number: A9C0308 (4-2, 2-3, 1-6, 3-7, and 5-20). The cable from the //e to the //c is plugged into a //c System Clock which in turn is plugged into the Modem Port.

The program should work with most any serial printer, and serial card, however, if the serial card cannot "eliminate the modem", you will need a modem-eliminator cable extension, or will have to reverse pins 2 and 3 and pins 6 and 20 of the DB-25 connector.  The Apple cable I used cannot be modified.

While the listing included with this article requires a 512K Applied Engineering Z-RAM board, I have also written versions that work in a 256K Z-RAM and in a stock Apple //c.  More on these versions later. The memory on a Z-RAM is implemented as additional banks of auxiliary memory.  Which of the auxiliary banks is the current auxiliary bank is controlled by a new hardware location at $C073.  The Z-RAM powers-up disabled, that is, with the //c's built-in auxiliary bank as the current auxiliary bank.  The //c powers-up with main memory enabled and all auxiliary memory disabled.  Once selected as the current auxiliary bank, a Z-RAM bank is switched around by all the normal soft switches in the same manner as the //c's built-in auxiliary bank.  A 512K Z-RAM has 8 additional banks and a 256K Z-RAM has 4 more.  Which additional bank is the current auxiliary bank is selected by writing an ODD number between 1 and $F (inclusive) to the bank register at $C073.  The 4 most significant data bits are ignored and any even number (usually zero) selects the //c's built-in auxiliary bank.  A 256K Z-RAM only has bank numbers 3, 7, $B, and $F.  To ease the task of writing programs that display 80 columns of text or double hires graphics, video data is always fetched from the //c's banks, even if a Z-RAM bank is the current auxiliary bank.  Because the Z-RAM plugs into the processor and MMU sockets of the //c, and since only one board may be added this way, the Z-RAM includes a Z-80 processor.  The Z-RAM is also totally compatible with the RamWorks board for the //e.

The //c's serial ports are a lot like Super Serial Cards in slots 1 and 2 of a //e.   The ports and the SSC both use the 6551 ACIA (Asynchronous Communications Interface Adapter) and the firmware is quite similar.  There is one significant difference that I found.  The SSC tells an external source of data to stop transmitting by asserting the Data Terminal Ready bit of the ACIA command register (and thus the DTR pin when the jumper block is in the terminal position), while the //c's ports control the DTR pin with the Request To Send (and transmitter control) bits.  It's right there on page 254 of The Apple //c Reference Manual Volume 1.  Compare this to the schematic on page 100 of the SSC Manual.

Because every //c has a 65C02 processor, I can write code using the new opcodes and it will work in other peoples' machines.  Of course if the code will also work in a //e, I can not be sure that it will be executed on a 65C02.  With the release of the //e enhancement kit, this situation should improve.  65802 opcodes, being new and rare, must be reserved for programs intended for use in a very few machines.

On to the program.  The target file is intended to load at $2000 in main memory. The code from lines 32 to 73 is executed in the $2000 area. This section does all of the setup for what is to come. The D and I flags are cleared and set respectively, ten soft switches are thrown, the screen is cleared, the remainder of the code is copied into ALL auxiliary zero pages and stacks, a text message is written to the screen, and the two ACIAs are initialized. The code copy and message printing share a loop. Lines 66 and 70 cheat a little. The INCs are assembled and the LDA #s are treated as comments. They work because the would-be operands of the LDA #s are one greater than the values just loaded by the previous LDA #s. The 'A' in line 74 is an open-apple MouseText character. The code in aux bank 0 is then entered at label 'Scan'.

The routines 'Write' and 'Read' (lines 79 and 88), handle all access to the buffer. In 'Write', the aux bank is selected, the address within that bank is written into the operand of a store absolute instruction (the copy in the bank just selected), and then the data byte is written. That's a total of four bytes of information passed in internal registers. The data byte had to be passed in the stack pointer! It couldn't have been passed in a memory location because it would have been switched out. 'Read' is a little simpler, it returns a data byte in the Acc. Since I'm using the S-reg for data and the aux bank 0 stack page for code, the program doesn't make any use of regular stack operations. After re-selecting aux bank 0, 'Write' and 'Read' jump back to the code just after the jumps that 'called' them. Even though the $2000 code copied the entire image into every aux bank, only 'Write' and 'Read' are not used as buffer in the Z-RAM banks.

Lines 99 to 108 allocate the (zero page!) variables required to keep track of the buffer. The 'Receive' variables indicate where the next byte received will be buffered, the 'Transmit' variables indicate where the next byte to be printed is buffered, and the 'Byte.Counter' variables keep track of how full (or empty) the buffer is. If the byte counter is zero, then the 'Transmit' variables are equal to the 'Receive' variables and the buffer is empty. 'RTS.Bit' is used to keep track of the //c's 'select' state.

Lines 110 to 128 run an indicator at the top-center of the screen and check to see if you've pressed a key. If you press the space bar, and if the program hasn't asserted the Request To (NOT) Send bit (because the buffer is nearly full), the //e may be halted. This works like a printer's select button.

Lines 129 to 207 handle buffering incoming data. If the Modem ACIA detects any transmission errors, you will see an indication of this at the left end of screen line three. If no character has been received, we go check the Printer port. When a character has been received, we test if the buffer is almost full. If it is, we assert RTS' (another character may already be on the way). The byte counter is incremented. If the buffer is completely full, we tick the third position of screen line one and go check the Printer port. This means that the RTS' handshaking isn't working. You will also get overrun errors. If we have room for the character, we increment the upper left screen position, and load the character from the RxD reg into the stack pointer. We then load the 'Receive' variables, maybe juggle the address high order nibble for the overlapping language card banks, and call 'Write'. Upon return, the 'Receive' variables are advanced through the buffer memory, avoiding our program and invalid aux banks. We then fall into the Printer port code.

Lines 208 to 271 handle printing buffered data as the printer can take it. This code is similar to the code for incoming data. Fewer things can go wrong, we of course test for an empty TxD reg and an empty buffer. We check to see if the buffer is somewhat less than almost full, and may release RTS'. The byte counter is decremented here. When a character is to be printed, we increment the upper right screen position, load the 'Transmit' variables, maybe juggle, call 'Read' and stuff the character into the TxD reg. Upon return, the 'Transmit' variables are advanced (same way), and we loop to 'Scan'. Forever. Reset exits the program.

The program loops VERY quickly. It has to. At 19200 baud, a character is received from the //e every half millisecond and at 9600 baud, a character may be printed every millisecond. The pair of locations at the top center of the screen, that are changed every time around the loop, give a good indication of how fast things are happening. The locations in the upper corners (my //e is to the left of the //c and the printer is to the right) are a good representation of the values of the 'Receive' and 'Transmit' variables. When buffering, the receive indicator races ahead while the transmit indicator lags behind, but since they are both initialized to blanks and the appropriate one is incremented when a character is moved, they come to rest displaying the same character when the buffer is empty.

The symbols 'Z.RAM.Banks.Avail', 'Z.RAM.Banks.Used', 'IIc.Aux.Bank.Avail' and 'BufLen' (lines 94, 96, 273-274) determine the size of the buffer. The ADC immediate operands in lines 195 and 259 cause the buffer to advance from bank 0 to 1 to 3 to 5... to $F. The listing is setup to use a //c's aux bank and a 512K Z-RAM. The changes for a 256K Z-RAM are easy: change the SAVE and .tf filenames (320K), change the 8 in line 96 to a 4, change the 9 in line 274 to a 5, and change the ADC #1s in lines 195 and 259 to ADC #3s. The changes for operating without a Z-RAM are not as simple. I removed all the bank stuff, made the byte counter only 16 bits, and combined the code copy with the screen clear instead of the message printing. It took about 5 minutes. The resulting code just fit into the aux zero page! The source code for all three versions will be on S-C Software's next quarterly disk, and I will send a paper listing of the //c only version to anyone who sends a self addressed stamped envelope to me care of Applied Engineering. I sometimes use the //c only version even though I have a Z-RAM. With the ProDrive disk emulation software, I can lock-out bank 0, leaving it available for double hires or a 64K buffer for my //e. With a 512K Z-RAM, I get a 1024 block /RAM volume.

The program does not use any main memory for the buffer because when you have 576K of aux memory, why bother programming for "only" another 64K? The //c only version, with 64K of buffer memory, is as big or bigger than most buffer boards/boxes.  If anyone writes a 128K main/aux version of the program I would appreciate a copy.

==============================================================
DOCUMENT :AAL-8508:Articles:Front.Page.txt
==============================================================

$1.80

Volume 5 -- Issue 11 August, 1985

In This Issue...



The 65816 continues to make news. We hear of at least two major books on 65816 Assembly Language, which should be in print soon. We also hear that sales of the chip are taking off, with some firms ordering multiplied thousands. Although we have yet to SEE one, we keep hearing reports of plug-in boards for Apples that contain a 65816 and lots of RAM: ComLog, MicroMagic, Checkmate Technology, and others.

Meanwhile, we contemplate the future advantages to just enhancing existing Apples with 65802's and big RAM boards. Applied Engineering or Checkmate will be delighted to stuff 512K additional RAM into your //c. You can add five times that much to your //e with AE's latest version of RAMWorks. Apple's forthcoming Slinky card will add up to a megabyte to any II, II Plus, or //e with a spare slot (1-7). Call APPLE's latest magazine offers the BIG BOARD for slot 0-7 use, one megabyte addressable either in Slinky fashion or with "standard" D000-FFFF mapping, for only $599. If you hurry, they have a special (even lower) price good until Sept 30th.

6800 Cross Assembler for ProDOS

The S-C 6800 Macro Cross Assembler is now also available in a ProDOS version. This is the Version 2.0 level Cross Assembler, including the additional opcodes of the Motorola 6801 and Hitachi 6301 microprocessors. Either the DOS or the ProDOS Version 2.0 Cross Assembler is $50; if you already have one you can add the other for only $20.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

================================================================================
DOCUMENT :AAL-8508:Articles:How.Many.Bytes.txt
================================================================================

How Many Bytes for each Opcode?............Bob Sander-Cederlof

I have been thinking about a semi-automatic object code relocation scheme lately. Steve Wozniak wrote one for the 6502 back in 1976, published in various places such as Call APPLE's "Wozpak". But we are needing one for the 65C02, and maybe for the 65816.

Steve's version used his "Sweet-16" interpreter for some of the address arithmetic. That was okay, because Sweet-16 was in ROM in every Apple in those days. Not so now, although it is available to DOS 3.3 users as part of the Integer BASIC package. But we should write one that does not require Sweet-16.

Steve's relocator also used a ROM-based routine (part of the built-in disassembler) to determine how many bytes are used by each opcode. This routine has been modified in the //c monitor and the new enhanced //e monitor to include the 65C02 opcodes. That's nice, because that means Woz's program will automatically work with 65C02 programs if you run it with the new monitors. However, since I want to include all the 65816 opcodes, I need a new version.

The first step seems to be to write a program which will tell me how many bytes each opcode uses. I know that opcodes which are only one or two bytes do not need any relocation adjustments when a program is moved to a different place in memory. Most 3-byte and all 4-byte instructions contain absolute addresses; if an absolute address is inside the program being moved, it will have to be adjusted for the new location.

I haven't written the entire relocator yet, but I have written a program which will tell me all I need to know about the length of an opcode. My program returns the length in bytes and also two flags. One flag indicates the opcode is a 3-byte instruction which does include an absolute address. The other flag indicates the opcode was an immediate mode instruction. Immediate mode in 65816 code is ambiguous in length, except during execution. My program calls them two-byte instructions, but they may be three bytes each if the status bits so indicate at execution time. I am not sure how my relocator will handle this ambiguity, but for now I am content just to set a flag.

The code in the monitor which determines the length of opcodes uses a table lookup method. I figure that I could do that too, with a 64-byte table, using two bits for each opcode. I would still need a way to test for immediate mode and the special three-byte opcodes which do not have absolute addresses (MVP, MVN, PER, and BRL).

After looking at a chart which showed all the lengths, I decided to do it with bit analysis rather than table lookup. It is probably a little slower, but also a little smaller.

It turns out that almost all of the opcodes whose second hex digit is less than 8 use two bytes. There are only nine exceptions. One interesting case here is BRK, which assembles to only one byte but is considered by the microprocessor to be a two-byte opcode. I am not sure whether the relocator should considere BRK as a single byte or a two-byte opcode, but I think it should probably be one byte.

All opcodes of the with the hex values of $xA, $xA, and $xB are one byte, without exception. All opcodes with the hex values $xC, $xD, and $xE are three bytes with absolute addresses, with only one exception: $5C is a four-byte instruction. All opcodes with value $xF are four bytes each.

The column of opcodes with values $x9 are divided into two groups. Those with the first digit even ($09, 29, 49, etc.) are all three bytes each with absolute addresses. The odd ones are immediate mode opcodes, which may be either two or three bytes each depending on status bits during execution.

Here is a table of the various byte counts, which was actually computed by my program. I printed "2#" for immediate mode opcodes, and "3+" for three-byte opcodes with absolute addresses.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | |
| A | | | | | | | | | | | | | | | | |
| B | | | | | | | | | | | | | | | | |
| C | | | | | | | | | | | | | | | | |
| D | | | | | | | | | | | | | | | | |
| E | | | | | | | | | | | | | | | | |
| F | | | | | | | | | | | | | | | | |

The program which printed the table is in lines 1050-1320 below. The program which computes how many bytes in an opcode follows that. By inserting a "BEQ .6" between lines 1410 and 1420 I could make BRK a one-byte opcode.

My relocator should probably also be on the lookout for calls to ProDOS MLI. This is in effect a six-byte instruction. The first three bytes are $20, $00, $BF (JSR MLI). The fourth byte is the MLI function code.  The last two bytes are the address of a parameter table, and so should be considered as a relocatable address.

I hope to continue to pursue this idea of a relocator, but I make no promises.  Maybe one of you would like to write one and share it with the rest of us.

```
=================================================================
DOCUMENT :AAL-8508:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.......................................................$30
Source Code for DISASM....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051,
   8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
                                  Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains       1980       -         -         -         1
 the source code from     1981       2         3         4         5
 three issues of AAL,     1982       6         7         8         9
 saving you lots of       1983      10        11        12        13
 typing and testing.      1984      14        15        16        17
                          1985      18        19

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Verbatim Diskettes (with hub rings)................ package of 20 for $32
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50
quikLoader EPROM System (SCRG)...............................($179) $170
PROmGRAMER (SCRG)..........................................($149.50) $140
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32

"Apple ProDOS:  Advanced Features for programmers", Little..($17.95)  $17
"Inside the Apple //c", Little..............................($19.95)  $18
"Inside the Apple //e", Little..............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler......................($22.95)  $21
"Understanding the Apple II", Sather........................($22.95)  $21
"Understanding the Apple //e", Sather.......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
```

```
"6502 Assembly Language Programming", Leventhal.............($18.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18
"AppleVisions", Bishop & Grossberger........................($39.95)  $36

 Add $1.50 per book for US shipping.  Foreign orders add postage needed.
      Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                  ***
        *** We accept Master Card, VISA and American Express ***
```

```
==============================================================================
DOCUMENT :AAL-8508:Articles:WildcardMatcher.txt
==============================================================================
```

A Wildcard Filename Search.................Bob Sander-Cederlof

Over the years I have fallen into certain habits when it comes to naming files. I find it convenient to use names starting with "S." for assembly language source files, "B." for binary object code files, and so on. Others like to use suffixes like ".SRC" and ".OBJ" for the same reasons. Some operating systems, like CP/M for example, use suffixes to indicate file type. Others, like ProDOS, let you build sub-directories to categorize your files.

Sometimes I would like to have the ability to do the same operation on a whole group of files. For example, I might want to DELETE all files starting with "B.". Or I might want to copy a whole group of files from one disk to another. If the files happen to have similar names, and if DOS allowed wildcards in filenames, it would be easier.

Some DOS 3.3 programs do have this feature: Apple's FID program, Sensible Software's Super Disk Copy, and others. They have a method for specifying a filename without spelling out the entire name.

The subroutine inside DOS 3.3 which compares a filename you have specified with the names in a catalog is found at $B1F5:

```
       LDY #0
       INX
       INX
 .1    INX
       LDA ($42),Y  Filename you specified
       CMP $B4C6,X  Filename in catalog sector
       BNE ...      ...did not match
       INY
       CPY #30
       BNE .1
   ... matched ...
```

This is a very straightforward string comparison. It requires an exact match of all 30 characters of a filename. There is a similar routine at $A782 which compares a filename you specify with the filenames in the open file buffers.

I wrote a subroutine called MATCH which compares two 30-character strings, allowing wildcards. Unfortunately, it not a simple matter to plug such a subroutine into DOS 3.3, and I have not done that. It is more likely that this subroutine will find its way into some future utility programs.

I also wrote a testing program, so that I could see if my code worked. The program in lines 1110-1380 searches through a list of 30-character strings, printing those which match a key string. To simplify my test program (a good idea to keep testers simple, so they are not themselves more buggy than the testees!) I assembled in the key string and the list of strings to be searched. A slightly better test would allow me to type in the key string.

My MATCH program assumes that the address of the string to be compared with the key is stored at FN and FN+1. Characters in the filename are addressed by "(FN),Y", and in the key are addressed by "KEY,X". MATCH will return with carry set if the filename matches the key, and carry clear if not.

Both the filename and the key are stored "left-justified, blank-filled". That means there may be any number of non-significant blanks on the right end. Lines 1490-1530 scan the current filename from right-to-left, looking for the last non-blank in the name. Lines 1550-1590 do the same for the key. If there is any chance either filename or key could be completely blank, an extra line "BMI ERROR" should be inserted at 1505 and 1565.

I save the index to the right end of the key in KEY.START. Because the end of the filename and key strings is variable, I actually do the comparison from right to left. This makes the "end" actually the beginning.

Line 1610 could be "JMP .4" or "BNE .4", because the object is to get to line 1660. However, the "INX" allows me to fall through lines 1630-1640 and it takes only one byte rather than two or three.

The comparison begins at line 1660. Remember we are scanning backwards, from right to left. Lines 1660-1670 save the two string pointers. Line 1680 gets the next character from the key. If it is a wildcard, I branch back to line 1630. Note that all that happens is that the wildcard is skipped over!

If the key character is not a wildcard, it gets compared to the next character of the filename at line 1710. If it matches, lins 1730-1760 advance both pointers and the comparison continues. These lines also check to see if we have come to the left end of the key or of the filename.

If we are at the end of the filename, lines 1770-1820 check the rest of the key. If there are any characters left in the key which are not wildcards, then the current filename does not match. Otherwise, it does match. Lines 1830-1880 set the appropriate carry status and return.

If we are at the end of the key, lines 1900-1910 check whether we are also at the end of the filename. If so, the filename matched. If not, maybe it did not match. I say maybe, because if there was a wildcard, we might come out with a match if we widen the amount matched by that wildcard. Lines 1920-1990 will handle that possibility.

Two conditions bring us to line 1930. Either a character in the key did not match the current character in the filename, or there are unmatched filename characters left over after the end of the key. In either case, if there has been no wildcard in the key (so far), then the filename does not match the key. If there has been a wildcard, we can try again to match from the most recent wildcard on. We can tell whether or not there has been a wildcard so far by comparing KEY.PNTR with KEY.START. If they are the same, there has been no wildcard. Lines 1920-1990 handle all these details.

I made the wild card character itself a variable, so that you could change it by program control. Since "=" is a valid character in a filename, you might want to use something else.

With this kind of MATCH subroutine, a key of "=.OBJ" would match all names ending with ".OBJ"; "S.=" would match all names starting with "S."; "=A=B=" would match all names containing "A" followed by "B".

You can see the similarity between MATCH and a global search capability such as you might find in a word processor, or in the S-C Macro Assembler. The FIND and REPLACE commands in S-C Macro allow wildcards. However, MATCH differs in that it anchors the key to the beginning and end of the file name (unless you specify a wildcard in those positions).

If string comparisons of this type intrigue you, the book "Software Tools" develops an extremely powerful one in chapter 5. "Software Tools" is a classic book by Kernighan and Plauger, available at many bookstores. (A "classic" in computer books is one still in print after five years; this one qualifies, since it was originally published in 1976.) Their string match routine allows single- and multi-character wildcards, semi-wildcards that match subsets of characters, control of anchoring, and more. It would be a worthwhile exercise to try implementing their algorithm in 6502 language.

```
==============================================================================
DOCUMENT :AAL-8508:DOS3.3:S.Byte.Table.txt
==============================================================================

 1000  *SAVE S.BYTE TABLE
 1010  *--------------------------------
 1020  COUT   .EQ $FDED
 1030  CROUT  .EQ $FD8E
 1040  *--------------------------------
 1050  T
 1060         LDX #0
 1070  .1     TXA
 1080         AND #$0F
 1090         BNE .2
 1100         JSR CROUT
 1110  .2     TXA
 1120         JSR GET.LENGTH.OF.OPCODE
 1130         PHA
 1140         AND #$07
 1150         ORA #"0"
 1160         JSR COUT
 1170         PLA
 1180         ASL          POSITION XY FOR INDEX
 1190         ROL
 1200         ROL
 1210         AND #$03     0000 00XY
 1220         TAY
 1230         LDA TABLE,Y
 1240         JSR COUT
 1250         LDA #" "
 1260         JSR COUT
 1270         INX
 1280         BNE .1
 1290         JMP CROUT
 1300  *--------------------------------
 1310  TABLE  .AS -/ #+/
 1320  *--------------------------------
 1330  *   CALL WITH (A)= OPCODE
 1340  *   RETURN WITH (Y)= OPCODE
 1350  *               (A)= XY000LLL
 1360  *                   LLL = # OF BYTES, 1...4
 1370  *                    X  = 1 IF ABS ADDRESS
 1380  *                    Y  = 1 IF IMMEDIATE
 1390  *--------------------------------
 1400  GET.LENGTH.OF.OPCODE
 1410         TAY
 1420         AND #$0F
 1430         CMP #$08
 1440         BCC .4       XXXX 0XXX
 1450         CMP #$0C
 1460         BCC .3       XXXX 10XX
 1470         CMP #$0F
 1480         BEQ .2       XXXX 1111, L=4
```

```
1490        CPY #$5C
1500        BEQ .2       0101 1100, L=4
1510 *---L=3, ABS ADDRESS-------------
1520 .1     LDA #$83
1530        RTS
1540 *---L=4-------------------------
1550 .2     LDA #4
1560        RTS
1570 *---XXXX 10XX--------------------
1580 .3     CMP #$09
1590        BNE .6       X8, XA, or XB
1600 *---XXXX 1001--------------------
1610        TYA
1620        AND #$10
1630        BNE .1       XXX1 1001, L=3
1640 *---XXX0 1001, IMMEDIATES, L=2---
1650        LDA #$42     OR 3 IF ## MODE
1660        RTS
1670 *---XXXX 0XXX--------------------
1680 .4     LSR          CHECK ODD/EVEN
1690        BCS .5       ODD, L=2
1700        CPY #$22
1710        BEQ .2       JSL LABS, L=4
1720        CPY #$20
1730        BEQ .1       JSR ABS, L=3
1740        CPY #$40
1750        BEQ .6       RTI, L=1
1760        CPY #$60
1770        BEQ .6       RTS, L=1
1780        CPY #$62
1790        BEQ .7       PER LREL, L=3
1800        CPY #$82
1810        BEQ .7       BRL LREL, L=3
1820        CPY #$44
1830        BEQ .7       MVP, L=3
1840        CPY #$54
1850        BEQ .7       MVN, L=3
1860        CPY #$F4
1870        BEQ .1       PEA ABS, L=3
1880 *---L=2-------------------------
1890 .5     LDA #2       L=2
1900        RTS
1910 *---L=1-------------------------
1920 .6     LDA #1
1930        RTS
1940 *---L=3, NON-ABS ADDRESS---------
1950 .7     LDA #3
1960        RTS
1970 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8508:DOS3.3:S.WILDCARD.txt
==============================================================================

 1000  *SAVE S.WILDCARD
 1010  *---------------------------------
 1020  COUT   .EQ $FDED
 1030  CROUT  .EQ $FD8E
 1040  *---------------------------------
 1050  KEY.PNTR   .EQ $00
 1060  BUF.PNTR   .EQ $01
 1070  FN         .EQ $02,03
 1080  KEY.START  .EQ $04
 1090  CNTR       .EQ $05
 1100  *---------------------------------
 1110  T
 1120         LDA #NAME.CNT
 1130         STA CNTR
 1140         LDA #FNLIST
 1150         LDY /FNLIST
 1160  .1     STA FN
 1170         STY FN+1
 1180         JSR MATCH
 1190         BCC .2       ...DID NOT MATCH
 1200         JSR DISPLAY
 1210  .2     LDA FN
 1220         CLC
 1230         ADC #30
 1240         LDY FN+1
 1250         BCC .3
 1260         INY
 1270  .3     DEC CNTR
 1280         BNE .1
 1290         RTS
 1300  *---------------------------------
 1310  DISPLAY
 1320         LDY #0
 1330  .1     LDA (FN),Y
 1340         JSR COUT
 1350         INY
 1360         CPY #30
 1370         BCC .1
 1380         JMP CROUT
 1390  *---------------------------------
 1400  *   COMPARE KEY TO A FILE NAME
 1410  *      KEY MAY CONTAIN WILDCARDS
 1420  *      TRAILING BLANKS DON'T COUNT
 1430  *      FILE NAME ADDRESSED VIA "(FN),Y"
 1440  *      KEY ADDRESSED VIA "KEY,X"
 1450  *      KEY AND FILE NAME ARE UP TO 30 CHARS LONG
 1460  *          (STORED LEFT-JUSTIFIED, BLANK-FILLED)
 1470  *---------------------------------
 1480  MATCH
```

```
       LDY #30      FIND LAST NON-BLANK CHAR
.1     DEY              IN FILE NAME
       LDA (FN),Y
       CMP #" "
       BEQ .1
*--------------------------------
       LDX #30      FIND LAST NON-BLANK CHAR
.2     DEX              IN KEY
       LDA KEY,X
       CMP #" "
       BEQ .2
       STX KEY.START
       INX
*---WILD CARD--------------------
.3     DEX          ADVANCE KEY POINTER
       BMI .8       ...END OF KEY IS WILD, SO MATCHES
*--------------------------------
.4     STX KEY.PNTR
.5     STY BUF.PNTR
.6     LDA KEY,X
       CMP WILD.CARD
       BEQ .3      ...WILD CARD CHARACTER
       CMP (FN),Y
       BNE .11      ...NO MATCH
       DEX
       BMI .10       ...END OF KEY
       DEY
       BPL .6       ...STILL MORE TO COMPARE
*---END OF FILE NAME, MORE KEY---
.7     LDA KEY,X
       CMP WILD.CARD
       BNE .9       ...REST OF KEY NOT WILD, NO MATCH
       DEX
       BPL .7
*---VALID MATCH------------------
.8     SEC          SIGNAL MATCH
       RTS
*---NOT A MATCH------------------
.9     CLC
       RTS
*---END OF KEY-------------------
.10    DEY          MATCH IF END OF NAME
       BMI .8       ...END OF NAME
*---IF AFTER WILDCARD, SLIP------
.11    LDX KEY.PNTR START KEY OVER AGAIN
       CPX KEY.START
       BEQ .9       ...NOT AFTER A WILDCARD
       LDY BUF.PNTR SLIP TO LEFT IN BUFFER
       DEY
       BPL .5       TRY AGAIN
       BMI .7       REST OF KEY BETTER BE WILD
*--------------------------------
WILD.CARD  .AS -/=/
*--------------------------------
```

```
2030  KEY        .AS -/A=                            /
2040  *--------------------------------
2050  FNLIST     .AS -/A SIMPLE KEY                  /
2060             .AS -/NOT SUCH A SIMPLE KEY         /
2070             .AS -/NOT A SIMPLE KEY AT ALL       /
2080             .AS -/A SIMPLE KEY AFTER ALL        /
2090  NAME.CNT   .EQ *-FNLIST/30
2100  *--------------------------------
```

==============================================================
DOCUMENT :AAL-8508:ProDOS:BUF.320K.txt
==============================================================

```
 .ti 81,BNApple //c w/256K Z-RAM buffering program 2.0 8/5/85 dcj
;SAVE Buf.320K
;-------------------------------
; Dedicated to Allan B. Calhamer.
;-------------------------------
Printer.ACIA.TxD     .eq $C098 (w)
Printer.ACIA.Status  .eq $C099 (r)
Printer.ACIA.Command .eq $C09A (r/w)
Printer.ACIA.Control .eq $C09B (r/w)
Modem.ACIA.RxD       .eq $C0A8 (r)
Modem.ACIA.Status    .eq $C0A9 (r)
Modem.ACIA.Command   .eq $C0AA (r/w)
Modem.ACIA.Control   .eq $C0AB (r/w)
Z.RAM.Bank.Reg       .eq $C073 (w) same as RamWorks
Keyboard             .eq $C000 (r)
Store80              .eq $C001 (w) on
RAMRd                .eq $C003 (w) aux
RAMWrt               .eq $C005 (w) aux
AltZP                .eq $C009 (w) aux
Vid40                .eq $C00C (w)
SetAltChr            .eq $C00F (w) w/MouseText
Clear.Key.Strobe     .eq $C010 (r)
Text                 .eq $C051 (r)
Page1                .eq $C054 (r) main
Page2                .eq $C055 (r) aux
Hires                .eq $C057 (r) $2000-$3FFF too...
LCRAM2               .eq $C083 (r/w; write doesn't
LCRAM1               .eq $C08B  change write enable)
;-------------------------------
       .op 65C02
       .or $2000
       .tf /IIc.buf/Bufit320K/
dcj    CLD                 rqd (now)
       SEI                 close this can of worms...
       LDA LCRAM2          1x...switches setup
       LDA Text
       LDA Page1
       LDA Hires
       STZ Store80
       STZ RAMRd
       STZ RAMWrt
       STZ AltZP
       STZ SetAltChr
       STZ Vid40
       LDA #" "            clear 40 column screen
       LDX #0
.1     STA $400,X
       STA $500,X
```

```
       STA $600,X
       STA $700,X
       INX
       BNE .1
       LDY #$0F          install Image in aux ZPs/Stacks
.2     STY Z.RAM.Bank.Reg
.3     LDA Image,X
       STA $00,X
       LDA Image+$100,X
       STA $100,X
       INX
       BNE .3
       LDA Msg,Y         put up a message
       STA $50C,Y
       DEY
       BPL .2
       LDA #%000.0.10.1.0         bop ACIAs
       STA Printer.ACIA.Command
  inc  LDA #%000.0.10.1.1         RTS' lo
       STA Modem.ACIA.Command
       LDA #%0.00.1.1110          9600 baud
       STA Printer.ACIA.Control
  inc  LDA #%0.00.1.1111          19200 baud!
       STA Modem.ACIA.Control
       LDA Modem.ACIA.RxD
       JMP Scan          go 2 it
Msg    .AS 'A'           as in Apple
       .AS -" //c buffer pgm"
Image  .ph $00
; aux bank specified by Acc, bank adr lo by X-reg,
;  bank adr hi by Y-reg, and byte passed in S-reg!
Write  STA Z.RAM.Bank.Reg  bank in Z-RAM
       STX <.1+1         modify STX operand in "this" bank
       STY <.1+2
       TSX               get byte to a usable reg!
.1     STX $FFFF         abs adr modified for each write
       STZ Z.RAM.Bank.Reg  revert to //c aux bank
       JMP W.Ret
; aux bank specified by Acc, bank adr lo by X-reg,
;  bank adr hi by Y-reg, and byte returned in Acc.
Read   STA Z.RAM.Bank.Reg  bank in Z-RAM
       STX <.1+1         modify LDA operand in "this" bank
       STY <.1+2
.1     LDA $FFFF         abs adr modified for each read
       STZ Z.RAM.Bank.Reg  revert to //c aux bank
       JMP R.Ret
Z.RAM.Banks.Avail   .eq *-3
; (-3 because JMP R.Ret never executed in Z-RAM)
Z.RAM.used          .eq Z.RAM.Banks.Avail*4
;---------------------------------
; buffer starts at first available location in //c aux bank
Receive.Adr.Lo      .da #IIc.Aux.Bank.Avail
Receive.Adr.Hi      .da /IIc.Aux.Bank.Avail
Receive.Bank        .da #$00
```

```
Transmit.Adr.Lo     .da #IIc.Aux.Bank.Avail
Transmit.Adr.Hi     .da /IIc.Aux.Bank.Avail
Transmit.Bank       .da #$00
Byte.Counter.Lo     .da #$000000        indicates empty
Byte.Counter.Mid    .da #$000000/256
Byte.Counter.Hi     .da #$000000/65536
RTS.Bit             .da #%000.0.10.0.0  RTS' lo
;--------------------------------
Scan    LDA Page1         access main text screen
        INC $413          show we're alive
        DEC $414
        LDA Page2         back to aux
        LDA Keyboard      scan keyboard
        BPL Scan.Modem.Port
        CMP #" "          space toggles RTS' (DTR2B) to //e
        BNE .2
        LDA Modem.ACIA.Command
        AND #%000.0.10.0.0
        BNE .1            =>It's ok, you can turn it off...
        LDA RTS.Bit
        BNE Scan.Modem.Port  =>don't do it! (yet)
.1      LDA Modem.ACIA.Command
        EOR #%000.0.10.0.0
        STA Modem.ACIA.Command
        AND #%000.0.10.0.0
        STA RTS.Bit
.2      BIT Clear.Key.Strobe
Scan.Modem.Port
        LDY Modem.ACIA.Status
        TYA
        AND #%0000.0111   error bits mask
        BEQ .1            =>error-free operation
        TAX
        LDA Page1         access main text screen
        INC $4FF,X        indicate error...
        LDA Page2         back to aux
.1      TYA
        AND #%0000.1000   receive data reg full mask
        BEQ CantRx        =>not full
        LDA Byte.Counter.Lo   received a byte,
        LDX Byte.Counter.Mid  do we assert RTS' ?
        LDY Byte.Counter.Hi
        CMP #BufLen-256
        BNE .2            =>buffer not @ full-256
        CPX /BufLen-256
        BNE .2            =>buffer not @ full-256
        CPY ^BufLen-256
        BNE .2            =>buffer not @ full-256
        LDA #%000.0.10.0.0   assert RTS'
        TRB Modem.ACIA.Command
        LDA Byte.Counter.Lo  reload it
.2      INC               fig next byte count
        BNE .3
        INX
```

```
       BNE .3
       INY
.3     CMP #BufLen       do we have room for it ?
       BNE Room          =>buffer not full
       CPX /BufLen
       BNE Room          =>buffer not full
       CPY ^BufLen
       BNE Room          =>buffer not full
       LDA Page1         access main text screen
       INC $402          indicate full
       LDA Page2         back to aux
CantRx BRA Cant.Receive  =>buffer is full!
Room   STA Byte.Counter.Lo
       STX Byte.Counter.Mid
       STY Byte.Counter.Hi
       LDA Page1         access main text screen
       INC $400          show we received a byte
       LDA Page2         back to aux
       LDX Modem.ACIA.RxD
       TXS               pass it in S-reg
       LDX Receive.Adr.Lo
       LDY Receive.Adr.Hi
       BIT LCRAM2        normally use LC bank 2
       TYA
       AND #$F0
       CMP /$C000        if adr is in $CXXX range
       BNE .1
       BIT LCRAM1         use LC bank 1
       TYA
       ORA /$D000
       TAY
.1     LDA Receive.Bank
       JMP Write
W.Ret  INC Receive.Adr.Lo  fig next receive adr
       BNE Scan.Printer.Port
       INC Receive.Adr.Hi
       BNE Scan.Printer.Port
       LDA Receive.Bank
       CMP #1
       ADC #3            clear carry if 0, else set it
       CMP #$10
       BCC .1            =>entering/still in Z-RAM
       LDA #$00          wrap to //c bank 0
       LDX #IIc.Aux.Bank.Avail
       LDY /IIc.Aux.Bank.Avail
       BRA .2
.1     LDX #Z.RAM.Banks.Avail
       LDY /Z.RAM.Banks.Avail
.2     STA Receive.Bank
       STX Receive.Adr.Lo
       STY Receive.Adr.Hi
Cant.Receive
Scan.Printer.Port
       LDA #%0011.0000   make transmit data reg empty and
```

```
        AND Printer.ACIA.Status  Data Carrier Detect mask
        CMP #%0001.0000   test empty and DCD' lo
        BNE Cant.Transmit =>not empty or not ready
        LDA Byte.Counter.Lo   printer can take another byte,
        ORA Byte.Counter.Mid  do we have one ?
        ORA Byte.Counter.Hi
        BEQ Cant.Transmit     =>buffer is empty!!!
        LDA Byte.Counter.Lo   do we release RTS' ?
        LDX Byte.Counter.Mid
        LDY Byte.Counter.Hi
        CMP #BufLen-2048
        BNE .1            =>buffer not @ full-2048
        CPX /BufLen-2048
        BNE .1            =>buffer not @ full-2048
        CPY ^BufLen-2048
        BNE .1            =>buffer not @ full-2048
        LDA RTS.Bit
        TSB Modem.ACIA.Command  release RTS' (maybe)
.1      STA Z.RAM.Bank.Reg+5
        LDA Byte.Counter.Lo     fig next byte count
        BNE .3
        LDA Byte.Counter.Mid
        BNE .2
        DEC Byte.Counter.Hi
.2      DEC Byte.Counter.Mid
.3      DEC Byte.Counter.Lo
        LDA Page1         access main text page
        INC $427          show we printed a byte
        LDA Page2         back to aux
        LDX Transmit.Adr.Lo
        LDY Transmit.Adr.Hi
        BIT LCRAM2        normally use LC bank 2
        TYA
        AND #$F0
        CMP /$C000        if adr in $CXXX range
        BNE .4
        BIT LCRAM1         use LC bank 1
        TYA
        ORA /$D000
        TAY
.4      LDA Transmit.Bank
        JMP Read
R.Ret   STA Printer.ACIA.TxD
        INC Transmit.Adr.Lo  fig next transmit adr
        BNE Next
        INC Transmit.Adr.Hi
        BNE Next
        LDA Transmit.Bank
        CMP #1            clear carry if 0, else set it
        ADC #3
        CMP #$10
        BCC .1            =>entering/still in Z-RAM
        LDA #$00          wrap to //c bank 0
        LDX #IIc.Aux.Bank.Avail
```

```
       LDY /IIc.Aux.Bank.Avail
       BRA .2
.1     LDX #Z.RAM.Banks.Avail
       LDY /Z.RAM.Banks.Avail
.2     STA Transmit.Bank
       STX Transmit.Adr.Lo
       STY Transmit.Adr.Hi
Cant.Transmit
Next   JMP Scan
IIc.Aux.Bank.Avail .eq *
BufLen .eq $50000-Z.RAM.Used-IIc.Aux.Bank.Avail
 .lif
```

==========================================================================
DOCUMENT :AAL-8508:ProDOS:BUF.576K.txt
==========================================================================

```
 .ti 81,BNApple //c w/512K Z-RAM buffering program 2.0 8/5/85 dcj
;SAVE Buf.576K
;-------------------------------
; Dedicated to Allan B. Calhamer.
;-------------------------------
Printer.ACIA.TxD     .eq $C098 (w)
Printer.ACIA.Status  .eq $C099 (r)
Printer.ACIA.Command .eq $C09A (r/w)
Printer.ACIA.Control .eq $C09B (r/w)
Modem.ACIA.RxD       .eq $C0A8 (r)
Modem.ACIA.Status    .eq $C0A9 (r)
Modem.ACIA.Command   .eq $C0AA (r/w)
Modem.ACIA.Control   .eq $C0AB (r/w)
Z.RAM.Bank.Reg       .eq $C073 (w) same as RamWorks
Keyboard             .eq $C000 (r)
Store80              .eq $C001 (w) on
RAMRd                .eq $C003 (w) aux
RAMWrt               .eq $C005 (w) aux
AltZP                .eq $C009 (w) aux
Vid40                .eq $C00C (w)
SetAltChr            .eq $C00F (w) w/MouseText
Clear.Key.Strobe     .eq $C010 (r)
Text                 .eq $C051 (r)
Page1                .eq $C054 (r) main
Page2                .eq $C055 (r) aux
Hires                .eq $C057 (r) $2000-$3FFF too...
LCRAM2               .eq $C083 (r/w; write doesn't
LCRAM1               .eq $C08B  change write enable)
;-------------------------------
       .op 65C02
       .or $2000
       .tf Bufit576K
dcj    CLD                 rqd (now)
       SEI                 close this can of worms...
       LDA LCRAM2          1x...switches setup
       LDA Text
       LDA Page1
       LDA Hires
       STZ Store80
       STZ RAMRd
       STZ RAMWrt
       STZ AltZP
       STZ SetAltChr
       STZ Vid40
       LDA #" "            clear 40 column screen
       LDX #0
.1     STA $400,X
       STA $500,X
```

```
       STA $600,X
       STA $700,X
       INX
       BNE .1
       LDY #$0F          install Image in aux ZPs/Stacks
.2     STY Z.RAM.Bank.Reg
.3     LDA Image,X
       STA $00,X
       LDA Image+$100,X
       STA $100,X
       INX
       BNE .3
       LDA Msg,Y         put up a message
       STA $50C,Y
       DEY
       BPL .2
       LDA #%000.0.10.1.0         bop ACIAs
       STA Printer.ACIA.Command
  inc  LDA #%000.0.10.1.1         RTS' lo
       STA Modem.ACIA.Command
       LDA #%0.00.1.1110          9600 baud
       STA Printer.ACIA.Control
  inc  LDA #%0.00.1.1111          19200 baud!
       STA Modem.ACIA.Control
       LDA Modem.ACIA.RxD
       JMP Scan          go 2 it
Msg    .AS 'A'           as in Apple
       .AS -" //c buffer pgm"
Image  .ph $00
; aux bank specified by Acc, bank adr lo by X-reg,
;  bank adr hi by Y-reg, and byte passed in S-reg!
Write  STA Z.RAM.Bank.Reg  bank in Z-RAM
       STX <.1+1         modify STX operand in "this" bank
       STY <.1+2
       TSX               get byte to a usable reg!
.1     STX $FFFF         abs adr modified for each write
       STZ Z.RAM.Bank.Reg  revert to //c aux bank
       JMP W.Ret
; aux bank specified by Acc, bank adr lo by X-reg,
;  bank adr hi by Y-reg, and byte returned in Acc.
Read   STA Z.RAM.Bank.Reg  bank in Z-RAM
       STX <.1+1         modify LDA operand in "this" bank
       STY <.1+2
.1     LDA $FFFF         abs adr modified for each read
       STZ Z.RAM.Bank.Reg  revert to //c aux bank
       JMP R.Ret
Z.RAM.Banks.Avail   .eq *-3
; (-3 because JMP R.Ret never executed in Z-RAM)
Z.RAM.used          .eq Z.RAM.Banks.Avail*8
;--------------------------------
; buffer starts at first available location in //c aux bank
Receive.Adr.Lo      .da #IIc.Aux.Bank.Avail
Receive.Adr.Hi      .da /IIc.Aux.Bank.Avail
Receive.Bank        .da #$00
```

```
Transmit.Adr.Lo     .da #IIc.Aux.Bank.Avail
Transmit.Adr.Hi     .da /IIc.Aux.Bank.Avail
Transmit.Bank       .da #$00
Byte.Counter.Lo     .da #$000000        indicates empty
Byte.Counter.Mid    .da #$000000/256
Byte.Counter.Hi     .da #$000000/65536
RTS.Bit             .da #%000.0.10.0.0  RTS' lo
;---------------------------------
Scan    LDA Page1         access main text screen
        INC $413          show we're alive
        DEC $414
        LDA Page2         back to aux
        LDA Keyboard      scan keyboard
        BPL Scan.Modem.Port
        CMP #" "          space toggles RTS' (DTR2B) to //e
        BNE .2
        LDA Modem.ACIA.Command
        AND #%000.0.10.0.0
        BNE .1            =>It's ok, you can turn it off...
        LDA RTS.Bit
        BNE Scan.Modem.Port  =>don't do it! (yet)
.1      LDA Modem.ACIA.Command
        EOR #%000.0.10.0.0
        STA Modem.ACIA.Command
        AND #%000.0.10.0.0
        STA RTS.Bit
.2      BIT Clear.Key.Strobe
Scan.Modem.Port
        LDY Modem.ACIA.Status
        TYA
        AND #%0000.0111   error bits mask
        BEQ .1            =>error-free operation
        TAX
        LDA Page1         access main text screen
        INC $4FF,X        indicate error...
        LDA Page2         back to aux
.1      TYA
        AND #%0000.1000   receive data reg full mask
        BEQ CantRx        =>not full
        LDA Byte.Counter.Lo   received a byte,
        LDX Byte.Counter.Mid  do we assert RTS' ?
        LDY Byte.Counter.Hi
        CMP #BufLen-256
        BNE .2            =>buffer not @ full-256
        CPX /BufLen-256
        BNE .2            =>buffer not @ full-256
        CPY ^BufLen-256
        BNE .2            =>buffer not @ full-256
        LDA #%000.0.10.0.0   assert RTS'
        TRB Modem.ACIA.Command
        LDA Byte.Counter.Lo  reload it
.2      INC               fig next byte count
        BNE .3
        INX
```

```
               BNE .3
               INY
        .3     CMP #BufLen       do we have room for it ?
               BNE Room          =>buffer not full
               CPX /BufLen
               BNE Room          =>buffer not full
               CPY ^BufLen
               BNE Room          =>buffer not full
               LDA Page1         access main text screen
               INC $402          indicate full
               LDA Page2         back to aux
        CantRx BRA Cant.Receive  =>buffer is full!
        Room   STA Byte.Counter.Lo
               STX Byte.Counter.Mid
               STY Byte.Counter.Hi
               LDA Page1         access main text screen
               INC $400          show we received a byte
               LDA Page2         back to aux
               LDX Modem.ACIA.RxD
               TXS               pass it in S-reg
               LDX Receive.Adr.Lo
               LDY Receive.Adr.Hi
               BIT LCRAM2        normally use LC bank 2
               TYA
               AND #$F0
               CMP /$C000        if adr is in $CXXX range
               BNE .1
               BIT LCRAM1         use LC bank 1
               TYA
               ORA /$D000
               TAY
        .1     LDA Receive.Bank
               JMP Write
        W.Ret  INC Receive.Adr.Lo  fig next receive adr
               BNE Scan.Printer.Port
               INC Receive.Adr.Hi
               BNE Scan.Printer.Port
               LDA Receive.Bank
               CMP #1
               ADC #1            clear carry if 0, else set it
               CMP #$10
               BCC .1            =>entering/still in Z-RAM
               LDA #$00          wrap to //c bank 0
               LDX #IIc.Aux.Bank.Avail
               LDY /IIc.Aux.Bank.Avail
               BRA .2
        .1     LDX #Z.RAM.Banks.Avail
               LDY /Z.RAM.Banks.Avail
        .2     STA Receive.Bank
               STX Receive.Adr.Lo
               STY Receive.Adr.Hi
        Cant.Receive
        Scan.Printer.Port
               LDA #%0011.0000   make transmit data reg empty and
```

```
        AND Printer.ACIA.Status  Data Carrier Detect mask
        CMP #%0001.0000   test empty and DCD' lo
        BNE Cant.Transmit =>not empty or not ready
        LDA Byte.Counter.Lo   printer can take another byte,
        ORA Byte.Counter.Mid  do we have one ?
        ORA Byte.Counter.Hi
        BEQ Cant.Transmit     =>buffer is empty!!!
        LDA Byte.Counter.Lo   do we release RTS' ?
        LDX Byte.Counter.Mid
        LDY Byte.Counter.Hi
        CMP #BufLen-2048
        BNE .1            =>buffer not @ full-2048
        CPX /BufLen-2048
        BNE .1            =>buffer not @ full-2048
        CPY ^BufLen-2048
        BNE .1            =>buffer not @ full-2048
        LDA RTS.Bit
        TSB Modem.ACIA.Command  release RTS' (maybe)
.1      STA Z.RAM.Bank.Reg+5
        LDA Byte.Counter.Lo     fig next byte count
        BNE .3
        LDA Byte.Counter.Mid
        BNE .2
        DEC Byte.Counter.Hi
.2      DEC Byte.Counter.Mid
.3      DEC Byte.Counter.Lo
        LDA Page1         access main text page
        INC $427          show we printed a byte
        LDA Page2         back to aux
        LDX Transmit.Adr.Lo
        LDY Transmit.Adr.Hi
        BIT LCRAM2        normally use LC bank 2
        TYA
        AND #$F0
        CMP /$C000        if adr in $CXXX range
        BNE .4
        BIT LCRAM1         use LC bank 1
        TYA
        ORA /$D000
        TAY
.4      LDA Transmit.Bank
        JMP Read
R.Ret   STA Printer.ACIA.TxD
        INC Transmit.Adr.Lo  fig next transmit adr
        BNE Next
        INC Transmit.Adr.Hi
        BNE Next
        LDA Transmit.Bank
        CMP #1            clear carry if 0, else set it
        ADC #1
        CMP #$10
        BCC .1            =>entering/still in Z-RAM
        LDA #$00          wrap to //c bank 0
        LDX #IIc.Aux.Bank.Avail
```

```
        LDY /IIc.Aux.Bank.Avail
        BRA .2
.1      LDX #Z.RAM.Banks.Avail
        LDY /Z.RAM.Banks.Avail
.2      STA Transmit.Bank
        STX Transmit.Adr.Lo
        STY Transmit.Adr.Hi
Cant.Transmit
Next    JMP Scan
IIc.Aux.Bank.Avail .eq *
BufLen .eq $90000-Z.RAM.Used-IIc.Aux.Bank.Avail
 .lif
```

```
==============================================================================
DOCUMENT :AAL-8508:ProDOS:BUF.64K.txt
==============================================================================

 1000   .ti 81,BNApple //c buffering program 2.0 8/5/85 dcj
 1010  ;SAVE Buf.64K
 1020  ;---------------------------------
 1030  ; Dedicated to Allan B. Calhamer.
 1040  ;---------------------------------
 1050  Printer.ACIA.TxD      .eq $C098 (w)
 1060  Printer.ACIA.Status   .eq $C099 (r)
 1070  Printer.ACIA.Command  .eq $C09A (r/w)
 1080  Printer.ACIA.Control  .eq $C09B (r/w)
 1090  Modem.ACIA.RxD        .eq $C0A8 (r)
 1100  Modem.ACIA.Status     .eq $C0A9 (r)
 1110  Modem.ACIA.Command    .eq $C0AA (r/w)
 1120  Modem.ACIA.Control    .eq $C0AB (r/w)
 1130  Keyboard              .eq $C000 (r)
 1140  Store80               .eq $C001 (w) on
 1150  RAMRd                 .eq $C003 (w) aux
 1160  RAMWrt                .eq $C005 (w) aux
 1170  AltZP                 .eq $C009 (w) aux
 1180  Vid40                 .eq $C00C (w)
 1190  SetAltChr             .eq $C00F (w) w/MouseText
 1200  Clear.Key.Strobe      .eq $C010 (r)
 1210  Text                  .eq $C051 (r)
 1220  Page1                 .eq $C054 (r) main
 1230  Page2                 .eq $C055 (r) aux
 1240  Hires                 .eq $C057 (r) $2000-$3FFF too...
 1250  LCRAM2                .eq $C083 (r/w; write doesn't
 1260  LCRAM1                .eq $C08B  change write enable)
 1270  ;---------------------------------
 1280         .op 65C02
 1290         .or $2000
 1300         .tf /IIc.buf/Bufit64K/
 1310  dcj    SEI                 close this can of worms...
 1320         LDA LCRAM2          1x...switches setup
 1330         LDA Text
 1340         LDA Page1
 1350         LDA Hires
 1360         STZ Store80
 1370         STZ RAMRd
 1380         STZ RAMWrt
 1390         STZ AltZP
 1400         STZ SetAltChr
 1410         STZ Vid40
 1420         LDX #0
 1430  .1     LDA #" "            clear 40 column screen
 1440         STA $400,X
 1450         STA $500,X
 1460         STA $600,X
 1470         STA $700,X
 1480         LDA Image,X         install Image in aux ZP/Stack
```

```
1490         STA $00,X
1500         INX
1510         BNE .1
1520         LDY #$0F
1530  .2     LDA Msg,Y          put up a message
1540         STA $50C,Y
1550         DEY
1560         BPL .2
1570         LDA #%000.0.10.1.0         bop ACIAs
1580         STA Printer.ACIA.Command
1590    inc  LDA #%000.0.10.1.1         RTS' lo
1600         STA Modem.ACIA.Command
1610         LDA #%0.00.1.1110          9600 baud
1620         STA Printer.ACIA.Control
1630    inc  LDA #%0.00.1.1111          19200 baud!
1640         STA Modem.ACIA.Control
1650         LDA Modem.ACIA.RxD
1660         JMP Scan           go 2 it
1670  Msg    .AS 'A'            as in Apple
1680         .AS -" //c buffer pgm"
1690  Image  .ph $00
1700  ;--------------------------------
1710  ; buffer starts at first available location in //c aux bank
1720  Receive.Adr.Hi      .da /IIc.Aux.Bank.Avail
1730  Receive.Adr.Lo      .da IIc.Aux.Bank.Avail
1740  Transmit.Adr.Hi     .da /IIc.Aux.Bank.Avail
1750  Transmit.Adr.Lo     .da IIc.Aux.Bank.Avail
1760  Byte.Counter.Lo     .da #$0000          indicates empty
1770  Byte.Counter.Hi     .da /$0000
1780  RTS.Bit             .da #%000.0.10.0.0  RTS' lo
1790  ;--------------------------------
1800  Scan   LDA Page1          access main text screen
1810         INC $413           show we're alive
1820         DEC $414
1830         LDA Page2          back to aux
1840         LDA Keyboard       scan keyboard
1850         BPL Scan.Modem.Port
1860         CMP #" "           space toggles RTS' (DTR2B) to //e
1870         BNE .2
1880         LDA Modem.ACIA.Command
1890         AND #%000.0.10.0.0
1900         BNE .1             =>It's ok, you can turn it off...
1910         LDA RTS.Bit
1920         BNE Scan.Modem.Port  =>don't do it! (yet)
1930  .1     LDA Modem.ACIA.Command
1940         EOR #%000.0.10.0.0
1950         STA Modem.ACIA.Command
1960         AND #%000.0.10.0.0
1970         STA RTS.Bit
1980  .2     BIT Clear.Key.Strobe
1990  Scan.Modem.Port
2000         LDY Modem.ACIA.Status
2010         TYA
2020         AND #%0000.0111   error bits mask
```

```
2030         BEQ .1            =>error-free operation
2040         TAX
2050         LDA Page1         access main text screen
2060         INC $4FF,X        indicate error...
2070         LDA Page2         back to aux
2080  .1     TYA
2090         AND #%0000.1000   receive data reg full mask
2100         BEQ Cant.Receive  =>not full
2110         LDX Byte.Counter.Lo   received a byte,
2120         LDY Byte.Counter.Hi   do we assert RTS' ?
2130         CPX #BufLen-256
2140         BNE .2            =>buffer not @ full-256
2150         CPY /BufLen-256
2160         BNE .2            =>buffer not @ full-256
2170         LDA #%000.0.10.0.0   assert RTS'
2180         TRB Modem.ACIA.Command
2190  .2     INX               fig next byte count
2200         BNE .3
2210         INY
2220  .3     CPX #BufLen       do we have room for it ?
2230         BNE Room          =>buffer not full
2240         CPY /BufLen
2250  ; $402 indicator gone to fit into ZP (cause I wanna)
2260         BEQ Cant.Receive  =>buffer is full!
2270  Room   STX Byte.Counter.Lo
2280         STY Byte.Counter.Hi
2290         LDA Page1         access main text screen
2300         INC $400          show we received a byte
2310         LDA Page2         back to aux
2320         LDY Receive.Adr.Hi
2330         BIT LCRAM2        normally use LC bank 2
2340         TYA
2350         AND #$F0
2360         CMP /$C000        if adr is in $CXXX range
2370         BNE .1
2380         BIT LCRAM1         use LC bank 1
2390         TYA
2400         ORA /$D000
2410         TAY
2420  .1     STY Receive.Adr.Lo+1
2430         LDA Modem.ACIA.RxD
2440         STA (Receive.Adr.Lo)
2450         INC Receive.Adr.Lo  fig next receive adr
2460         BNE Scan.Printer.Port
2470         INC Receive.Adr.Hi
2480         BNE Scan.Printer.Port
2490         LDX #IIc.Aux.Bank.Avail
2500         STX Receive.Adr.Lo
2510  Cant.Receive
2520  Scan.Printer.Port
2530         LDA #%0011.0000   make transmit data reg empty and
2540         AND Printer.ACIA.Status  Data Carrier Detect mask
2550         CMP #%0001.0000   test empty and DCD' lo
2560         BNE Cant.Transmit =>not empty or not ready
```

```
2570         LDA Byte.Counter.Lo   printer can take another byte,
2580         ORA Byte.Counter.Hi   do we have one ?
2590         BEQ Cant.Transmit     =>buffer is empty!!!
2600         LDX Byte.Counter.Lo   do we release RTS' ?
2610         LDY Byte.Counter.Hi
2620         CPX #BufLen-2048
2630         BNE .1            =>buffer not @ full-2048
2640         CPY /BufLen-2048
2650         BNE .1            =>buffer not @ full-2048
2660         LDA RTS.Bit
2670         TSB Modem.ACIA.Command  release RTS' (maybe)
2680  .1     STA $C073+5
2690         LDA Byte.Counter.Lo     fig next byte count
2700         BNE .2
2710         DEC Byte.Counter.Hi
2720  .2     DEC Byte.Counter.Lo
2730         LDA Page1         access main text page
2740         INC $427          show we printed a byte
2750         LDA Page2         back to aux
2760         LDY Transmit.Adr.Hi
2770         BIT LCRAM2        normally use LC bank 2
2780         TYA
2790         AND #$F0
2800         CMP /$C000        if adr in $CXXX range
2810         BNE .4
2820         BIT LCRAM1         use LC bank 1
2830         TYA
2840         ORA /$D000
2850         TAY
2860  .4     STY Transmit.Adr.Lo+1
2870         LDA (Transmit.Adr.Lo)
2880         STA Printer.ACIA.TxD
2890         INC Transmit.Adr.Lo  fig next transmit adr
2900         BNE Next
2910         INC Transmit.Adr.Hi
2920         BNE Next
2930         LDX #IIc.Aux.Bank.Avail
2940         STX Transmit.Adr.Lo
2950  Cant.Transmit
2960  Next   JMP Scan
2970  IIc.Aux.Bank.Avail .eq *
2980  BufLen .eq $10000-IIc.Aux.Bank.Avail
2990   .lif
```

================================================================================
DOCUMENT :AAL-8509:Articles:Convert.65802.txt
================================================================================

Short Binary-to-Decimal Conversion in 65802...........Bob S-C

Since the 65802 supports 16-bit registers, it is possible to write a very tiny loop that will convert 16-bit binary numbers to four- or five-digit decimal values.  Jim Poponoe called today and suggested the idea to me.

The idea is to count down the binary number in binary mode while incrementing a four-digit decimal value in the A-register.  It certainly isn't very fast, but it is small.

The two programs below illustrate the technique.  CONV.1 converts a two-byte value at $0000 (and $0001) and stores the four-digit result in $0002 (and $0003).  CONV.1 goes one step further, handling a fifth digit which is stored in $0004.

You could use CONV.1 inside the CATALOG command to convert volume numbers and sector counts.  It is probably shorter than the existing code.  Since the numbers converted are less than 500, the maximum time is still less than half a millisecond.

Lines 1080 and 1090 put the 65802 into "native" mode, so that we can use the 16-bit features.  Lines 1210,1220 put the 65802 back into 6502 "emulation" mode, since the subroutine was written under the assumption that the caller would be in emulation mode.  If you plan to use the subroutine within a program that runs entirely in native mode, you could leave these four lines out.  If you plan to call it from both native mode and emulation mode, you need to save the E status and restore it at the end.  You can do that like this:

```
       CONV.1 CLC      ENTER NATIVE MODE
              XCE
              PHP      SAVE CALLER'S MODE (IN C-BIT)
               .
               .
               .
              PLP      GET CALLER'S MODE
              XCE      RESTORE CALLER'S MODE
              RTS
```

Line 1100 clears both the X- and M-bits, so that all 16-bit features are on.  Note that when either of these bits are cleared, immediate-mode operands are two bytes long.  The assembler doesn't keep track of the state of these two bits, because it would be impossible in the general case without a complete flow analysis of the program.  It is up to the programmer to tell the assembler whether to assemble one- or two-byte immediate operands.  You do this in S-C Macro Assembler by using a double pound-sign notation, as in lines 1110 and 1160.

Line 1110 loads a full 16-bit value zero into the A-register. Line 1120 loads the 16-bit value from location $0000 and $0001. the low byte of the value is in $0000, and the high byte in $0001. If all 16-bits of this value are zero, line 1130 will branch around the conversion loop. If not, it will not branch.

Line 1140 sets the decimal mode, which affects only the ADC and SBC instructions. Line 1190 turns it back to binary. If you use the PHP and PLP steps shown above in the discussion about native and emulation modes, you could leave out the CLD in line 1190: the PHP would restore the D-bit properly.

The loop in lines 1160-1180 adds one to the A-register and subtracts one from the X-register, until the X-register reaches zero. Since we are in decimal mode, the A-register counts up in BCD format. The largest number the loop can handle correctly is 9999 decimal ($270F). Larger values will not even have the correct lower four digits, since CARRY gets set when 9999 is incremented.

After the loop finishes, line 1200 stores the result low-byte- first at $0002 and $0003.

CONV.2 is almost identical to CONV.1, on purpose. There are five new lines of code, at lines 1330, 1390-1410, and 1480. We use the Y-register to keep track of the fifth digit, so that we can convert numbers larger than 9999. Line 1330 sets Y=0. Line 1390 checks for the carry that occurs when 9999 is incremented. If there is no carry, the loop is the same as in CONV.1. If there is a carry, line 1400 increments the Y-register and line 1410 clears carry. (We could save one byte at the expense of slower operation by including the CLC on line 1370 inside the conversion loop.)

Line 1480 stores the fifth digit in location $0004. I put it after the switch back to emulation mode, since I only wanted to store one byte.

I timed these subroutines by counting cycles, as shown in the comments in lines 1040,1050 and 1250,1260. In the process I was suprised to learn that the DEX opcode still takes only two cycles, even when in 16-bit mode. Of course, the same goes for INX, DEY, INY. It is also true of ASL, LSR, ROL, ROR, INC, and DEC when the operand is the A-register.
1

-----------------------------------------------------------------------------
DOCUMENT :AAL-8509:Articles:DOS.PDos.Init.txt
-----------------------------------------------------------------------------

Put DOS and ProDOS Files on Same Disk......Bob Sander-Cederlof

In the February 1985 issue of AAL I showed how to create a DOS-less DOS 3.3 data disk. Tracks 1 and 2, normally full of the DOS image, were instead made available for files. Booting the disk gets you a message that such a disk cannot be booted.

Now that we are publishing more and more programs intended for use under ProDOS, we foresee the need to publish Quarterly Disks that contain both DOS and ProDOS programs. Believe it or not, this is really possible.

The DOS operating system keeps its Volume Table of Contents (VTOC) and catalog in track $11. The VTOC is in sector 0 of that track, and the catalog normally fills the rest of the track. A major part of the VTOC is the bit map, which shows which sectors are as yet unused by any files. If we want to reserve some sectors for use by a ProDOS directory on the same disk, we merely mark those sectors as already being in use in the DOS bit map.

ProDOS keeps its directory and bit map all in track 0. This track is not available to DOS for file storage anyway, so we can be comfortable stealing it for a ProDOS setup on the same diskette.

I decided to keep things fairly simple, by splitting the disk into two parts purely on a track basis. ProDOS gets some number of tracks starting with track 0, and DOS gets all the tracks from just after ProDOS to track 34. If ProDOS gets more than 17 tracks, it will hop over track $11 (since DOS's catalog is there). Normally I will split the disk in half, giving tracks 0-16 to ProDOS and tracks 17-34 to DOS. With this arrangement, ProDOS thus starts with 129 free blocks, and DOS starts with 272 free sectors.

The program I wrote does not interact with the user; instead, you set all the options by changing the source code and re-assembling. It would be nice to have an interactive front end to get slot, drive, volume number for the DOS half, volume name for the ProDOS half, and how many tracks to put in each half. Maybe we'll add this stuff later, or maybe you would like to try your hand at it.

The parameters you might want to change are found in lines 1020-1050. You can see that I started the DOS allocation at track $12, just after the catalog track. I also chose volume 1, drive 1, slot 6. You can use any volume number from 1 to 254. Since these numbers were under my control, I did not bother to check for legal values. If we add an interactive front end, we will have to validate them. We might also want to display the number of ProDOS blocks and DOS sectors that result from the DOS.LOW.TRACK selection, maybe in a graphic format. You might even use a joystick or mouse....

You might also want to change the ProDOS volume name. I am calling it "DATA". The name is in line 2850. It can be up to 15 characters long, and the number of characters must be stored in the right nybble of the byte just before the name. This is automatically inserted for you, by the assembler. If you should try to assemble a name larger than 15 characters, line 2870 will cause a RANGE ERROR. Another way of changing the ProDOS volume name is to do so after initialization using the ProDOS FILER program.

Lines 1090 and 1100 compute the number of free DOS sectors and ProDOS blocks. The values are not used anywhere in the program, but are nice to know.

Line 1300 sets the program origin at $803. Why $803, and not $800? If we load and run an assembly language program at $800, and then later try to load and run an Applesoft program, Applesoft can get confused. Applesoft requires that $800 contain a $00 value, but it does not make sure it happens when you LOAD an Applesoft program from the disk. By putting our program at $803 we make sure we don't kill the $00 and $800. Well, then why not start at $801? I don't know, we just always did it that way. (It would make good sense if our program started by putting $00 in $801 and $802, indicating to Applesoft that it had no program in memory.)

DOUBLE.INIT is written to run under DOS 3.3, and makes calls on the RWTS subroutine to format and write information on the disk. The entire DOUBLE.INIT program is driven by lines 1320-1490. The flow is very straightforward:

1. Format the disk as 35 empty tracks.
2. Write DOS VTOC and Catalog in track 17.
3. Write ProDOS Directory and bit map in track 0.
4. Write "YOU CANNOT BOOT" code in boot sector.

Formatting a blank disk is simple, unless you have a modified DOS with the INIT capability removed. Lines 1510-1590 set up a format call to RWTS, and fall into my RWTS caller.

Lines 1600-1800 call RWTS and return, unless there was an error condition. If there was an error, I will print out "RWTS ERROR" and the error code in hex. The error code values you might see are:

    $08 -- Error during formatting
    $10 -- Trying to write on write protected disk
    $40 -- Drive error

I don't think you can get $20 (volume mismatch) or $80 (read error) from DOUBLE.INIT. After printing the error message, DOS will be warm started, aborting DOUBLE.INIT.

Building the DOS VTOC and Catalog is handled by lines 1820- 2310. The beginning section of the VTOC contains information about the number of tracks and sectors, where to find the catalog, etc. This is all assembled in at lines 2260-2310, and is copied into my buffer by lines 1880-1930. Since the volume number is a parameter, I specially load it in with lines 1940 and 1950. The rest of the VTOC is a bit map showing which sectors are not yet used. Lines 1960-2090 build this bit map. Lines 1840-1870 and 2100-2120 cause the VTOC image to be written on track 17 ($11) sector 0.

There are some unused bytes in the beginning part of the VTOC, so I decided to put some private information in there. See line 2270 and line 2290.

The rest of track 17 is a series of empty linked sectors comprising the catalog. The chain starts with sector $0F, and works backward to sector 1. Lines 2130-2240 build each sector in turn and write it on the disk.

The ProDOS directory and bit map are installed in track 0 by lines 2330-2900. This gets a little tricky, because we are trying to write ProDOS blocks with DOS 3.3 RWTS. Here is a correspondence table, showing the blocks and sectors in track 0:

```
       ProDOS Block:  0   1   2   3   4   5   6   7
    DOS 3.3 Sectors: 0,E D,C B,A 9,8 7,6 5,4 3,2 F,1
```

The first sector of each pair contains the first part of each block, and so on.

The ProDOS bit map goes in block 6, which is sectors 3 and 2. Even if we had an entire diskette allocated to ProDOS the bit map would occupy very little of the first of these two sectors. Since formatting the disk wrote 256 zeroes into every sector, we can leave sector 2 unchanged. Lines 2700-2820 build the bit map data for sector 3 and write it out. Note that block 7 is available, all blocks in track 17 are unavailable.

The ProDOS Directory starts in block 2. The first two bytes of a directory sector point to the previous block in the directory chain, and the next two bytes point to the following block in the chain. We follow the standard ProDOS convention of linking blocks 3, 4, and 5 into the directory. Those three blocks contain no other information, since there are as yet no filenames in the directory. Here's how the chain links together:

```
              Previous  Next
               Block   Block
       Block 2:   0       3   (zero means the beginning)
       Block 3:   2       4
       Block 4:   3       5
       Block 5:   4       0   (zero means the end)
```

Block 2 gets some extra information, the volume header. Lines 2840-2900 contain the header data, which is copied into my buffer by lines 2590-2630.

The no-booting boot program is shown in lines 3000-3190. This is coded as a .PHase at $800 (see lines 3010 and 3190), since the disk controller boot ROM will load it at that address. All the program does is turn off the disk motor and print out a little message. Lines 1410-1490 write this program on track 0 sector 0.

I think if you really wanted to you could put a copy of the ProDOS boot program in block 0 (sectors 0 and E). Then if you copied the file named PRODOS into the ProDOS half of the disk, you could boot ProDOS.

There is one thing to look out for if you start cranking out DOUBLE DISKS. There are some utility programs in existence which are designed to "correct" the DOS bitmap in the VTOC sector. Since these programs have never heard of ProDOS, let alone of DOUBLE DISKS, they are going to tell DOS that all those tracks we carefully gave to ProDOS belong to DOS. If you let that happen to a disk on which you have already stored some ProDOS files, zzzaaaapppp!

```
==============================================================================
DOCUMENT :AAL-8509:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 5 -- Issue 12                September, 1985

In This Issue...



Many of you have added 65802 processors to your Apples, and are now looking for more data on programming the new features of this powerful chip. We've been getting several calls per week asking: "Now that I have this thing, how can I find out more about it?" Well, this issue of AAL will keep you folks busy for a while! We have that standard benchmark, the Sieve of Eratosthenes, coded in 65802, along with a startlingly small routine to convert binary to decimal. And more to come...

In another couple of months there will be a significant addition to the 65816 library. We've been previewing a copy of the galley proofs of a new book on the 65816 by David Eyes. We will have a full review of this important resource, and copies for sale, as soon as the book is really available.

"Now that You Know Apple Assembly Language, What Can You Do with It?" That's the title of a new book written and published by Jules Gilder, a long time contributor to Apple magazines. We'll have a full review next month, and may be carrying the book. In the meantime, see Jules' ad on page 7 of this issue.

S-C Macro Assembler Version 2.0 DOS Source Code

Here's another item we've had many requests for: the source code to S-C Macro Assembler Version 2.0. Now that the DOS ver- sion has been out a few months, and seems to be stable, we're releasing the source code. Registered owners of S-C Macro Assembler Version 2.0 can purchase the source, on disk, for only $100. Those of you who purchased the source of an earlier Macro version may add the 2.0 source for only $50. It will be a few more months until the ProDOS Version source code appears.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage

for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

```
=================================================================
DOCUMENT :AAL-8509:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.......................................................$30
Source Code for DISASM....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051,
   8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
                                 Jan-Mar   Apr-Jun   Jul-Sep   Oct-Dec
 Each disk contains       1980      -         -         -         1
 the source code from     1981      2         3         4         5
 three issues of AAL,     1982      6         7         8         9
 saving you lots of       1983     10        11        12        13
 typing and testing.      1984     14        15        16        17
                          1985     18        19        20

(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $32
Vinyl disk pages, 6"x8.5", hold two disks each.................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50
quikLoader EPROM System (SCRG)...............................($179) $170
PROmGRAMER (SCRG)..........................................($149.50) $140
Switch-a-Slot (SCRG).........................................($190) $175
Extend-a-Slot (SCRG)..........................................($35)  $32

"Apple ProDOS:  Advanced Features for programmers", Little..($17.95)  $17
"Inside the Apple //c", Little.............................($19.95)  $18
"Inside the Apple //e", Little.............................($19.95)  $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18
"Apple ][ Circuit Description", Gayler.....................($22.95)  $21
"Understanding the Apple II", Sather.......................($22.95)  $21
"Understanding the Apple //e", Sather......................($24.95)  $23
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20
"Beneath Apple DOS", Worth & Lechner.......................($19.95)  $18
```

```
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18
"6502 Subroutines", Leventhal...............................($18.95)  $18
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9
"Microcomputer Graphics", Myers.............................($12.95)  $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner........................($19.95)  $18
"AppleVisions", Bishop & Grossberger........................($39.95)  $36
```

Add $1.50 per book for US shipping.  Foreign orders add postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

```
*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
***                  (214) 324-2050                  ***
*** We accept Master Card, VISA and American Express ***
```

==============================================================================
DOCUMENT :AAL-8509:Articles:PrimeSieve65802.txt
==============================================================================

Prime Benchmark for 65802..................Bob Sander-Cederlof

Jim Gilbreath really started something. He is the one who popularized the use of the Sieve of Eratosthenes as a benchmark program for microcomputers and their various languages. You can read about it in BYTE September 1981, "A High-Level Language Benchmark"; and later in BYTE January 1983, "Eratosthenes Revisited".

In a nutshell, the benchmark creates an array of 8192 bytes, representing the odd numbers from 1 to 16383. The prime numbers in this array are flagged by the program using the Eratosthenes algorithm. All of the times published in the BYTE articles are for ten repetitions of the algorithm.

The second article lists page after page of timing results for various computers and languages. They range from .0078 seconds for an assembly language version running in an IBM 3033, to 5740 seconds for a Cobol version in a Xerox 820.

There are many factors which affect the results, not just the basic speed of the computer involved. The language used is obviously significant, as some languages are more efficient than others for particular purposes. Slight variations in the implementation of the Eratosthenes algorithm can be very significant. The skill and persistence of the programmer are also very important.

Gilbreath's times for the Apple II vary from 2806 seconds for an Applesoft version to 160 seconds for a Pascal version. The same table shows an OSI Superboard, using a 6502 like the Apple, ran an assembly language version in 13.9 seconds. (I don't know what the clock rate of the OSI board was.)

We have published a series of articles in AAL on the same subject. "Sifting Primes Faster and Faster", in October 1981, gave programs in Apple assembly language by William Robert Savoie and myself. At the time I had overlooked the fact that BYTE's times were for ten trips through the program, so I was perhaps a little overly enthusiastic. The table below shows the adjusted times for ten repetitions.

| Version | Time in seconds |
|---|---|
| My Integer BASIC version | 1880 |
| Mike Laumers Int BASIC | 2380 |
| Mike's compiled by FLASH! | 200 |
| Bill Savoie's 6502 assembly | 13.9 |
| My first re-write of Bill's | 9.3 |
| My 6502 version | 7.4 |
| My 6502 with faster clear | 6.9 |

I challenged you readers to do it faster, and some of you did. Charles Putney ("Even Faster Primes", Feb 1982 AAL) knocked the time for ten trips down to 3.3 seconds. Tony Brightwell ("Faster than Charlie", Nov 1982 AAL) combined tricks from number theory with a faster array clear technique to trim the time to 1.83 seconds. Peter McInerney sent us an implementation he did on the DTACK Grounded 68000 board, which uses a 12.5 MHz clock. His program ("68000 Sieve Benchmark", July 1984 AAL) did 10 repetitions in .4 seconds. (An 8 MHz time was logged in the BYTE article at .49 seconds. Upping the clock speed does not always speed everything up proportionally, due to the need to wait for slower memory chips.) I translated Peter's code back to 6502 code in "Updating the 6502 Prime Sifter", same issue. My time for ten loops was 1.75 seconds. In that article I stated, "...it remains to be seen what the 65802 could do.

David Eyes, in his new book on 65816 Assembly Language, presents a version which uses the expanded capabilities in that chip. He evidently did not build on our base, because his time for a 4 MHz 65816 was 1.56 seconds. I presume that means if the clock rate was the same as Apple's it would have taken 6.24 seconds. I have been previewing David's book, from the galleys, but the listing of that program was not included in the material I received from the typesetter.

I decided to try updating my 1984 version to 65802 code, using whatever tricks I could come up with. The result runs ten times in 1.4 seconds in the 65802 plugged into my Apple II Plus. I suppose that means a 4 MHz version would run in .35 seconds, or faster than a 12.5 MHz 68000!

Lines 1100-1210 are an outer shell to drive the PRIME program. The shell begins and ends by ringing the Apple bell, to help me run my stopwatch. I ran the PRIME program 1000 times, and then divided the time by 100 to get the seconds for ten repetitions. In between ringing the bells everything is done in 65802 mode. Lines 1110-1120 turn on "native" mode, and lines 1190-1200 restore "emulation" mode.

When you switch on native mode the M and X bits always come up as 1's. That is, both are set to 8-bit mode. The M-bit controls the size of operations on the A-register, and the X-bit controls the size for the X- and Y-registers. Line 1130 turns on 16-bit mode for the A-register. I use this setting throughout the rest of the program, until we go back to emulation mode. All operations which affect the A-register will be 16-bits, while I will only use X and Y with 8-bit values.

Lines 1140-1180 call PRIME 1000 times. Since I have Mbit=0, line 1140 uses the 16-bit LDA immediate. STA COUNT stores both bytes: the low byte at COUNT and and the high byte at COUNT+1. DEC COUNT decrements the full 16-bit value, returning a .NE. status until both bytes are zero. This is certainly a lot easier than a two-byte decrement in 6502 code:

```
       LDA COUNT
```

```
       BNE .1
       DEC COUNT+1
  .1   DEC COUNT
       BNE ...      ...NOT AT 0000 YET
       LDA COUNT+1
       BNE ...      ...NOT AT 0000 YET
```

Line 1140 may need some explanation, since there are now at least four assemblers available for the Apple which handle 65802 assembly language.  Each of the four have chosen a different way to inform the assembler about the number of bytes to assemble for immediate operands.  S-C Macro uses  a "#" to indicate and 8-bit operand, and "##" to indicate a 16-bit immediate operand.  This seems to me to be the easiest to figure out when I come back to read a program listing after several weeks of working on something else.  The "double #" is an immediate visual clue (pun intended) that the immediate operand is double size.

Since ORCA/M was a Hayden Software product, and David Eyes was product manager of ORCA/M at Hayden as well as an early contributor to 65816 design, ORCA/M turned out to be the first assembler to include 65816 support.  Mike Westerfield had a version running before the rest of us even knew the 65816 was going to exist.  Consequently, Mike's and David's choices for assembly syntax and rules has achieved the honor of being used in the 65816 data sheet and in David's book.

Mike and David decided to inform the assembler what size immediate operands to use with two assembler directives.  LONGA controls the size of immediate operands on LDA, CMP, ADC, ORA, EOR, AND, BIT, and SBC:  LONGA ON makes them 16-bits, LONGA OFF makes them 8-bits.  Likewise, LONGI ON or OFF controls the immediate operands on LDX, LDY, CPX, and CPY.  You have to sprinkle your code with these so that the assembler always knows which size to use.  Since the directives may not be close to the affected lines of code, it can be a chore to read unfamiliar source code.

Merlin Pro uses a single directive to inform the assembler as to the settings of M and X which will be in effect at execution time.  The directive is called "MX", and can have an operand of 0, 1, 2, or 3 (or a symbol whose value is 0-3).  The bits of the value correspond to the M- and X-bit settings:

```
       MX 0    M=0, X=0  (both 16-bits)
       MX 1    M=0, X=1  (A/16, XY/8)
       MX 2    M=1, X=0  (A/8, XY/16)
       MX 3    M=1, X=1  (A/16, XY/16)
```

I understand that the latest version of Lazerware's Lisa Assembler supports the 65816, but I don't have a copy.  I do not know how Randy Hyde indicates immediate operand size.

By the way, in all of the assemblers it is entirely up to the programmer to be sure that you keep all the immediate sizes correct. There is no way for an assembler to second-guess you on this.  If you tell it to make a 16-bit operand, and then execute that instruction in

8-bit mode, the third byte will be treated as the next opcode. Vice versa is just as bad. I have blown it many times already, with the result that I am a lot more careful now.

Now let's look at the PRIME subroutine itself. The first section clears an array of 8192 bytes, storing $00 in each byte. There are a lot of ways to store zeroes. The most obvious is with a loop of STA addr,X lines, such as we used in previous versions. The 65802 has a STZ instruction, which stores zero without using the A-register, but it is not faster. We could store a zero at the beginning of the array and then use an overlapping MVN instruction to copy that zero through the whole array:

```
       LDX ##BASE
       LDY ##BASE+1
       LDA ##8190
       MVN 0,0
```

That would be simple, but it would take over 56000 cycles. We can do a lot better than that.

My version uses the PHD instruction 4096 times to push 8192 zeroes on the stack. I start by setting the stack register to point at the last byte of my array (BASE+8191). Each PHD pushes the direct page register (which is currently set to $0000) on the stack. My loop includes 16 PHD's, so 256 times around will fill the array (or empty it, if you like). All this action is in lines 1320-1380. To save space in the source code, rather than write 16 lines of PHD's, I wrote them out as hex strings in lines 1350-1360.

Lines 1310, 1390-1410 save and restore the original stack pointer. (At first I didn't do this, with disastrous results! The stack pointer was sitting just below the cleared array. When I did an RTS, the next opcode encountered was $00, which is a BRK. Since I was in native mode, the BRK vectored through $FFE6,7 instead of $FFFE,F. Et cetera.) Note that the TSX only saves the low byte of S, because X is in 8-bit mode. I am assuming that the high byte was $01, since I came from normal Apple 6502 code. Lines 1390-1400 put $01 in front of the low byte, and the TCS puts both bytes back in the S-register.

Lines 1430-1440 push the address of the fifth byte in the array onto the stack. Since the 65802 has a stack-relative addres- sing mode, we can access the pointer with an address of "1,S". Remember the bytes in the array represent the odd numbers. The fifth byte represents the number 9, which is the square of the first odd prime (3). (At a very slight penalty in speed, we can change line 1430 to "LDA ##BASE" and delete line 1460.)

Lines 1480-1520 update the pointer we are keeping on the stack to point to the next square. For an explanation of how this works, go to the July 1984 and Nov 1982 articles. Lines 1530-1540 skip the sifting process for numbers that have already been flagged as non-prime.

Lines 1550-1580 compute the prime number itself from the index (2*index+1) and store it into the operand bytes of the "ADC ##" instruction at line 1630. Ouch! Self-modifying code! But that is often the price of speed.

Line 1590 picks up the pointer to the square of the prime, which is the first number that must be flagged as non-prime, from our holding location on the stack. Lines 1610-1640 get tricky. Line 1610 puts the current pointer in the D-register, which tells where in RAM the direct page starts. This means that the "STX 0" in line 1620 stores into the byte pointed to. X was holding the current index, so we are storing a non-zero number into that byte, which flags it as being non-prime.

As a pleasant side effect, the non-zero numbers being stored in the array have meaning. If we double the value we stored and add one, we will get the value of the prime factor of the non-prime number. After the whole PRIME program has executed, the flag value will produce the largest prime factor.

In the loop of lines 1610-1640, we keep adding the prime number to the pointer value in the A-register, and transferring the result to the D-register. Hence the STX 0 will store X at multiples of the prime number. The loop terminates when the pointer value in the A-register goes negative. Why? Because we carefully positioned the array from $6000 to $7FFF. The first time we add the prime to the pointer and get an address $8000 or higher, we know we went off the end of the array. Addresses of $8000 or higher will set the negative status flag, so our loop terminates.

Lines 1660-1680 bump the prime index by one, and test for hav- ing reached the largest prime of interest. If not, we go back to sift out the next one. If we are finished, lines 1690-1700 restore the D-register to point to true page zero. Line 1710 pops the pointer off the stack, and that's all there is to it!

<<<<listing>>>>

Here is an Applesoft program which will look through the array PRIME produces. Every zero byte in the array indicates a prime number. The value of the prime number at ARRAY+I is I*2+1, since the array only represents odd numbers. This program prints out the value 1 first, which really is not considered a prime number, but it does make the table easier to read.

The program is designed to display 10 8-character fields on a line, which works well on the Apple 80-column screen. I left out the code to print a RETURN after 10 numbers, because the Apple screen automatically goes to the next line.

Line 120 prints out the primes. Delete line 125 if all you want to see is primes. Line 125 prints the largest prime fac- tor of nonprimes, followed by "*" and the other factor (which may not be prime). For example, 16383 is printed as 127*129.

```
100  HIMEM: 24576
110  FOR A = 24576 TO 32767
120  IF  PEEK (A) = 0 THEN
      PRINT RIGHT$("       " + STR$((A - 24576)*2+1),8);
125  IF  PEEK (A) <> 0 THEN
      F1 =  PEEK (A) * 2 + 1
      : F2 = ((A - 24576) * 2 + 1) / F1
      : PRINT RIGHT$("      "+STR$(F1)+"*"+STR$(F2),8);
140  NEXT
```

================================================================
DOCUMENT :AAL-8509:Articles:Problems.65802.txt
================================================================

Problems with 65802's in Apple II+..........Bob Sander-Cederlof

Much to our dismay, we have just learned that some Apple II+ machines will not function properly with a 65802 installed. It is probably the same kind of timing problem that exists with the 65C02 in nearly all Apple II and Apple II+ machines. We had thought the 65802 would work in all II and II+ machines, but it will not. It works in my old II, and one of my II+ machines, but not the other one. We have heard of lots of successful installations, and a few unsuccessful ones. We have not yet heard whether changing to 74F257's will fix things up, as it does with the 65C02.

If you would like to try this exciting enhancement in your Apple, we are selling the 2 MHz 65802 for only $50 (plus $1.50 shipping, and plus 6.125% sales tax if you are in Texas). (The price direct from Western Design Center is still $95 each.) If you want to try it in a II or II+, go ahead and order one; if it turns out to be incompatible, you can send it back for a refund.

I hope we are safe in assuming that anyone who orders such a chip knows how to properly handle, install, and remove CMOS parts. They are extremely sensitive to static electricity, at levels too small for humans to even feel. You can kill them with the voltage generated by moving your arm, if you are wearing a synthetic shirt. You need to be careful, very careful. It is also very easy to bend the leads, or insert the parts backwards. I know, because I have done it. If you want a 65802 but are not confident about the installation, find someone who will do it for you.

=====================================================================
DOCUMENT :AAL-8509:Articles:RainbowProgInfo.txt
=====================================================================

Transfer ProDOS Files to DOS Disks.........Bob Sander-Cederlof

The CONVERT utility program which Apple supplies with ProDOS can be used to transfer ProDOS files to DOS disks, but only if the file type is supported under DOS.  Types BAS, INT, TXT, and BIN can be transferred; other types, such as SYS, cannot.

We needed a program which could transfer any reasonable file.  By reasonable, I mean the file must be able to fit on a floppy, and must not be a sparse file.  Sparse files could also be transferred, but we would need to handle them in a special manner beyond what we have done so far.

The files we need to transfer will be stored on the DOS disk, but they will not actually be used there.  We need them on a DOS disk (either floppy or a harddisk volume) so that a telecommunications downloading system can send them to a ProDOS-based caller.  Since our central system is DOS-based, all the files which are available for downloading must be on DOS disks.

We devised a scheme which should allow the files to be stored under the DOS-based central system, making very few changes to the central system.  We think the only central system change needed will be to use a different menu tree for callers who are calling with ProDOS callup disks.

ProDOS files have three essential descriptive items in addition to the filename:  file type (a single byte), auxiliary file type (two bytes), and end-of-file (three bytes).  DOS files also have a single byte file type, but it only may have one of eight possible values (0, 1, 2, 4, 8, 16, 32, or 64).  DOS files which need the equivalent of the auxiliary file type store it at the beginning of the data area of the file.  DOS files signal end-of-file by a terminal 00 byte, or by adding a two-byte value to the beginning of the data area.

ProDOS filenames are up to 15 characters long.  DOS filenames can be up to 30 characters long.  We decided to use the extra 15 characters in the DOS filename to encode the ProDOS filetype, auxiliary file type, and end-of-file value.  Thus a ProDOS file named "MY.FILE" with a type $FC (BAS), auxiliary type $0801 (this is the starting address), and end-of-file $030F (length of the file) would receive the DOS name of

```
       "MY.FILE        .FC.0801.00030F"
```

All of the data blocks of the ProDOS file will be directly written on DOS sectors, and the DOS file will be arbitrarily classified as a type T file.

When the central telecommunications system downloads this camouflaged ProDOS file, the ProDOS callup disk will first create a TXT file with all the data bytes. When the DOS filename is received, the callup disk will modify the information in the ProDOS directory to the correct file type, auxiliary file type, and end-of-file values.

The transfer utility is easy to use. It runs in the DOS environment, and is invoked by the BRUN command. The first thing the program wants to know is the slot and drive of the floppy disk drive containing the ProDOS diskette with files to be transferred. You will see prompts "PRODOS SLOT:" and "DRIVE:", and should type in the appropriate data:

```
       PRODOS SLOT: 6
             DRIVE: 1
```

Since both slot and drive can only be one digit, the program responds as soon as you have typed a valid digit. You will not need to type a RETURN. If you type ESCAPE instead of a digit, the program will terminate. If you type a backspace (left arrow) you will get another chance to enter both slot and drive.

Once the program knows the slot and drive, it will read in the catalog of the ProDOS in that drive. The file names in the main directory will be displayed, along with the file type, auxiliary file type, and end-of-file values. To the left of each filename the program will display a menu-selection letter. Up to 20 filenames can be listed at once on the Apple screen. At this point you can abort by typing the ESCAPE or the RETURN key, see another screenful of filenames by typing the SPACE key, or select a filename by typing a menu letter.

If the filename you select is a subdirectory file, you will get a new menu displaying the filenames in that subdirectory. In this way you can go down any branch of the tree-structured directory. Typing the SPACE key when there are no more filenames in particular subdirectory returns you to the beginning of the main directory.

If the filename you select is any other file type, that file will be transferred to a DOS disk volume. You have to specify the slot, drive, and volume of the DOS volume. You will be prompted like this:

```
          DOS SLOT: 7
             DRIVE: 1
            VOLUME: 14
```

Both slot and drive always require only one digit, so you do not type return after entering those items. However, volume may be from 1 to 3 digits long. You do have to type RETURN after the volume number. If you type RETURN without entering any volume number, V0 will be assumed. This will be valid for ANY volume number in a floppy drive. You can abort the program by typing ESCAPE instead of any of these numbers, and you can go back to entering DOS SLOT by typing a backspace.

If you are transferring to a hard disk volume V0 indicates that the same volume number should be used as that of the most recent disk access. It is hard to predict what that volume will be. Moral: be specific about the volume number if you are writing to a hard disk volume.

After the file you selected has been transferred, you have the option to quit or to transfer another file. If you want to transfer another file, you have the option to use the same slot/drive/volume numbers, or to change them.

=============================================================================
DOCUMENT :AAL-8509:Articles:Software.65802.txt
=============================================================================

Software Sources for 65802 and 65816.......Bob Sander-Cederlof

Western Design Center reports rising interest among software developers in supporting the new 65802 and 65816 microprocessors.

Since the 65802 chip can be plugged into almost any old Apple, and 65816 co-processor cards are available for Apples, most new software is designed to run in Apples. Of course, the 65802 will also fit in old Ataris and Commodores and even the venerable KIM-1, but these are of lesser interest to AAL.

Four companies have adapted their Apple assemblers to include the new opcodes and addressing modes of the 65802 and 65816.

Of course, you know we have. Last December we released Version 2.0 of the S-C Macro Assembler, and in July we released the ProDOS version. Both of these include full support for the 6502, 65C02 (both standard and Rockwell versions), 65802, and 65816. The DOS version requires at least 48K RAM, and the ProDOS version requires at least 64K.

Other companies supporting the 658xx are Roger Wagner Publishing (Merlin Pro), The Byte Works (ORCA/M 3.5), and Lazerware (Lisa 3.2).

Merlin Pro is the latest version of Merlin, by Glen Bredon. (Big Mac, marketed by Call APPLE, is virtually the same as Merlin, not Merlin Pro.) Merlin Pro will only run in a //c or a //e with at least 128K RAM. In order to assemble the 65C02 additions, you must either be in a //c or in a //e with the enhanced monitor ROM. (If you have an older //e, you must first BRUN a file named MON.65C02.) 65816 support is not complete: the long 24-bit addressing modes were omitted on the premise that these are useless in a 65802 environment. (But what if I am developing code for a co-processor card with a 65816 on it?) The special opcodes in Rockwell's 65C02 are not directly supported, but a file of macro definitions is provided. Merlin Pro does include the capability of producing and linking relocatable object files with external symbols.

Lisa 3.2 is Randy Hyde's latest version of one of the fastest 6502 assemblers around. I have not seen 3.2, but it is reported to support the 65816.

ORCA/M (which is MACRO spelled backwards) was originally published by Hayden Software. They let it go after spending a lot of money promoting it as "the world's best assembler." I remember seeing that claim appear for the first time in Nibble magazine only a few pages away from the same claim in an ad for Nibble's own MicroSparc assembler. Anyway, ORCA/M is now published by The Byte Works, apparently connected more directly with the author (Mike Westerfield). ORCA/M was the first assembler to be revised to support the 65816, and as such Mike had the honor of deciding what some of the assembler rules and syntax would be.

David Eyes, author of the first book on 65816 assembly language, has developed a Pascal P-Code Interpreter which takes advantage of the 65802 features and works with Apple Pascal. (191 Parkview Ave., Lowell, MA 01852)

Starlight Forth Systems has a FIG Forth compatible package for the 802/816, for operation in an Apple. (15247 North 35 St., Phoenix, AZ 85032)

Comlog offers an Applesoft compatible, extended Basic which can be used in an Apple //e equipped with their 65816 co-processor board. (7825 E. Redfield Rd, Scottsdale, AZ 85260)

Manx Software claims to have a 65816 C compiler and assembler under development. (Box 55, Shrewsbury, NJ 07701)

Will Troxell, of MicroMagic, is not only developing a co-processor card for Apples. He is also producing the first operating system for the 65816, which will be similar to Unix.

==============================================================================
DOCUMENT :AAL-8509:DOS3.3:PrintPrimeTable.txt
==============================================================================

```
d£24576nÅA–24576¡32767Mx≠,(A)–0ƒ∫È("
"»‰((A…24576) 2»1),8);ú}≠,(A)—œ0ƒF1–,(A) 2»1:F2–
((A…24576) 2»1)ÀF1:∫È("      "»‰(F1)»"*"»‰(F2),8);¢åÇ
```

```
=====================================================================
DOCUMENT :AAL-8509:DOS3.3:S.65802.Convers.txt
=====================================================================

 1000  *SAVE S.65802.CONVERSIONS
 1010  *--------------------------------
 1020         .OP 65802
 1030  *--------------------------------
 1040  *   CONVERT UP TO 9999, MAX TIME < 80 MSEC
 1050  *      # CYCLES = 8*NUMBER + 44
 1060  *--------------------------------
 1070  CONV.1
 1080         CLC          ENTER 65802 NATIVE MODE
 1090         XCE
 1100         REP #$30     16-BIT MODES
 1110         LDA ##0      START WITH 0000
 1120         LDX 0        GET NUMBER TO BE CONVERTED
 1130         BEQ .2       ...IT IS 0000
 1140         SED          ENTER DECIMAL MODE
 1150         CLC
 1160  .1     ADC ##1      INCREMENT BCD VALUE
 1170         DEX          DECREMENT BINARY VALUE
 1180         BNE .1       ...NOT FINISHED YET
 1190         CLD          BACK TO BINARY MODE
 1200  .2     STA 2        STORE RESULT
 1210         SEC          BACK TO 6502 EMULATION MODE
 1220         XCE
 1230         RTS          RETURN TO CALLER
 1240  *--------------------------------
 1250  *   CONVERT UP TO 65535, MAX TIME < 705 MSEC
 1260  *      # CYCLES = 11*NUMBER +3*INT(NUMBER/10000) + 50
 1270  *--------------------------------
 1280  CONV.2
 1290         CLC          ENTER 65802 NATIVE MODE
 1300         XCE
 1310         REP #$30     16-BIT MODES
 1320         LDA ##0      START WITH 0000
 1330         TAY          CLEAR 10000'S DIGIT
 1340         LDX 0        GET NUMBER TO BE CONVERTED
 1350         BEQ .2       ...IT IS 0000
 1360         SED          ENTER DECIMAL MODE
 1370         CLC
 1380  .1     ADC ##1      INCREMENT BCD VALUE
 1390         BCC .3
 1400         INY          INCREMENT 10000'S DIGIT
 1410         CLC
 1420  .3     DEX          DECREMENT BINARY VALUE
 1430         BNE .1       ...NOT FINISHED YET
 1440         CLD          BACK TO BINARY MODE
 1450  .2     STA 2        STORE RESULT
 1460         SEC          BACK TO 6502 EMULATION MODE
 1470         XCE
 1480         STY 4        STORE 10000'S DIGIT
```

```
1490        RTS          RETURN TO CALLER
1500  *--------------------------------
1510        .LIF
```

```
==============================================================================
DOCUMENT :AAL-8509:DOS3.3:S.BINDEC.txt
==============================================================================

 1000  *SAVE S.BINDEC
 1010  *--------------------------------
 1020  XL     .EQ $00
 1030  XH     .EQ $01
 1040  SL     .EQ $10
 1050  SH     .EQ $11
 1060  *--------------------------------
 1070  BELL   .EQ $FBDD
 1080  RDLINE .EQ $FD6A
 1090  PRBYTE .EQ $FDDA
 1100  COUT   .EQ $FDED
 1110  CROUT  .EQ $FD8E
 1120  *--------------------------------
 1130  T
 1140         JSR RDLINE
 1150         TXA
 1160         BNE .1
 1170         RTS
 1180  .1     JSR CONVERT.DEC.TO.BIN
 1190         LDA XH
 1200         JSR PRBYTE
 1210         LDA XL
 1220         JSR PRBYTE
 1230         LDA #"="
 1240         JSR COUT
 1250         JSR CONVERT.BIN.TO.DEC
 1260         JSR CROUT
 1270         JMP T
 1280  *--------------------------------
 1290  CONVERT.DEC.TO.BIN
 1300         LDX #0
 1310         STX XL       least significant byte
 1320  *--    STX XI    ---ANY INTERMEDIATE BYTES---
 1330         STX XH       most significant byte
 1340  .1     LDA $200,X
 1350         EOR #"0"
 1360         CMP #10
 1370         BCS .3       ...END OF NUMBER
 1380         TAY          SAVE CURRENT DIGIT
 1390         LDA XL
 1400         STA SL
 1410  *--    LDA XI    ---ANY INTERMEDIATE BYTES---
 1420  *--    STA SI    ---FOLLOW THIS PATTERN------
 1430         LDA XH
 1440         JSR SHIFT.X
 1450         BCS .2       ...OVERFLOW
 1460         JSR SHIFT.X
 1470         BCS .2       ...OVERFLOW
 1480         STA SH
```

```
1490        CLC
1500        LDA XL
1510        ADC SL
1520        STA XL
1530  *--    LDA XI    ---ANY INTERMEDIATE BYTES---
1540  *--    ADC SI    ---FOLLOW THIS PATTERN------
1550  *--    STA XI    ----------------------------
1560        LDA XH
1570        ADC SH
1580        BCS .2       ...OVERFLOW
1590        JSR SHIFT.X
1600        BCS .2       ...OVERFLOW
1610        STA XH
1620        INX          SCAN TO NEXT DIGIT
1630        TYA          GET DIGIT
1640        ADC XL       LEAST SIGNIFICANT BYTE
1650        STA XL
1660        BCC .1       ...NO CARRY
1670  *--    INC XI    ---ANY INTERMEDIATE BYTES---
1680  *--    BNE .1    ---FOLLOW THIS PATTERN------
1690        INC XH       MOST SIGNIFICANT BYTE
1700        BNE .1       ...UNLESS OVERFLOW
1710  .2    JSR BELL     SIGNAL OVERFLOW
1720  .3    RTS
1730  *---------------------------------
1740  SHIFT.X
1750        ASL XL       LEAST SIGNIFICANT BYTE
1760  *--    ROL XI    ---ANY INTERMEDIATE BYTES---
1770        ROL          ...MOST SIGNIFICANT BYTE IN A
1780        RTS
1790  *---------------------------------
1800  CONVERT.BIN.TO.DEC
1810        LDX #0       DIGIT COUNTER
1820  *---DIVIDE BY TEN----------------
1830  .1    LDA #0
1840        LDY #16      2*(# Bytes being converted)
1850  .2    CMP #5
1860        BCC .3
1870        SBC #5
1880  .3    ROL XL
1890  *--    ROL XI    ---ANY INTERMEDIATE BYTES---
1900        ROL XH
1910        ROL
1920        DEY
1930        BNE .2
1940        PHA          SAVE DIGIT ON STACK
1950        INX          COUNT THE DIGIT
1960  *---NEXT DIGIT-------------------
1970        LDA XL
1980  *--    ORA XI    ---ANY INTERMEDIATE BYTES---
1990        ORA XH
2000        BNE .1
2010  *---PRINT DECIMAL----------------
2020  .4    PLA
```

```
2030        ORA #"0"
2040        JSR COUT
2050        DEX
2060        BNE .4
2070        RTS
2080  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8509:DOS3.3:S.Init.Dos.PDos.txt
==============================================================================

 1000  *SAVE S.INIT DOS & PRODOS
 1010  *--------------------------------
 1020  DOS.LOW.TRACK .EQ $12    DOS $12...$22
 1030  DOS.VOLUME    .EQ 1
 1040  SLOT          .EQ 6
 1050  DRIVE         .EQ 1
 1060  *--------------------------------
 1070  PRODOS.MAX.BLOCKS .EQ DOS.LOW.TRACK*8
 1080  *--------------------------------
 1090  ACTUAL.DOS.SECTORS   .EQ DOS.LOW.TRACK>$11+34-DOS.LOW.TRACK*16
 1100  ACTUAL.PRODOS.BLOCKS .EQ DOS.LOW.TRACK<$12+DOS.LOW.TRACK-2*8+1
 1110  *--------------------------------
 1120  DOS.WARM.START .EQ $03D0
 1130  RWTS       .EQ $03D9
 1140  GETIOB     .EQ $03E3
 1150  *--------------------------------
 1160  R.PARMS    .EQ $B7E8
 1170  R.SLOT16   .EQ $B7E9
 1180  R.DRIVE    .EQ $B7EA
 1190  R.VOLUME   .EQ $B7EB
 1200  R.TRACK    .EQ $B7EC
 1210  R.SECTOR   .EQ $B7ED
 1220  R.BUFFER   .EQ $B7F0,B7F1
 1230  R.OPCODE   .EQ $B7F4
 1240  R.ERROR    .EQ $B7F5
 1250  *--------------------------------
 1260  MON.CROUT  .EQ $FD8E
 1270  MON.PRBYTE .EQ $FDDA
 1280  MON.COUT   .EQ $FDED
 1290  *--------------------------------
 1300         .OR $803
 1310  *--------------------------------
 1320  DOUBLE.INIT
 1330         JSR FORMAT.35.TRACKS
 1340         LDA #INIT.BUFFER
 1350         STA R.BUFFER
 1360         LDA /INIT.BUFFER
 1370         STA R.BUFFER+1
 1380         JSR BUILD.DOS.CATALOG
 1390         JSR BUILD.PRODOS.CATALOG
 1400  *---WRITE BOOT PROGRAM-----------
 1410         LDA #BOOTER
 1420         STA R.BUFFER
 1430         LDA /BOOTER
 1440         STA R.BUFFER+1
 1450         JSR CLEAR.INIT.BUFFER
 1460         LDA #0
 1470         STA R.TRACK
 1480         STA R.SECTOR
```

```
1490        JMP CALL.RWTS
1500 *--------------------------------
1510 FORMAT.35.TRACKS
1520        LDA #SLOT*16
1530        STA R.SLOT16
1540        LDA #DRIVE
1550        STA R.DRIVE
1560        LDA #DOS.VOLUME
1570        STA R.VOLUME
1580        STA V.VOLUME
1590        LDA #$04     INIT OPCODE FOR RWTS
1600 CALL.RWTS.OP.IN.A
1610        STA R.OPCODE
1620 CALL.RWTS
1630        JSR GETIOB
1640        JSR RWTS
1650        BCS .1       ERROR
1660        RTS
1670 .1     LDY #0       PRINT "ERROR"
1680 .2     LDA ERMSG,Y
1690        BEQ .3
1700        JSR MON.COUT
1710        INY
1720        BNE .2       ...ALWAYS
1730 .3     LDA R.ERROR  GET ERROR CODE
1740        JSR MON.PRBYTE
1750        JSR MON.CROUT
1760        JMP DOS.WARM.START
1770 *--------------------------------
1780 ERMSG  .HS 8D87
1790        .AS -/RWTS ERROR /
1800        .HS 00
1810 *--------------------------------
1820 BUILD.DOS.CATALOG
1830        JSR CLEAR.INIT.BUFFER
1840        LDA #17
1850        STA R.TRACK
1860        LDA #0
1870        STA R.SECTOR
1880 *---BUILD GENERIC VTOC-----------
1890        LDY #VTOC.SZ-1
1900 .0     LDA VTOC,Y
1910        STA INIT.BUFFER,Y
1920        DEY
1930        BPL .0
1940        LDA #DOS.VOLUME
1950        STA V.VOLUME
1960 *---PREPARE BITMAP---------------
1970        LDY #4*34
1980        LDA #$FF
1990 .1     CPY #4*17    ARE WE ON CATALOG TRACK?
2000        BEQ .2
2010        CPY #4*DOS.LOW.TRACK
2020        BCC .3            IN PRODOS ARENA
```

```
2030        STA V.BITMAP+1,Y
2040        STA V.BITMAP,Y
2050 .2     DEY
2060        DEY
2070        DEY
2080        DEY
2090        BNE .1
2100 *---WRITE VTOC ON NEW DISK-------
2110 .3     LDA #2           RWTS WRITE OPCODE
2120        JSR CALL.RWTS.OP.IN.A
2130 *---WRITE CATALOG CHAIN----------
2140        JSR CLEAR.INIT.BUFFER
2150        LDA #17     TRACK 17
2160        LDY #15     START IN SECTOR 15
2170        STA C.TRACK
2180 .4     STY R.SECTOR
2190        DEY
2200        STY C.SECTOR
2202        BNE .5
2203        STY C.TRACK  TERMINATE THE CHAIN
2210 .5     JSR CALL.RWTS
2220        LDY C.SECTOR
2230        BNE .4
2240        RTS
2250 *--------------------------------
2260 VTOC   .HS 04.11.0F.03.00.00.01
2270        .AS "COMBINATION DOS/PRODOS DATA DISK"
2280        .HS 7A
2290        .AS /07-25-85/
2300        .HS 11.01.00.00.23.10.00.01
2310 VTOC.SZ .EQ *-VTOC
2320 *--------------------------------
2330 BUILD.PRODOS.CATALOG
2340        LDA #0
2350        STA R.TRACK
2360        JSR CLEAR.INIT.BUFFER
2370 *--------------------------------
2380        LDA #5           SECTOR 5 = BLOCK 5
2390        STA R.SECTOR     BACK LINK = 0004
2400        LDA #4            FWD LINK = 0000
2410        STA INIT.BUFFER
2420        JSR CALL.RWTS
2430 *--------------------------------
2440        LDA #7           SECTOR 7 = BLOCK 4
2450        STA R.SECTOR     BACK LINK = 0003
2460        DEC INIT.BUFFER   FWD LINK = 0005
2470        LDA #5
2480        STA INIT.BUFFER+2
2490        JSR CALL.RWTS
2500 *--------------------------------
2510        LDA #9           SECTOR 9 = BLOCK 3
2520        STA R.SECTOR     BACK LINK = 0002
2530        DEC INIT.BUFFER   FWD LINK = 0004
2540        DEC INIT.BUFFER+2
```

```
2550        JSR CALL.RWTS
2560 *--------------------------------
2570        LDA #11           SECTOR 11 = BLOCK 2
2580        STA R.SECTOR      BACK LINK = 0000
2590        LDY #HDR.SZ-1      FWD LINK = 0003
2600 .1     LDA HEADER,Y
2610        STA INIT.BUFFER,Y GET VOLUME HEADER
2620        DEY
2630        BPL .1
2640        LDA #PRODOS.MAX.BLOCKS
2650        STA INIT.BUFFER+$29
2660        LDA /PRODOS.MAX.BLOCKS
2670        STA INIT.BUFFER+$2A
2680        JSR CALL.RWTS
2690 *--------------------------------
2700        LDA #3
2710        STA R.SECTOR
2720        JSR CLEAR.INIT.BUFFER
2730        LDA #$FF
2740        LDY #DOS.LOW.TRACK-1
2750 .2     CPY #17      SKIP OVER DOS CATALOG TRACK
2760        BEQ .3
2770        STA INIT.BUFFER,Y
2780 .3     DEY
2790        BPL .2
2800        LDA #1       MAKE ONLY BLOCK 7 AVAILABLE
2810        STA INIT.BUFFER   IN TRACK 0
2820        JMP CALL.RWTS
2830 *--------------------------------
2840 HEADER .DA 0,3,#$F0+VNSZ
2850 VN     .AS /DATA/
2860 VNSZ   .EQ *-VN
2870        .BS 15-VNSZ
2880        .HS 00.00.00.00.00.00.00.00.00.00.00.00
2890        .HS 00.00.C3.27.0D.00.00.06.00.08.00
2900 HDR.SZ .EQ *-HEADER
2910 *--------------------------------
2920 CLEAR.INIT.BUFFER
2930        LDY #0
2940        TYA
2950 .1     STA INIT.BUFFER,Y
2960        INY
2970        BNE .1
2980        RTS
2990 *--------------------------------
3000 BOOTER
3010        .PH $800
3020 BOOTER.PHASE
3030        .HS 01
3040        LDA $C088,X  MOTOR OFF
3050        LDY #0
3060 .1     LDA MESSAGE,Y
3070        BEQ .2
3080        JSR $FDF0
```

```
3090        INY
3100        BNE .1
3110 .2     JMP $FF59
3120 *--------------------------------
3130 MESSAGE
3140        .HS 8D8D8787
3150        .AS -"COMBINATION DOS/PRODOS DATA DISK"
3160        .HS 8D8D8787
3170        .AS -/NO DOS IMAGE ON THIS DISK/
3180        .HS 8D8D00
3190        .EP
3200 *--------------------------------
3210 INIT.BUFFER .BS 256
3220 *--------------------------------
3230 V.VOLUME   .EQ INIT.BUFFER-$BB+$C1
3240 V.BITMAP   .EQ INIT.BUFFER-$BB+$F3
3250 *--------------------------------
3260 C.TRACK        .EQ INIT.BUFFER+1
3270 C.SECTOR       .EQ INIT.BUFFER+2
3280 *--------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8509:DOS3.3:S.SF802PrmPlus.txt
=============================================================================

1000        .OP 65816
1010 *SAVE S.SUPER-FAST PRIMES 65802+
1020        .OR $8000    SAFELY OUT OF WAY
1030 *--------------------------------
1040 BASE   .EQ $6000    PRIME ARRAY $6000...7FFF
1050 BEEP   .EQ $FF3A    BEEP THE SPEAKER
1060 COUNT  .EQ 0,1
1070 *--------------------------------
1080 *      MAIN CALLING ROUTINE
1090 *--------------------------------
1100 MAIN   JSR BEEP
1110        CLC          ...ENTER NATIVE MODE
1120        XCE
1130        REP #$20     A/16, XY/8
1140        LDA ##1000     DO IT 1000 TIMES
1150        STA COUNT
1160 .1     JSR PRIME
1170        DEC COUNT
1180        BNE .1
1190        SEC          ...ENTER EMULATION MODE
1200        XCE
1210        JMP BEEP     SAY WE'RE DONE
1220 *--------------------------------
1230 *      PRIME ROUTINE
1240 *      SETS ARRAY STARTING AT BASE
1250 *      TO $FF IF NUMBER IS NOT PRIME
1260 *      CHECKS ONLY ODD NUMBERS > 3
1270 *      INC = INCREMENT OF KNOCKOUT
1280 *      N = KNOCKOUT VARIABLE
1290 *--------------------------------
1300 PRIME
1310        TSX          SAVE STACK PNTR
1320        LDY #0       256 * 16 * 2 = 8192 BYTES
1330        LDA ##BASE+8191   BASE...BASE+8191
1340        TCS          TEMPORARY STACK PNTR
1350 .1     .HS 0B.0B.0B.0B.0B.0B.0B.0B  ...16 PHD'S
1360        .HS 0B.0B.0B.0B.0B.0B.0B.0B
1370        DEY          256 TIMES
1380        BNE .1
1390        TXA
1400        ORA ##$0100  RESTORE STACK PNTR
1410        TCS
1420 *--------------------------------
1430        LDA ##BASE+4  POINT AT FIRST PRIME-SQUARED
1440        PHA               (WHICH IS 3*3=9)
1450        LDX #1       POINT AT FIRST PRIME (3)
1460        BNE .4       ...ALWAYS
1470 *--------------------------------
1480 .2     TXA
```

```
        ASL
        ASL          *4, CLEARS CARRY TOO
        ADC 1,S      TO PNTR
        STA 1,S
        LDY BASE,X   GET A POSSIBLE PRIME
        BNE .8       THIS ONE HAS BEEN KNOCKED OUT
.4      TXA
        ASL          INC = START*2 + 1
        INC
        TAY
        STY .7+1
        LDA 1,S      MOVE MULT TO N
*---STRIKE OUT MULTIPLES---------
.5      SEP #$20     A/8
.6      TCD
        STX 0
.7      ADC #*-*
        BCC .6
        REP #$20     A/16
        ADC ##$FF    APPLY CARRY
        BPL .5       ...NOT FINISHED
*--------------------------------
.8      INX
        CPX #64      UP TO 127
        BCC .2       WE'RE DONE IF X>127
        LDA ##0
        TCD
        PLA
        RTS
*--------------------------------
```

```
===========================================================================
DOCUMENT :AAL-8509:DOS3.3:S.SFast802Prm.txt
===========================================================================

 1000          .OP 65816
 1010  *SAVE S.SUPER-FAST PRIMES 65802
 1020          .OR $8000    SAFELY OUT OF WAY
 1030  *---------------------------------
 1040  BASE    .EQ $6000    PRIME ARRAY $6000...7FFF
 1050  BEEP    .EQ $FF3A    BEEP THE SPEAKER
 1060  COUNT   .EQ 0,1
 1070  *---------------------------------
 1080  *       MAIN CALLING ROUTINE
 1090  *---------------------------------
 1100  MAIN    JSR BEEP
 1110          CLC          ...ENTER NATIVE MODE
 1120          XCE
 1130          REP #$20     A/16, XY/8
 1140          LDA ##1000     DO IT 1000 TIMES
 1150          STA COUNT
 1160  .1      JSR PRIME
 1170          DEC COUNT
 1180          BNE .1
 1190          SEC          ...ENTER EMULATION MODE
 1200          XCE
 1210          JMP BEEP     SAY WE'RE DONE
 1220  *---------------------------------
 1230  *       PRIME ROUTINE
 1290  *---------------------------------
 1300  PRIME
 1310          TSX          SAVE STACK PNTR
 1320          LDY #0       256 * 16 * 2 = 8192 BYTES
 1330          LDA ##BASE+8191   BASE...BASE+8191
 1340          TCS          TEMPORARY STACK PNTR
 1350  .1      .HS 0B.0B.0B.0B.0B.0B.0B.0B  ...16 PHD'S
 1360          .HS 0B.0B.0B.0B.0B.0B.0B.0B
 1370          DEY          256 TIMES
 1380          BNE .1
 1390          TXA
 1400          ORA ##$0100  RESTORE STACK PNTR
 1410          TCS
 1420  *---------------------------------
 1430          LDA ##BASE+4  POINT AT FIRST PRIME-SQUARED
 1440          PHA               (WHICH IS 3*3=9)
 1450          LDX #1       POINT AT FIRST PRIME (3)
 1460          BNE .4       ...ALWAYS
 1470  *---------------------------------
 1480  .2      TXA
 1490          ASL
 1500          ASL          *4, CLEARS CARRY TOO
 1510          ADC 1,S      ADD TO PREVIOUS PNTR
 1520          STA 1,S      PNTR TO SQUARE OF ODD NUMBER
 1530          LDY BASE,X   GET A POSSIBLE PRIME
```

```
1540        BNE .8       THIS ONE HAS BEEN KNOCKED OUT
1550  .4    TXA
1560        ASL          DELTA = START*2 + 1
1570        INC
1580        STA .7+1
1590        LDA 1,S      PNTR TO SQUARE OF PRIME
1600  *---STRIKE OUT MULTIPLES---------
1610  .6    TCD          POINT AT MULTIPLE
1620        STX 0        STORE NON-ZERO AS FLAG
1630  .7    ADC ##*-*    (VALUE FILLED IN)
1640        BPL .6       ...NOT FINISHED
1650  *---NEXT ODD NUMBER--------------
1660  .8    INX
1670        CPX #64      UP TO 127
1680        BCC .2       WE'RE DONE IF X>127
1690        LDA ##0      RESTORE DIRECT PAGE REGISTER
1700        TCD
1710        PLA          POP PNTR OFF STACK
1720        RTS
1730  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8510:Articles:Another65C02Fix.txt
==============================================================================

A Different Patch for 65C02 & Old Apples...William O'Ryan Jr.

Since my earlier letter (Jun 84) on the 65C02 and the Apple II+ I was interested and gratified to read Andrew Jackson's (Dec 84) and Jim Sather's (Mar 85) letters on the same subject. However, two things began to worry me. First, the smallness of the time gain in the F257 chips (around 7 nanoseconds, I understand). That did not seem enough to be very reliable. Second, a friend in town has an Apple whose speed was not sufficiently improved to allow the 65C02 to work (although there was some noticeable improvement).

After reading the first few chapters of Jim Sather's book, "Understanding the Apple II", I was able to come up with a new solution. As I figure it, this new solution yields an improvement of around 70 nanoseconds, more than enough. Simply put, just replace the -RAS line inputs to the 74LS174 chips at B5 and B8 with AX. AX rises 70 nsec earlier than -RAS, enabling those chips to latch RAM output 70 nsec earlier. It is a simple patch and may be done either with or without altering the motherboard.

I tried it first without altering my motherboard, on a Rev 44-1 Apple using 200 nsec 16K RAM chips. I was surprised to see it work, as I had expected that 200 nsec RAM chips would be too slow for the patch. (I haven't tried it yet with 250 nsec RAM chips.) Actually, this particular Apple did not need any speed-up -- the 65C02 was already working in it.

To do this patch: remove the chips at B5 and B8; seat an extra socket under each of them; pin 9 on these sockets should be bent out so they do not go into the motherboard sockets; remove the chip at C2 and put an extra socket under it; connect a wire from pin 14 of the C2 socket to the bent out pins 9 of B5 and B8. Pin 14 of the 74S195 at C2 is a source of the AX signal; pin 9 of B5 and B8 was previously connected to -RAS.

<<<picture>>>

I have another Apple (Rev 4) which has 24 150 nsec 64K RAM chips (using the Cramapple mod). This Apple already had F257's in it with a 65C02. I put the old LS257's back in, and sure enough the 65C02 began to stumble. Then I removed the motherboard and on the underside cut the trace to -RAS and soldered in a jumper wire to pin 14 of C2. It worked perfectly!

<<<picture>>>

Naturally those who try any of these patches do so at their own risk.

I must thank Jim Sather for his book; it was only by studying the timing diagrams in that book and staring at the circuit diagram published by Apple that I was able to do this. I hope some of the hardware types will be able to tell me if I have built a time bomb. I am also very interested to hear whether the problem with the 65802 is the same.

Jumper wires

Pins 9 not
plugged into RAS

View from top front

74LS195

74LS174

74LS174

AX

Underside of
motherboard
viewed from rear

Jumper wire

Cut
trace here

AX

RAS

RAM

=================================================================
DOCUMENT :AAL-8510:Articles:Apple.Manuals.txt
=================================================================

Apple Manuals from Addison-Wesley...................Bill Morgan

Those elusive Apple technical manuals are finally coming out of hiding! As we reported some months ago, Addison-Wesley is beginning to distribute Apple's manuals, and we can now supply them for you. The ones we have seen are at least as good as Apple's own editions, and in some cases better.

Here are the titles that we can order for you:

Applesoft Tutorial - $29.95, disk. Beginner's introduction to Applesoft, with a disk of examples.

Applesoft BASIC Programmer's Reference Manual - $22.95, 373+xxv pages. Complete reference manual for Applesoft, documenting all features with many examples.

BASIC Programming with ProDOS - $29.95, 264+xxix pages, disk Covers using ProDOS from BASIC, including command and file handling. The disk includes lots of examples, and the useful Applesoft Programmer's Assistant program, which includes RENUMBER, MERGE, AUTOmatic line numbering, REM deletion, variable cross reference, and other features.

And here are the ones that look most important, that we expect to keep in stock here at S-C:

Apple //e Technical Reference manual - $24.95, 409+xxxii pages. Here's Apple's documentation of all the internals of the //e, including I/O devices and firmware, memory organi- zation, the System Monitor, peripheral-card programming, the Super Serial Card, and hardware implementation. The new edition includes all the new features of the Enhanced //e and a complete source listing of the ROMs. This book is essential for serious //e programming.

Apple //c Technical Reference Manual - $24.95. And here is the same detailed coverage of the //c, and more. Additional topics documented in this book are the built-in serial I/O ports, the mouse input, and interrupt handling. If you want to use these features of the //c, get this book.

ProDOS Technical Reference Manual- $29.95, 186+xvii pages, disk. This is the official book on ProDOS, covering files, MLI calls, System programs, interrupt handling, and more. The disk is the ProDOS Exerciser, which allows you to experiment with all of the MLI calls without writing special programs. This book completes a ProDOS programmer's reference shelf, along with Beneath Apple ProDOS, and Apple ProDOS: Advanced Features.

The //e manual was scheduled for July publication:  we just received it and the ProDOS manual today.  The //c manual is scheduled for November delivery:  we'll accept orders and ship the book as soon as A-W comes through.

Many thanks to Apple and to Addison-Wesley for making these important documents so easily available.

```
=============================================================================
DOCUMENT :AAL-8510:Articles:ErvEdgeExecFile.txt
=============================================================================

MON I
CALL -151
9D26:B2 B6
9D3E:DC A5
9FA8:CA
9FC5:A9 A0 2C A9 8D 6C 36 00
A710:CA
A186:AC 5F AA B9 1F 9D 48 B9
    :1E 9D 48 60 EA
A56E:4C DD A5 A5 EC 20 CA 9F
    :4C C5 9F EA
A5DD:AD 75 AA 85 EE 8E 75 AA
    :A9 0D 4C AA A2 EA
A921:60 70
A929:60
AAE3:9A AD
AB10:C9 0E
A4F0:A9 A0 BE C8 B4 10 02 A9
    :AA 4C CA 9F
A9FD:D2 2C 06 E0 30 03 4C 24
    :ED 4C 1B E5
A021:EA EA EA
AA2C:C9 A0 F0 0C A0 A0 CC 76
    :AA F0 03 4C C4 A6 C5 EC
    :60 EA EA
AD98:20 84 A8 20 DC AB D0 57
    :4C F4 AD EA EA 20 84 A8
    :20 84 A8 20 38 AE
ADAE:20 2F AE A2 0C BD AE B3
    :20 CA 9F CA D0 F7 20 69
    :BA 20 2F AE 20 2F AE
ADC5:18 90 04 EA 20 84 A8 20
    :11 B0 B0 5B A2 00 8E 9C
    :B3 BD C6 B4 F0 51 30 48
ADDD:BD C8 B4 0A A0 07 0A B0
    :03 88 D0 FA
ADE9:B9 A7 B3 85 EC A0 1E 84
    :EF D0 4B
ADF4:20 84 A8 20 F7 AF 90 AF
ADFC:EA EA AC 9C B3 20 F0 A4
    :20 71 A5
AE07:BE E7 B4 B9 E8 B4 20 FE
    :A9 A0 07 84 24 AE 9C B3
AE17:BD C9 B4 20 DA B6 E8 C6
    :EF D0 F5 20 2F AE
AE25:20 30 B2 90 A9 B0 A0 4C
    :7F B3
AE2F:C6 EB D0 09 20 8D B7 F0
    :F4 A9 15 85 EB 4C C8 9F
```

```
AE3F:A5 EE C9 DE F0 07 20 2C
    :AA F0 B4 D0 D9
AE4C:84 ED A0 01 C6 ED 30 D1
    :CA 88 D0 FC C8 E8 B9 75
    :AA C9 A0 F0 9D DD C9 B4
    :F0 F2 E8 D0 E7 EA
BA69:86 44 86 45 A0 C8 B9 F2
    :B3 0A 90 06 E6 44 D0 F9
    :E6 45 D0 F5 88 D0 EF A6
    :44 A5 45 4C FE A9
BA87:20 A8 FC C6 55 D0 82 A9
    :4F 85 55 4C C8 9F EA
B6B3:20 A3 A2 20 8E BA 20 8C
    :A6 F0 14 C9 8D F0 F4 20
    :DA B6 A5 F1 20 87 BA AD
    :00 C0 10 EA 8D 10 C0 20
    :8D B7 D0 E2 4C FC A2
B6DA:A8 10 08 C9 A0 B0 0E 24
    :EA 30 0A 46 32 46 32 29
    :3F 69 1F 49 E0
B6EF:C9 E0 90 02 29 FF 20 CA
    :9F A9 FF 85 32 60
B78D:20 0C FD C9 91 60
B3AF:BD C5 C3 C1 D0 D3 A0 C5
    :C5 D2 C6 A0
A884:A9 A0 85 EE 18 60
A88A:49 4E 49 D4 4C 4F 41 C4
    :53 41 56 C5 52 55 CE 54
    :59 50 C5
A89D:44 45 4C 45 54 C5 4C 4F
    :43 CB 55 4E 4C 4F 43 CB
A8AD:43 4C 4F 53 C5 52 45 41
    :C4 45 58 45 C3 57 52 49
    :54 C5
A8BF:44 49 D2
9FFB:B9 8A A8
9FED:59 8A A8
48:04 N 3D0G
NOMON I
```

```
=========================================================================
DOCUMENT :AAL-8510:Articles:ErvEdgeWildcat.txt
=========================================================================
```

WildCAT for DOS 3.3....................................Erv Edge
Anchorage, Alaska

WildCAT is a series of patches to DOS 3.3 which modify the CATALOG command.  The new features include:

* A catalog by "wildcard" FILENAME facility.
* A catalog by FILETYPE facility.
* An alternate, short-form: either DIR or CAT.
* Catalog free space patch.
* Ctrl-Q catalog abort.
* TYPE a random or sequential text file.

Lee Reynold's FILEDUMP command has been re-packaged and re-presented as TYPE (see Call-A.P.P.L.E.  6/82 p47).  More on this later. WildCAT, along with TYPE, is an attempt to teach new tricks to an old dog, as it were.

The normal DOS catalog command allows slot, drive, and volume parameters.  I have added a filename parameter, but it is processed a little differently than the way file names are usually processed.  To display the catalog entries for all files whose names contain a particular string, type any of the folowing:

```
CATALOG ^string [,Dn] [,Sn] [,Vn]
DIR ^string [,Dn] [,Sn] [,Vn]
CAT ^string [,Dn] [,Sn] [,Vn]
```

where "^string" begins with the "^" or caret symbol (shifted N on the ][+ or shifted 6 on the //e); the string should contain no blanks, although it may "end" with them; the string would normally end with a carriage return or with a comma if a drive or slot number is specified.  Only those files that contain the "string" somewhere in the filename will be listed.  (Of course you already know that the D, S, and V parameters are shown in brackets above because they are optional; you do not type the brackets.)

For example, "CATALOG ^TEST" would list each file with 'TEST' as part of the filename; while "DIR ^PAY." would list those with 'PAY.' somewhere in the filename; and "CAT^.OBJ,D2" would list filenames on drive 2 that contain the partial string '.OBJ'.  "CAT" and "DIR" are simply synonyms for "CATALOG".

I have also arranged things so you can list the catalog entries of a specified file-type.  You simply type the file taype code in the CATALOG command:

```
CATALOG t [,Dn] [,Sn] [,Vn]
DIR t [,Dn] [,Sn] [,Vn]
```

```
   CAT t [,Dn] [,Sn] [,Vn]
```

where "t" is any of the unadorned, single-letter filetypes: A B I R S T. Only that type of file (if present) will be listed.

For example, "CATALOG T" would list all the text files; "DIR A,D2" would list all of the Applesoft files on drive 2; "CAT B,S5,D1" would list all the binary files on slot 5, drive 1. Yes, "DIRT" works just fine.

I added the TYPE command, which allows you to display the contents of text files. Both CATALOG and TYPE will optionally:

1. Print "hidden" control characters as inverse:
   POKE 234,0 to print as inverse (default)
   POKE 234,255 to function as-is

2. Lower case letters may be shifted to upper case:
   POKE -18700,255 no shift (default)
   POKE -18700,223 to shift lower to upper case.

You can slow down TYPE's output via SPEED=xx or POKE 241,xx; it can be paused by pressing any key; then Ctrl-Q to abort. Also, TYPE pauses and waits for a keypress when it encounters a hex 00 imbedded in the file or at end of file; press Ctrl-Q to quit. Random text files may be TYPE'd by holding down REPT-SPACE to get past the hex 00's at the end of each logical record.

The listing that follows is intended for information only: it is not BRUNable. My intention is that you prepare the EXEC shown below to actually install the patches. Any word processor that produces a straight, sequential text file may be used to prepare the EXEC. Of course you can also use the S-C Macro Assembler for this purpose. Then, type EXEC WILDCAT to apply the patches to DOS 3.3 in memory. After checking it out and running any other tests you like, put in a new diskette, enter a HELLO program, and type INIT HELLO to "permanently" install WildCAT in the DOS on tracks 0, 1, and 2.

When I wrote WildCAT, I had two main goals in mind: it should be a (mostly in-place) code replacement, and it should be compatible with the known means of using (abusing?) the existing CATALOG code at $AD98-AE69.

One major design consideration was a mechanism for entering the ^string/type parameter. This required merely changing the "keyword parameter table" to allow CATALOG to have a "filename".

Next, a distinction had to be made between a "wildcard" and a "filetype" parameter. It made sense to 'delimit' the wildcard string; then the single-character filetype would be just that: a single character, entered without a delimiter. But this "phony" name mechanism has it's own problems:

First, "What's in a Name?" (DOS Manual p. 16): it has to start with a letter...which automatically eliminates most special characters (eg, equal, pound, slash, colon, etc) as the delimiter. The command parsing routine doesn't really know what it's working on at the time. All it knows is: if a name may be present, it must be valid. The validity test is only that the first character be equal to or greater that $C0 or an @-sign. The @-sign could have been used, but it's a problem on some 80-column boards; the ^ or caret works nicely (and besides, it looks good).

Second, now that we have a name (however, phony) and since the CATALOG command lives in the File Manager (FM) portion of DOS, there will be a buffer allocated for it. Unfortunately, the Command Interpreter (CI) DOCAT routine, which calls the FM, already "knows" that there will not really be a name, so it does not include housekeeping code to deallocate a buffer. So merrily allocating files without closing them...after the third time: NO BUFFERS AVAILABLE. And naively adding CLOSE (even if there were room for it), would have one very undesirable side effect if a "regular" catalog were requested: CATALOG-CLOSE without FNAME means close all open files. WildCAT, instead, plays a little shell game with DOS: The new DOCAT routine saves the first character of FNAME and substitutes a zero. Thereafter, neither the File Manager nor the rest of DOS ever knows that a name has been entered, and a buffer is never actually allocated.

Third, what really should happen if a phony name is not entered? A regular catalog, of course, but how would this be indicated to WildCAT? Well, the shell game has a sting. Early on when the CI PARSE routine discovers that a filename is a valid parameter, it first clears FNAME to all blanks, expecting to fill it in with whatever comes in next. If a comma or carriage return comes in next, then FNAME still contains the blank; and that's what WildCAT saves off (under the shell) before it substitutes the zero.

Thus, the "sting" is that the CI "tricks" itself into telling WildCAT what to do in the absence of a ^string/type specification: WildCAT takes a blank to indicate "do a regular" catalog; just as positively as a "^" indicates "do a wildcard" catalog, and a single, undelimited character indicates "do a filetype" catalog.

The blank indicator also helps satisfy the second goal above and solves the problem of compatibilty with the "known means" of using/abusing the existing CATALOG code. WildCAT simply has to put a blank under the shell at each of the points where the code could most reasonably be entered without going thru the Command Interpreter's new DOCAT routine. That's exactly what all the JSR's to the routine AllowENTry are doing.

Satisfying that second goal takes up considerable space, however; and has somewhat undermined the first constraint: WildCAT certainly isn't "in-place" in one place! And I apologize for this rather bizarre, serpentine code; I do hope that now you understand why some things were done the way they were.

Although considerable effort was spent to maintain compatibilty with the existing DOS commands, there were some compromises:

1. While the DOS manual (page 22) states: "To specify drive 1, you use the notation D1 separated from the file name by a comma", you can in fact leave out the comma between CATALOG and D1. With WildCAT that comma is now required; otherwise, it would take the "D" as a filetype and try to find it...which of course it wouldn't and there would be no files reported. This would also be a problem for Applesoft programs that do something like: PRINT D$"CATALOG D1" without the separating comma. Therefore, WildCAT issues a (late) "SYNTAX ERROR" message if it encounters an undelimited string of length 2 or more.

2. CATALOG is a favorite routine to execute directly, bypassing the DOS Command Interpreter. FID, for example, provides its CATALOG via the "external" entry to the File Manager, which means that the main entry at CATHNDLR must provide for a "regular" catalog. It is also possible from machine language, however, to bypass both the CI and the FM. This usually involves changing the exit JMP address at DONEXT2 (to return to the user's code) and then jumping directly into almost anywhere in the CATALOG code (see the Listing 1 labels that begin "at"). I believe most of these cases are covered, but you may find some programs, which provide an "internal" CATALOG, that just won't work with WildCAT.

3. In order to both gain some patch space and provide the DIR/CAT short-form command name, the DOS command POSITION was eliminated. You may have to read about it just to find out that it is, much less what it is. Its relative lack of use may be due to its implementation: it, like APPEND, finds its way through the file one byte at a time...all day long. Any program that uses it will now get a syntax error. If POSITION is really needed, it can be readily simulated by programming a read-loop to discard N-1 fields before processing the desired Nth field.

The following is a brief commentary on the assembly listing. The paragraph numbers correspond to comment numbers in the listing.

The page zero locations I used ($EB thru $EF) are free, i.e. not used by DOS, the Monitor, or the Basics.

(1) In CMDTBL, replace Integer CHAIN address with TYPE and DOCAT address with NewDOCAT.

(2) Rearrange some code (and change the two references to it) to add a "print blank" capability. The Command Interpreter uses its own vector to a "COUT" routine via CSW at $36; however, the File Manager (previously) used the Monitor COUT and CROUT routines for printing the catalog. With WildCAT all of DOS now consistently uses the vector at $9FCA for output; plus it has a new BlankOUT routine, all within the original code space.

(3) Recode a very cumbersome form of the "indexed indirect jump" to use register Y and leave X (which is zero by a previous operation) so it can be used in NewDOCAT.

(4) Replace old DOCAT's 12 bytes of code with a JMP to NewDOCAT and use the remainder to space over to column 7 after the file length has been displayed.

(5) NewDOCAT saves the first character of FNAME and substitutes a zero to prevent buffer allocation. It then loads 13, the new Catalog Function Code, and proceeds to CMDHNDLR2. Function 13 enters the catalog code past the "allow for irregular, direct entry".

(6) In the keyword parameter table, change parms to allow a filename with CATALOG and a filename, drive, and slot with DIR. Set new Function 13 address (previously a useless "no-op" to NOERROR routine) to WildCAT and change the range check to 14 to allow for it.

(7) Replace the Integer CHAIN code; PrtLOCK displays an asterisk or blank if the file is locked or not.

(8) Shorten the "NO BUFFERS AVAILABLE" message to "NO BUFFER" and re-use the space to decide which Basic is active, then JMP to the appropriate decimal print routine; used to print the free sector value and catalog filesizes. The value to be printed has been previously loaded into A and X.

(9) First, eliminate the need for "NOT DIRECT COMMAND" error message and then re-use the space to check for a "regular" catalog (no filename) or for a catalog by filetype (undelimited, single character). If more than a single, non-blank character is detected (ie, 2nd byte of FNAME is not blank), then "SYNTAX ERROR" message is issued.

(10) At the beginning of catalog code, allow for most reasonable points where the code could be directly entered. The new "official" function 13, WildCAT entry initializes the FM workarea (per normal) and branches to Read VTOC to "find" the first catalog sector.

(11) Freespace "prolog"; clear carry and branch around another likely "irregular" entry point. Read first/next catalog sector, then lookup and save the filetype. Setup Y with 30 for name length and branch to CkFNAME.

(12) AllowVTOC fakes a "regular" catalog and falls into a JSR to read the VTOC. The BCC to initialize linecount is always taken; only if there had been an I/O error would the carry be set, in which case, control would have passed to the error-message-print exit anyway.

(13) PrtCat displays a catalog line. Note that loc $24, CH, is "POKEd" with 7 for uniform spacing over to the filename. If your printer interface board or 80-column card do not support this convention, then the display will not be properly spaced. The DONEXT routine is unchanged. SKIPLN has been re-arranged to allow init

linecount, put out a carriage return, and check for a keypress (Ctrl-Q to quit) after 22 lines. Note: This leaves the cursor in column 37; see below.

(14) CkFNAME "looks under the shell" to figure out what to do. A caret indicates to check for a wildcard string. After JSR to CkCAT, if the equal status is set, then branch to print the catalog line. DoWild checks for the occurence of the wildcard string within the filename. $B4C9,X indexes the name in the Catalog Sector; $AA75,Y indexes the wildcard string; CatNmLen counts from 30 to 0, to scan the whole name.

(15) FreeSpce counts the free sectors, as indicated by the VTOC, loads X and A with the count, and JMPs ToPrtDec.

(16) WaitCk79 provides the "wait" for TYPE; also checks and puts out a carriage return after 79 characters to avoid over-printing long lines on certain printers, such as the MX-80.

(17) TYPE displays the contents of a sequential or random text file. A keypress will pause the display, and Ctrl-Q aborts or quits the display.

(18) InvCOUT is used by both CATALOG and TYPE. It converts hi-bit off characters to proper inverse. It will optionally display control characters as inverse or allow them the "function" as-is; and it will optionally "shift" lower case letters to upper case, if you do not have a lower case adapter; see "...Options" above. Location $EA, decimal 234, is the Applesoft Hi-Res collision counter; it should always be zero, unless you POKE it.

(19) WaitCQ waits for a keypress and sets the equal status, if Ctrl-Q was pressed.

(20) Replace the inverted phrase DISK VOLUME with FREE SPACE=.

(21) The DOSCMDS list is moved down 6 bytes. AllowENT re-uses these 6 bytes to force a blank in FName1 "under the shell" to facilitate "irregular" entries into the catalog code; and clears the carry in case the entry was 'atADC9' which also previously cleared the carry. In the command list, TYPE replaces CHAIN and DIR replaces POSITION; change $A8BF:43 41 D4 to replace with CAT.

(22) Change the two references to DOSCMDS to the new location. These two changes must be done last as the EXEC is changing the very code that is executing.

I would like to thank Lee Reynolds and Art Schumer for their helpful comments and suggestions.

```
=============================================================================
DOCUMENT :AAL-8510:Articles:ErvEdgeWildcatx.txt
=============================================================================
```

WildCAT for DOS 3.3...................................Erv Edge
Anchorage, Alaska

WildCAT is a series of patches to DOS 3.3 which modify the CATALOG command.  The new features include:

* A catalog by "wildcard" FILENAME facility.
* A catalog by FILETYPE facility.
* An alternate, short-form: either DIR or CAT.
* Catalog free space patch.
* Ctrl-Q catalog abort.
* TYPE a random or sequential text file.

Lee Reynold's FILEDUMP command has been re-packaged and re-presented as TYPE (see Call-A.P.P.L.E.  6/82 p47).  More on this later.  WildCAT, along with TYPE, is an attempt to teach new tricks to an old dog, as it were.

The normal DOS catalog command allows slot, drive, and volume parameters.  I have added a filename parameter, but it is processed a little differently than the way file names are usually processed.  To display the catalog entries for all files whose names contain a particular string, type any of the folowing:

```
   CATALOG ^string [,Dn] [,Sn] [,Vn]
   DIR ^string [,Dn] [,Sn] [,Vn]
   CAT ^string [,Dn] [,Sn] [,Vn]
```

where "^string" begins with the "^" or caret symbol (shifted N on the ][+ or shifted 6 on the //e); the string should contain no blanks, although it may "end" with them; the string would normally end with a carriage return or with a comma if a drive or slot number is specified.  Only those files that contain the "string" somewhere in the filename will be listed.  (Of course you already know that the D, S, and V parameters are shown in brackets above because they are optional; you do not type the brackets.)

For example, "CATALOG ^TEST" would list each file with 'TEST' as part of the filename; while "DIR ^PAY." would list those with 'PAY.' somewhere in the filename; and "CAT^.OBJ,D2" would list filenames on drive 2 that contain the partial string '.OBJ'.  "CAT" and "DIR" are simply synonyms for "CATALOG".

I have also arranged things so you can list the catalog entries of a specified file-type.  You simply type the file taype code in the CATALOG command:

```
   CATALOG t [,Dn] [,Sn] [,Vn]
   DIR t [,Dn] [,Sn] [,Vn]
```

```
   CAT t [,Dn] [,Sn] [,Vn]
```

where "t" is any of the unadorned, single-letter filetypes: A B I R S T. Only that type of file (if present) will be listed.

For example, "CATALOG T" would list all the text files; "DIR A,D2" would list all of the Applesoft files on drive 2; "CAT B,S5,D1" would list all the binary files on slot 5, drive 1. Yes, "DIRT" works just fine.

I added the TYPE command, which allows you to display the contents of text files. Both CATALOG and TYPE will optionally:

1. Print "hidden" control characters as inverse:
   POKE 234,0 to print as inverse (default)
   POKE 234,255 to function as-is

2. Lower case letters may be shifted to upper case:
   POKE -18700,255 no shift (default)
   POKE -18700,223 to shift lower to upper case.

You can slow down TYPE's output via SPEED=xx or POKE 241,xx; it can be paused by pressing any key; then Ctrl-Q to abort. Also, TYPE pauses and waits for a keypress when it encounters a hex 00 imbedded in the file or at end of file; press Ctrl-Q to quit. Random text files may be TYPE'd by holding down REPT-SPACE to get past the hex 00's at the end of each logical record.

The listing that follows is intended for information only: it is not BRUNable. My intention is that you prepare the EXEC shown below to actually install the patches. Any word processor that produces a straight, sequential text file may be used to prepare the EXEC. Of course you can also use the S-C Macro Assembler for this purpose. Then, type EXEC WILDCAT to apply the patches to DOS 3.3 in memory. After checking it out and running any other tests you like, put in a new diskette, enter a HELLO program, and type INIT HELLO to "permanently" install WildCAT in the DOS on tracks 0, 1, and 2.

When I wrote WildCAT, I had two main goals in mind: it should be a (mostly in-place) code replacement, and it should be compatible with the known means of using (abusing?) the existing CATALOG code at $AD98-AE69.

One major design consideration was a mechanism for entering the ^string/type parameter. This required merely changing the "keyword parameter table" to allow CATALOG to have a "filename".

Next, a distinction had to be made between a "wildcard" and a "filetype" parameter. It made sense to 'delimit' the wildcard string; then the single-character filetype would be just that: a single character, entered without a delimiter. But this "phony" name mechanism has it's own problems:

First, "What's in a Name?" (DOS Manual p. 16): it has to start with a letter...which automatically eliminates most special characters (eg, equal, pound, slash, colon, etc) as the delimiter. The command parsing routine doesn't really know what it's working on at the time. All it knows is: if a name may be present, it must be valid. The validity test is only that the first character be equal to or greater that $C0 or an @-sign. The @-sign could have been used, but it's a problem on some 80-column boards; the ^ or caret works nicely (and besides, it looks good).

Second, now that we have a name (however, phony) and since the CATALOG command lives in the File Manager (FM) portion of DOS, there will be a buffer allocated for it. Unfortunately, the Command Interpreter (CI) DOCAT routine, which calls the FM, already "knows" that there will not really be a name, so it does not include housekeeping code to deallocate a buffer. So merrily allocating files without closing them...after the third time: NO BUFFERS AVAILABLE. And naively adding CLOSE (even if there were room for it), would have one very undesirable side effect if a "regular" catalog were requested: CATALOG-CLOSE without FNAME means close all open files. WildCAT, instead, plays a little shell game with DOS: The new DOCAT routine saves the first character of FNAME and substitutes a zero. Thereafter, neither the File Manager nor the rest of DOS ever knows that a name has been entered, and a buffer is never actually allocated.

Third, what really should happen if a phony name is not entered? A regular catalog, of course, but how would this be indicated to WildCAT? Well, the shell game has a sting. Early on when the CI PARSE routine discovers that a filename is a valid parameter, it first clears FNAME to all blanks, expecting to fill it in with whatever comes in next. If a comma or carriage return comes in next, then FNAME still contains the blank; and that's what WildCAT saves off (under the shell) before it substitutes the zero.

Thus, the "sting" is that the CI "tricks" itself into telling WildCAT what to do in the absence of a ^string/type specification: WildCAT takes a blank to indicate "do a regular" catalog; just as positively as a "^" indicates "do a wildcard" catalog, and a single, undelimited character indicates "do a filetype" catalog.

The blank indicator also helps satisfy the second goal above and solves the problem of compatibilty with the "known means" of using/abusing the existing CATALOG code. WildCAT simply has to put a blank under the shell at each of the points where the code could most reasonably be entered without going thru the Command Interpreter's new DOCAT routine. That's exactly what all the JSR's to the routine AllowENTry are doing.

Satisfying that second goal takes up considerable space, however; and has somewhat undermined the first constraint: WildCAT certainly isn't "in-place" in one place! And I apologize for this rather bizarre, serpentine code; I do hope that now you understand why some things were done the way they were.

Although considerable effort was spent to maintain compatibilty with the existing DOS commands, there were some compromises:

1. While the DOS manual (page 22) states: "To specify drive 1, you use the notation D1 separated from the file name by a comma", you can in fact leave out the comma between CATALOG and D1. With WildCAT that comma is now required; otherwise, it would take the "D" as a filetype and try to find it...which of course it wouldn't and there would be no files reported. This would also be a problem for Applesoft programs that do something like: PRINT D$"CATALOG D1" without the separating comma. Therefore, WildCAT issues a (late) "SYNTAX ERROR" message if it encounters an undelimited string of length 2 or more.

2. CATALOG is a favorite routine to execute directly, bypassing the DOS Command Interpreter. FID, for example, provides its CATALOG via the "external" entry to the File Manager, which means that the main entry at CATHNDLR must provide for a "regular" catalog. It is also possible from machine language, however, to bypass both the CI and the FM. This usually involves changing the exit JMP address at DONEXT2 (to return to the user's code) and then jumping directly into almost anywhere in the CATALOG code (see the Listing 1 labels that begin "at"). I believe most of these cases are covered, but you may find some programs, which provide an "internal" CATALOG, that just won't work with WildCAT.

3. In order to both gain some patch space and provide the DIR/CAT short-form command name, the DOS command POSITION was eliminated. You may have to read about it just to find out that it is, much less what it is. Its relative lack of use may be due to its implementation: it, like APPEND, finds its way through the file one byte at a time...all day long. Any program that uses it will now get a syntax error. If POSITION is really needed, it can be readily simulated by programming a read-loop to discard N-1 fields before processing the desired Nth field.

The following is a brief commentary on the assembly listing. The paragraph numbers correspond to comment numbers in the listing.

The page zero locations I used ($EB thru $EF) are free, i.e. not used by DOS, the Monitor, or the Basics.

(1) In CMDTBL, replace Integer CHAIN address with TYPE and DOCAT address with NewDOCAT.

(2) Rearrange some code (and change the two references to it) to add a "print blank" capability. The Command Interpreter uses its own vector to a "COUT" routine via CSW at $36; however, the File Manager (previously) used the Monitor COUT and CROUT routines for printing the catalog. With WildCAT all of DOS now consistently uses the vector at $9FCA for output; plus it has a new BlankOUT routine, all within the original code space.

(3) Recode a very cumbersome form of the "indexed indirect jump" to use register Y and leave X (which is zero by a previous operation) so it can be used in NewDOCAT.

(4) Replace old DOCAT's 12 bytes of code with a JMP to NewDOCAT and use the remainder to space over to column 7 after the file length has been displayed.

(5) NewDOCAT saves the first character of FNAME and substitutes a zero to prevent buffer allocation.  It then loads 13, the new Catalog Function Code, and proceeds to CMDHNDLR2.  Function 13 enters the catalog code past the "allow for irregular, direct entry".

(6) In the keyword parameter table, change parms to allow a filename with CATALOG and a filename, drive, and slot with DIR.  Set new Function 13 address (previously a useless "no-op" to NOERROR routine) to WildCAT and change the range check to 14 to allow for it.

(7) Replace the Integer CHAIN code; PrtLOCK displays an asterisk or blank if the file is locked or not.

(8) Shorten the "NO BUFFERS AVAILABLE" message to "NO BUFFER" and re-use the space to decide which Basic is active, then JMP to the appropriate decimal print routine; used to print the free sector value and catalog filesizes.  The value to be printed has been previously loaded into A and X.

(9) First, eliminate the need for "NOT DIRECT COMMAND" error message and then re-use the space to check for a "regular" catalog (no filename) or for a catalog by filetype (undelimited, single character).  If more than a single, non-blank character is detected (ie, 2nd byte of FNAME is not blank), then "SYNTAX ERROR" message is issued.

(10) At the beginning of catalog code, allow for most reasonable points where the code could be directly entered.  The new "official" function 13, WildCAT entry initializes the FM workarea (per normal) and branches to Read VTOC to "find" the first catalog sector.

(11) Freespace "prolog"; clear carry and branch around another likely "irregular" entry point.  Read first/next catalog sector, then lookup and save the filetype.  Setup Y with 30 for name length and branch to CkFNAME.

(12) AllowVTOC fakes a "regular" catalog and falls into a JSR to read the VTOC.  The BCC to initialize linecount is always taken; only if there had been an I/O error would the carry be set, in which case, control would have passed to the error-message-print exit anyway.

(13) PrtCat displays a catalog line.  Note that loc $24, CH, is "POKEd" with 7 for uniform spacing over to the filename.  If your printer interface board or 80-column card do not support this convention, then the display will not be properly spaced.  The DONEXT routine is unchanged.  SKIPLN has been re-arranged to allow init

linecount, put out a carriage return, and check for a keypress (Ctrl-Q to quit) after 22 lines. Note: This leaves the cursor in column 37; see below.

(14) CkFNAME "looks under the shell" to figure out what to do. A caret indicates to check for a wildcard string. After JSR to CkCAT, if the equal status is set, then branch to print the catalog line. DoWild checks for the occurence of the wildcard string within the filename. $B4C9,X indexes the name in the Catalog Sector; $AA75,Y indexes the wildcard string; CatNmLen counts from 30 to 0, to scan the whole name.

(15) FreeSpce counts the free sectors, as indicated by the VTOC, loads X and A with the count, and JMPs ToPrtDec.

(16) WaitCk79 provides the "wait" for TYPE; also checks and puts out a carriage return after 79 characters to avoid over-printing long lines on certain printers, such as the MX-80.

(17) TYPE displays the contents of a sequential or random text file. A keypress will pause the display, and Ctrl-Q aborts or quits the display.

(18) InvCOUT is used by both CATALOG and TYPE. It converts hi-bit off characters to proper inverse. It will optionally display control characters as inverse or allow them the "function" as-is; and it will optionally "shift" lower case letters to upper case, if you do not have a lower case adapter; see "...Options" above. Location $EA, decimal 234, is the Applesoft Hi-Res collision counter; it should always be zero, unless you POKE it.

(19) WaitCQ waits for a keypress and sets the equal status, if Ctrl-Q was pressed.

(20) Replace the inverted phrase DISK VOLUME with FREE SPACE=.

(21) The DOSCMDS list is moved down 6 bytes. AllowENT re-uses these 6 bytes to force a blank in FName1 "under the shell" to facilitate "irregular" entries into the catalog code; and clears the carry in case the entry was 'atADC9' which also previously cleared the carry. In the command list, TYPE replaces CHAIN and DIR replaces POSITION; change $A8BF:43 41 D4 to replace with CAT.

(22) Change the two references to DOSCMDS to the new location. These two changes must be done last as the EXEC is changing the very code that is executing.

I would like to thank Lee Reynolds and Art Schumer for their helpful comments and suggestions.

```
==============================================================================
DOCUMENT :AAL-8510:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 6 -- Issue 1                    October, 1985

In This Issue...



Book, Books, Books

Inside this issue you will find a review of Jules Gilder's new book on intermediate-level Apple assembly language programming, and the details on those long-awaited Addison-Wesley editions of Apple's Technical Manuals.  We're now offering these items for sale, and the details are in our ad.

The latest word from Prentice-Hall is that David Eyes' "Programming the 65816" will be shipped on October 29, so we may actually have copies by the time you read this.  Bob will have a full review next month, and we are beginning to get orders already.  The list price is expected to be $22.95.  If that holds, our price will be $21.00 + postage.

A Rumor Regarding the Next Apple II

We have heard from two sources now a rumor that Apple does not plan to use the 65816 in its next Apple II.  Nor the 65802, nor the 65C02. Instead, we heard, they will use a custom version of the 68000 family with 65C02 emulation capability.  I think that I hope that the rumor is groundless, but I'll keep my ear to the ground anyway.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

=============================================================================
DOCUMENT :AAL-8510:Articles:Gilder.Review.txt
=============================================================================

Now That You Know Apple Assembly Language,
What Can You Do With It?..................Review by Bill Morgan

Do you know the difference between LDA LABEL,X and LDA (LABEL),Y but wonder when to use which? Are you confused by the way PHA, PHA, RTS doesn't go home, but jumps somewhere else entirely? Do you know what the 6502 opcodes do, but still feel lost when it comes time to combine them into a program?

Jules Gilder, a long-time contributor to several of the Apple Magazines, has written a book just for you. He spends about 190 pages covering the intermediate level of assembly language programming in the Apple II computer. His programs are very well commented, and the accompanying text contains almost a line-by-line discussion of how and why each program works.

Gilder concentrates on the Apple-specific features of 6502 programming: input and output hooks, the internal speaker, and basic linkage to Applesoft. This combination should make this book especially appealing to those of you who have learned 6502 from a "generic" book and want to find out how to apply your new knowledge to your Apple II's.

Here is a summary of each chapter of Now That You Know...:

1) Before You Get Started -- This is an introduction to assemblers and their conventions.

2) Getting Information out of Your Computer -- This chapter covers simple output, including message printing and decimal number display.

3) Getting Information into Your Computer -- Here we get into reading keystrokes and lines, handling decimal input, and also menu control structures.

4) Stealing Control of the Output -- This one goes into taking over the output hook to do custom printer setup codes and drivers, output filtering, and formatting.

5) Stealing Control of the Input -- Learn how to grab the input hook to add a custom cursor, numeric keypad, an in-memory EXEC simulator, an Applesoft keyboard macro facility, and a lower-case input driver using the shift-key modification.

6) Using Sound in Your Programs -- How to use the Apple's built-in speaker to create a variety of sounds.

7) Learning to Use the Ampersand -- Here are techniques for hooking into the &-vector to do hexadecimal input and output in Applesoft,

find a program line in memory, append two Applesoft programs, and revive a program lost by the NEW command.

8) Expanding Applesoft BASIC -- Now we can have computed GOTO, GOSUB and LIST, do double-byte PEEKs and POKEs, switch between two Applesoft programs sharing memory and variables, and add function keys to control output modes.
The only real weakness in this book is the complete lack of attention to the Apple's graphic display possibilities, and comparatively little coverage of dealing with DOS (and only one small appendix covering conversion to ProDOS.) I suppose Gilder regards these as more advanced topics. Hopefully he will see fit to focus on such subjects in a future book.

Gilder's company, Redlig Sytems, Inc., also has diskettes of all the programs in the book, in either source or object form.

We'll be carrying Now That You Know... for only $18 + shipping.

Apple Software Protection Digest

Gilder is also starting a newsletter on the subject of Apple software protection. This publication is devoted both to protecting your own programs and defeating the protection on others'. Here is part of Jules' description:

Apple computer owners need a place where they can get more information about software protection. They need a forum where they can exchange ideas with others who face the same or similar problems. They need to know what software protection is, how it's implemented, what are the consequences of it, how it can be overcome if necessary and if there are any comparable unprotected alternatives to particular protected software packages.

Apple Software Protection Digest will provide you with this information and more. It will show you new ways to protect, unprotect and backup your programs. It will teach you how to prevent others from accessing your programs and it will show you how to make them more difficult to copy. In addition, you'll learn how to overcome these and other protection schemes that are in use. You'll learn how to use the powerful, but compli- cated nibble copy programs. You'll also learn how to crack or remove protection entirely from many programs.

In the first issue he covers hiding Applesoft program lines (and finding them once they're hidden), making a machine language program automatically execute when BLOADed, protecting a disk by adding extra tracks and leaving some tracks unformatted, backing up The Print Shop, and he reviews the Copy II Plus nibble copier.

As a special offer for AAL subscribers, Gilder will give you a free copy of the first issue of Apple Software Protection Digest. Just send your name and address to Redlig Systems, Inc., 2068) 79th St., Brooklyn, NY, 11214.  Be sure to mention that you are an AAL reader. The subscription rate is $24 for one year, or $42 for two years.

==============================================================================
DOCUMENT :AAL-8510:Articles:Index.2.Vol.5.txt
==============================================================================

INDEX

Apple Assembly Line, Volume 5
October, 1984 through September, 1985

AAAA



BBBB

CCCC



DDDD

SSSS



TTTT

=============================================================================
DOCUMENT :AAL-8510:Articles:JohnLoveArticle.txt
=============================================================================

(Semi-) Protect a Disk......................John A. Love, III
Washington Apple Pi

In the September 1985 issue of AAL Bob S-C developed a program to create a combination DOS 3.3 and ProDOS Data disk. My first program listing is in response to his invitation to readers for the development of a front-end wherein the user can select the following:

1. Slot #.
2. Drive #.
3. DOS 3.3 low Track #.
4. ProDOS Volume Name.

Notice that I did NOT include the DOS 3.3 Volume number which default value provided by DOS 3.3 is 254. Since I do not own a Hard Disk, I saw no need for changing it.

Several items pertaining to my front-end should be noticed. First, the error trapping; for example, the Slot number must be between 1 & 7 and the ProDOS Volume Name not only must begin with a letter but also must NOT exceed 15 characters in length. Second, since each input is either one Byte or a string of single Bytes, there were a few instances in which I had to change Bob's code. For example, such expressions as "LDA #SLOT*16" and "CPY #4*DOS.LOW.TRACK" were out of the question. As you compare his listing with mine, you will notice the changes that I had to incorporate in order to accomodate user input.

Well, enough of un-finished business. The other evening I was asked how to protect a Disk from un-warranted intrusion. Initially, I knew nothing about Disk protection except to state the obvious; namely, that given enough time and talent ANY protection scheme can be broken. As the very old adage stipulates -- "Locks are meant only to keep honest people out". With this "awesome" knowledge in mind and Bob's program at my elbow, I decided to apply his program's logic to simply slow the folks down a bit.

My second program does NOT inhibit COPYA in the slightest. But, that's okay because the caller, a teacher at one of the local colleges, didn't mind. You see, his Disk contained quizzes along with the answers. Get the drift.... So, I decided to:

1. Defeat the CATALOG Command on the DOS stored on the Disk. If the Disk was cold-booted, the user would not know the names of the files to LOAD & LIST.

2. Place the CATALOG info on a Track other than #17 (I chose #18 since DOS searches uphill from 17 before going downhill -- in short, keep the time element at a minimum). In this manner, if the

inquisative user attempts to CATALOG the quiz Disk with his own System Master, he will get a blank screen.  Part of my code to follow will ensure not only a blank screen, but also a full disk indication because I place 00's throughout Track 17, Sector 0.

Admittedly, such a scheme is very UN-sophisticated.  However, all I wished to do was to slow the inquisative folks down a bit.

With respect to my adaptation of Bob's program, notice that the only parameter I chose for user input was the Drive number.  Others, such as the Slot #, the Volume # and the "real" CATALOG Track # I inserted directly into the code.  Using my logic pertaining to inputting the Drive #, these other parameters can easily be input as well.  Also, notice that I have included remarks at the end of the Source Code on how to use "DISK.SAFE" .

================================================================================
DOCUMENT :AAL-8510:Articles:Mcinerney.Sieve.txt
================================================================================

Feedback about the latest Sieve............Peter J. McInerney

So the Sieve lives!  Bob's article last month misses some of the facts, however.  He states that my improved 68000 version on my 12.5 MHz DTACK Grounded board ran in .4 seconds; the actual time was .33 seconds.  This is proportional to the .49 seconds claimed in the later Byte article for an 8 MHz 68000.  My DTACK Grounded board uses 120 nanosecond static RAM and runs at a full 12.5 MHz speed (DTACK grounded means that the processor CANNOT wait for memory).

Hal Hardenburgh (editor of the now sadly no more DTACK newsletter and no slouch when it comes to assembly programming on the 68000) produced his own version of the original algorithm, essentially hand-compiled BASIC since that was what he wanted to compare to, and that ran in 1.29 secs for 10 iterations on a 10MHz board.

My faster 68000 sieve was my first 68000 program, so in light of my now more extended experience I tried to tighten it up even further. The result runs in .28 seconds for ten iterations on my DTACK board, and .72 seconds on a Macintosh.  The main speed improvement comes from loading two extra registers for comparisons rather than doing CMPI's. The use of MOVEM for clearing the array was pointed out to me by Hal Hardenburgh and accounts for about .02 secs saved, at the expense of a large amount of elegance (oh well, what price aesthetics?).

In trying to guess the comparisons of the 65816 systems of the future with existing 68000 systems, two questions come to mind.  First, if 6 or 8 MHz 65816s become available in quantity, how fast will the memory have to be to keep up?  The 68000 can automatically adjust for slower memories, but is this true of the 65816?  Second, and more importantly, is the question of memory addressing.

I wrote a version of the sieve that sifts the first 262143 integers. This took 13.5 seconds for 10 iterations on a Macintosh (this should equate to 5.3 seconds on my DTACK board, but I don't have enough memory to test it.)  The program is only minimally different from the original (some constants changed and some address modes changed from word to long.)

How about writing a 65816 program to handle this large of an array? How much extra baggage is required to test page boundaries, move base addresses, etc?  My point is that the restriction of 64K banks can really hurt in accessing large data arrays.  Memory is getting cheaper all the time, so using more bytes for a 68000 program may well be no penalty, compared with the extra difficulty of writing 65816 code to handle large amounts of data.

```
==============================================================================
DOCUMENT :AAL-8510:Articles:My.Ad.txt
==============================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners........................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.......................................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $32 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *
Switch-a-Slot (SCRG).........................................($190) $175 *
Extend-a-Slot (SCRG)..........................................($35)  $32 *

"Programming the 65816", Eyes...............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
```

```
"Beneath Apple ProDOS", Worth & Lechner....................($19.95)  $18 *
"6502 Subroutines", Leventhal..............................($18.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner.......................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger.......................($39.95)  $36 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
              Foreign customers inquire for postage needed.
        Texas residents please add 6 1/8 % sales tax to all orders.
```

=============================================================================
DOCUMENT :AAL-8510:Articles:PolyCol.Disasm.txt
=============================================================================

Multiple Column Dis-Assembly.......................Adam Levin

When I'm writing and debugging a program, I always use a lot of printer paper as I list and re-list version after version of my creation. Using the Apple monitor's 'L' command wastes a lot of that paper, too. Since each disassembled line takes at most 36 characters, I end up wasting half of each page!. I know I could feed the paper through a second time with the right hand side now on the left, but the left hand listing isn't always the same length as the right, so I end up with listings that span several separate lengths of paper. I've written a program to solve this dilemma (as if you hadn't guessed!), and I call it PolyCol.

PolyCol will be of use no matter what type of printer you have: daisywheel printer and 80-column video card owners will get two columns per page (screen), 80-column dot matrix owners can get up to four columns per page by using compressed printing, and those with wider carriages can get even more! In addition, by compressing the print size vertically as well, it is possible to get a disassembly of all the ROMs in the Apple onto only 16 pages! (It's also possible to go blind trying to read it!)

Note that rather than creating all the text in memory, and then dumping an entire page at once, PolyCol calculates which opcode to disassemble where, 'on-the-fly'. You might think that this would slow things down appreciably; but in fact unless you require tens of columns, the listing is done relatively quickly.

As you will see from the listing, seven zero-page locations are used to hold the parameters which the user must specify. You must store the starting and ending addresses of the area to be dis-assembled into locations $00-03. Locations $04-07 control the number of lines per page and columns per line, as well as several other features. Here are some examples to show what you can do with different parameter settings:

```
     $04  $05  $06
     ---  ---  ---

     $01  $14  $FE  - Standard monitor 'L' listing.
                      Press any key to see the next page.

     $02  $36  $FF  - Two column, 54 line page with a form feed
                      in between pages

     $04  $4C  $0C  - Four column, 76 line page with 12 spaces
                      between pages.  Don't forget to set
                      elite typeface and compressed print.
```

```
    $04  $70  $FF  - Four column, 112 lines per page!
                     To do this I had to use compressed
                     elite super- script, with a line
                     spacing of 1/12th in.
```

You can add just a little code to POLYCOL to set it up as a control-Y command.  Then you could set the starting and ending addresses as in normal monitor commands.  The other four parameters could also be specified in the control-Y command format, if you really get serious about modifications.

=============================================================================
DOCUMENT :AAL-8510:Articles:Puzzle.txt
=============================================================================

Paint Yourself into the Corner....................Adam Levin

I think I have come up with an interesting puzzle. Pretend that your Apple has only 48K of RAM: no ROM, no soft switches, no memory cards, just 48152 bytes of contiguous RAM from $0000 through $BFFF. Now, write a program which will store one number (of your choosing) into each and every one of these 49152 locations. The stumper here is creating a program which can overwrite itself completely, and which will not go running off through the I/O area causing disks to spin, etc.

There are certain limitations to actually implementing this on an Apple. When you hit <RESET> to examine the contents of memory after running your program, memory will be changed before you can look at it. It is unavoidable that page zero, the stack, and text screen memory will all get disrupted as soon as <RESET> is pressed. You still need to include these areas in your program, but you just will not be able to check them.

You will have to figure out some way of stopping the program before it runs off into the $Cxxx space. I decided to accept this limitation by allowing three bytes at $BFFD-F to contain a JMP instruction, not stuffing my favorite number in them. So my solution actually only stuffs my number into $0000-$BFFC.

Bob Sander-Cederlof has a solution that stuffs the same number in every byte from $0000 through $BFFF, but depends on two locations in the I/O area to stop the program from rampaging around $Cxxx space.

Try your hand at this puzzle! Next month we'll show some of the best solutions.

```
==============================================================================
DOCUMENT :AAL-8510:Articles:QD20.CoverSheet.txt
==============================================================================
```

QUARTERLY DISK #20 contains all the source code from Volume 5, Issues 10-12 of the Apple Assembly Line newsletter. The files are formatted for the S-C Macro Assembler, on a combination DOS 3.3 and ProDOS disk.

DOS Files

S.BINDEC -- A program to do binary/decimal/binary conversions. This one is designed to be especially easy to modify for the precision you need.

S.BYTE TABLE -- This routine returns the number of bytes used by each opcode (including 65816), along with flags to indicate if it's immediate or absolute. Here's a piece of a possible future relocator.

S.WILDCARD -- A filename search routine, with wildcards, useful in any kind of DOS utility.

S.65802.CONVERSIONS -- Extremely short binary-to-decimal conversion using the 65802.

S.INIT DOS & PRODOS -- Program to initialize a data disk with partitions for both DOS 3.3 and ProDOS. This is the program used to produce this disk.

S.SUPER-FAST PRIMES 65802, S.SUPER-FAST PRIMES 65802+, PRINT PRIME TABLE -- The Sieve of Eratosthenes prime-number generator, coded for the 65802.

ProDOS Files

S.RECURCAT -- A program to list all files in all subdirectories of a ProDOS disk. Also an interesting example of recursive techniques.

S.DOS.LOAD -- This program LOADs DOS 3.3 source files into the ProDOS version of the S-C Macro Assembler.

BUF.576K, BUF.320K, BUF.64K -- Use your extra Apple //c with Z-RAM for a serial printer buffer!

```
=============================================================================
DOCUMENT :AAL-8510:Articles:Snooper.txt
=============================================================================
```

ProDOS Snooper.............................Bob Sander-Cederlof

This past week I have been working on a project which involved creating a new device driver for a disk-like device. In the process of debugging my driver, I had to write a "snooper" program.

By "snooper", I meean a program which will make a list of all calls to the driver, recording the origin of the call and the parameters of the call.

ProDOS keeps a table of the addresses of the device drivers assigned to each slot and drive between $BF10 and $BF2F. There are two bytes for each slot and drive. $BF10-1F is for drive 1, and $BF20-2F is for drive 2. For example, the address of the device driver for slot 6 drive 1 is at $BF1C,1D. (Normally this address is $D000.)

I have a Sider drive in slot 7. The device driver address for the Sider is $C753, and is kept at $BF1E,1F and $BF2E,2F.

By patching the device driver address to point to my own code, I can get control whenever ProDOS tries to read or write or whatever. If I save and restore all the registers, and jump to the REAL device driver after I am finished, ProDOS will never be the wiser. But I will!

While my program has control, I can capture all the information I am interested in. Unfortunately I cannot print it out at this time, because if I try to ProDOS will get stuck in a loop. Instead I will save the data in a buffer so I can look at it later.

The program which follows has three distinct parts. Lines 1140-1290 are an installation and removal tool. If the program has just been BLOADed or LOADed and ASMed, running INSTALL.SNOOPER will (you guessed it!) install the snooper. The actual device driver address for the slot (which you specified in line 1060 before assembling the program) will be saved in my two-byte variable DRIVER. The previous contents of DRIVER, which is the address of my snoop routine, will be copied into ProDOS's table. The value of DRIVES, which you specified before assembling the program at line 1070, will determine whether SNOOPER is connected to drive 2 or not. It will always be connected to drive 1.

If SNOOPER has already been installed, running INSTALL.SNOOPER will reverse the installation process, returning ProDOS to its original state. INSTALL.SNOOPER also resets the buffer I use to keep the captured information. To make it easy to run INSTALL.SNOOPER, I put a JMP to it at $300. After assembly you can type "$300G" to install the snooper, and type the same again to dis-install it.

The JMP at $303 (line 1120) goes to the display program. After SNOOPER has been installed, all disk accesses on the installed slot will cause information to be accumulated in BUFFER. Typing "$303G" will cause the contents of BUFFER to be displayed in an easy-to-read format.

I set up SNOOPER to capture eight bytes of information each time it is activated. You might decide to save more or less. I save the return address from the stack, to get some idea of which routine inside ProDOS is trying to access the disk. I also save the six bytes at $42-47, which are the calling parameters for the device driver. Page 6-8 of Beneath Apple ProDOS describes these parameters; you can also find out about them in Apple's ProDOS Technical Reference Manual and in Gary Little's "Apple ProDOS--Advanced Features".

$42 contains the command code: 00=status, 01=read, 02=write, and 03=format. $43 contains the unit number, in the format DSSS0000 (where SSS=slot and D=0 for drive 1, D=1 for drive 2). $44-45 contain the address of the memory buffer, lo-byte first; the buffer is 512 bytes long. $46-47 contain the block number to be read or written.

My DISPLAY program displays each group of eight bytes on a separate line, in the following format:

```
hhll:cc.uu.buff.blok
```

where hhll is the return address from the stack, hi-byte first; cc is the command code; uu is the unit number; buff is the buffer address, hi-byte first; blok is the block number, hi-byte first.

If you get into figuring out more of what ProDOS is doing, you might want to save more information from the stack. You can look behind the immediate return address to get more return addresses and other data which have been saved on the stack before calling the device driver.

a word of explanation about lines 1040, 1360, 1370, 1490, and 1500. Line 1040 tells the S-C Macro Assembler that it is OK to assemble opcodes legal in the 65C02. The PHX, PHY, PLX and PLY opcodes are in the 65C02, 65802, and 65816; however, they are not in the 6502. If you have only the 6502 in your Apple, you will need to substitute the longer code shown in the comments. Leave out line 1040, and use the following:

```
1360    TYA
1365    PHA
1370    TXA
1375    PHA
.
.
.
1490    PLA
1495    TAX
1500    PLA
1505    TAY
```

In the process of "snooping" I was able to debug my new device drivers for the project I was developing. I also discovered what appear to be some gross in-efficiencies in ProDOS. In the course of even simple CATALOGs, LOADs, and SAVEs the same blocks are read into the same buffers over and over, at times when it would appear to be totally unnecessary. If there was some mechanism inside MLI to keep track of the fact that a complete un-spoiled copy of a particular block was already in RAM, it could save a lot of time. On the other hand, it could be that the current approach is safer. I think it is a potentially fruitful area for further investigation. Any takers?

DOS 3.3 RWTS Snooper.......................Bob Sander-Cederlof

Of course if I want to look around in ProDOS the same curiosity certainly applies to DOS 3.3. The fact of the matter is, I started snooping in DOS first; nevertheless, the ProDOS article took precedence in these pages.

There are several nice places to patch a snooper into DOS 3.3. One is right at the beginning of RWTS, $BD00. This position is usually taken by hard disks, however. For example, Sider and Corvus use $BD00. I could skip down below $BD00, but Sider for one expects several bytes after $BD00 to be normal DOS code. Looking backward, $BD00 is normally called only from a subroutine which starts at $B7B5. This subroutine, in turn, is normally only called from $B090. Your own programs may call RWTS differently, but DOS itslef almost always goes through $B090. (The exceptions are the reading and writing of the DOS image during boot or INITialization.)

Therefore...I patched my SNOOPER program in at $B090. The INSTALL.SNOOPER code in lines 1060-1160 is very similar to that in the ProDOS snooper. It swaps the address currently in my variable DRIVER with the address at $B091,2. Typing "$800G" will install SNOOPER, and typing it again will dis-install SNOOPER.

The DOS snooper prints out each line of information as it goes along, without storing the data. Each line contains the two most recent return address from the stack, so you can trace who is calling RWTS. I also print out the RWTS command, the track and sector, and the buffer address.

Here is an example of the printout, in this case during a SAVE operation:

```
:LOAD S.RWTS.SNOOPER
:ASM                      Assembler SNOOPER

0000 ERRORS IN ASSEMBLY
:$800G                    install SNOOPER
:SAVE S.RWTS.SNOOPER      sample DOS command
AB24.AD45.01.11.00.B3BB  read VTOC
AB45.B1E6.01.11.0F.B4BB  read Catalog sector
A6AA.AB24.01.1F.0F.9700  T/S list
C3E9.ACDD.01.1F.0E.9600  read 1st data sector
```

```
ACDD.B0C8.02.1F.0E.9600  write 1st data sector
D349.ACDD.01.1F.0D.9600  read 2nd data sector
ACDD.B0C8.02.1F.0D.9600  write 2nd data sector
D328.ACDD.01.1F.0C.9600  read 3rd data sector
ACDD.B0C8.02.1F.0C.9600  write 3rd data sector
D352.ACDD.01.1F.0B.9600  read 4th data sector
ACDD.B0C8.02.1F.0B.9600  write 4th data sector
A2F8.A6AA.01.11.00.B3BB  read VTOC
A6AA.AC1E.01.11.0F.B4BB  read catalog sector
A2F8.A6AA.02.11.0F.B4BB  write catalog sector
AD1A.AB45.01.11.00.B3BB  read VTOC
AB45.B1E6.01.11.0F.B4BB  read catalog sector
A6AA.AD1A.01.1F.0F.9700  read T/S list
A6AA.AD1D.01.1F.0E.9600  read 4 data sectors
A6AA.AD1D.01.1F.0D.9600    to VERIFY the file
A6AA.AD1D.01.1F.0C.9600
A6AA.AD1D.01.1F.0B.9600
:$800G                   dis-install SNOOPER
```

=============================================================================
DOCUMENT :AAL-8510:Articles:Snoopers.txt
=============================================================================

ProDOS Snooper.............................Bob Sander-Cederlof

This past week I have been working on a project which involved creating a new device driver for a disk-like device. In the process of debugging my driver, I had to write a "snooper" program.

By "snooper", I mean a program which will make a list of all calls to the driver, recording the origin of the call and the parameters of the call.

ProDOS keeps a table of the addresses of the device drivers assigned to each slot and drive between $BF10 and $BF2F. There are two bytes for each slot and drive. $BF10-1F is for drive 1, and $BF20-2F is for drive 2. For example, the address of the device driver for slot 6 drive 1 is at $BF1C,1D. (Normally this address is $D000.)

I have a Sider drive in slot 7. The device driver address for the Sider is $C753, and is kept at $BF1E,1F and $BF2E,2F.

By patching the device driver address to point to my own code, I can get control whenever ProDOS tries to read or write or whatever. If I save and restore all the registers, and jump to the REAL device driver after I am finished, ProDOS will never be the wiser. But I will!

While my program has control, I can capture all the information I am interested in. Unfortunately I cannot print it out at this time, because if I try to ProDOS will get stuck in a loop. Instead I will save the data in a buffer so I can look at it later.

The program which follows has three distinct parts. Lines 1140-1290 are an installation and removal tool. If the program has just been BLOADed or LOADed and ASMed, running INSTALL.SNOOPER will (you guessed it!) install the snooper. The actual device driver address for the slot (which you specified in line 1060 before assembling the program) will be saved in my two-byte variable DRIVER. The previous contents of DRIVER, which is the address of my snoop routine, will be copied into ProDOS's table. The value of DRIVES, which you specified before assembling the program at line 1070, will determine whether SNOOPER is connected to drive 2 or not. It will always be connected to drive 1.

If SNOOPER has already been installed, running INSTALL.SNOOPER will reverse the installation process, returning ProDOS to its original state. INSTALL.SNOOPER also resets the buffer I use to keep the captured information. To make it easy to run INSTALL.SNOOPER, I put a JMP to it at $300. After assembly you can type "$300G" to install the snooper, and type the same again to dis-install it.

The JMP at $303 (line 1120) goes to the display program. After SNOOPER has been installed, all disk accesses on the installed slot will cause information to be accumulated in BUFFER. Typing "$303G" will cause the contents of BUFFER to be displayed in an easy-to-read format.

I set up SNOOPER to capture eight bytes of information each time it is activated. You might decide to save more or less. I save the return address from the stack, to get some idea of which routine inside ProDOS is trying to access the disk. I also save the six bytes at $42-47, which are the calling parameters for the device driver. Page 6-8 of Beneath Apple ProDOS describes these parameters; you can also find out about them in Apple's ProDOS Technical Reference Manual and in Gary Little's "Apple ProDOS--Advanced Features".

$42 contains the command code: 00=status, 01=read, 02=write, and 03=format. $43 contains the unit number, in the format DSSS0000 (where SSS=slot and D=0 for drive 1, D=1 for drive 2). $44-45 contain the address of the memory buffer, lo-byte first; the buffer is 512 bytes long. $46-47 contain the block number to be read or written.

My DISPLAY program displays each group of eight bytes on a separate line, in the following format:

```
     hhll:cc.uu.buff.blok
```

where hhll is the return address from the stack, hi-byte first; cc is the command code; uu is the unit number; buff is the buffer address, hi-byte first; blok is the block number, hi-byte first.

If you get into figuring out more of what ProDOS is doing, you might want to save more information from the stack. You can look behind the immediate return address to get more return addresses and other data which have been saved on the stack before calling the device driver.

A word of explanation about lines 1040, 1360, 1370, 1490, and 1500. Line 1040 tells the S-C Macro Assembler that it is OK to assemble opcodes legal in the 65C02. The PHX, PHY, PLX and PLY opcodes are in the 65C02, 65802, and 65816; however, they are not in the 6502. If you have only the 6502 in your Apple, you will need to substitute the longer code shown in the comments. Leave out line 1040, and use the following:

```
       1360    TYA
       1365    PHA
       1370    TXA
       1375    PHA
       .
       .
       .
       1490    PLA
       1495    TAX
       1500    PLA
       1505    TAY
```

In the process of "snooping" I was able to debug my new device drivers for the project I was developing. I also discovered what appear to be some gross in-efficiencies in ProDOS. In the course of even simple CATALOGs, LOADs, and SAVEs the same blocks are read into the same buffers over and over, at times when it would appear to be totally unnecessary. If there was some mechanism inside MLI to keep track of the fact that a complete un-spoiled copy of a particular block was already in RAM, it could save a lot of time. On the other hand, it could be that the current approach is safer. I think it is a potentially fruitful area for further investigation. Any takers?

DOS 3.3 RWTS Snooper.......................Bob Sander-Cederlof

Of course if I want to look around in ProDOS the same curiosity certainly applies to DOS 3.3. The fact of the matter is, I started snooping in DOS first; nevertheless, the ProDOS article took precedence in these pages.

There are several nice places to patch a snooper into DOS 3.3. One is right at the beginning of RWTS, $BD00. This position is usually taken by hard disks, however. For example, Sider and Corvus use $BD00. I could skip down below $BD00, but Sider for one expects several bytes after $BD00 to be normal DOS code. Looking backward, $BD00 is normally called only from a subroutine which starts at $B7B5. This subroutine, in turn, is normally only called from $B090. Your own programs may call RWTS differently, but DOS itself almost always goes through $B090. (The exceptions are the reading and writing of the DOS image during boot or INITialization.)

Therefore...I patched my SNOOPER program in at $B090. The INSTALL.SNOOPER code in lines 1060-1160 is very similar to that in the ProDOS snooper. It swaps the address currently in my variable DRIVER with the address at $B091,2. Typing "$800G" will install SNOOPER, and typing it again will dis-install SNOOPER.

The DOS snooper prints out each line of information as it goes along, without storing the data. Each line contains the two most recent return address from the stack, so you can trace who is calling RWTS. I also print out the RWTS command, the track and sector, and the buffer address.

Here is an example of the printout, in this case during a SAVE operation:

```
:LOAD S.RWTS.SNOOPER
:ASM                       Assembler SNOOPER

0000 ERRORS IN ASSEMBLY
:$800G                     install SNOOPER
:SAVE S.RWTS.SNOOPER       sample DOS command
AB24.AD45.01.11.00.B3BB  read VTOC
AB45.B1E6.01.11.0F.B4BB  read Catalog sector
A6AA.AB24.01.1F.0F.9700  T/S list
C3E9.ACDD.01.1F.0E.9600  read 1st data sector
```

```
ACDD.B0C8.02.1F.0E.9600  write 1st data sector
D349.ACDD.01.1F.0D.9600  read 2nd data sector
ACDD.B0C8.02.1F.0D.9600  write 2nd data sector
D328.ACDD.01.1F.0C.9600  read 3rd data sector
ACDD.B0C8.02.1F.0C.9600  write 3rd data sector
D352.ACDD.01.1F.0B.9600  read 4th data sector
ACDD.B0C8.02.1F.0B.9600  write 4th data sector
A2F8.A6AA.01.11.00.B3BB  read VTOC
A6AA.AC1E.01.11.0F.B4BB  read catalog sector
A2F8.A6AA.02.11.0F.B4BB  write catalog sector
AD1A.AB45.01.11.00.B3BB  read VTOC
AB45.B1E6.01.11.0F.B4BB  read catalog sector
A6AA.AD1A.01.1F.0F.9700  read T/S list
A6AA.AD1D.01.1F.0E.9600  read 4 data sectors
A6AA.AD1D.01.1F.0D.9600    to VERIFY the file
A6AA.AD1D.01.1F.0C.9600
A6AA.AD1D.01.1F.0B.9600
:$800G                   dis-install SNOOPER
```

```
=======================================================================
DOCUMENT :AAL-8510:DOS3.3:S.POLYCOL.txt
=======================================================================
```

```
 1000  *SAVE S.POLYCOL
 1010  *--------------------------------
 1020  *  PolyCol
 1030  *  Produces multi-column Apple monitor dis-assemblies.
 1040  *  Copyright (c) 1986 Adam Levin
 1050  *--------------------------------
 1060         .OR $800
 1070  *---User parameters-------------
 1080  STRTL  .EQ $00      Starting address
 1090  STRTH  .EQ $01
 1100  ENDL   .EQ $02      Ending address
 1110  ENDH   .EQ $03
 1120  NCPP   .EQ $04      # Columns per page
 1130  *                     (0 <= NCPP <= FF)
 1140  *                     (each column takes 34 chars.)
 1150  NLPP   .EQ $05      # Lines printed per page
 1160  *                     (0 <= NLPP <= FF)
 1170  NSKP   .EQ $06      # Blank lines between pages
 1180  *                     (0 <= NSKP <= FF)
 1190  *                     (FF = Form feed)
 1200  *                     (FE = pause between pages)
 1210  SLOT   .EQ $07      Slot # to direct output to
 1220  *                     (0 <= SLOT <= 7)
 1230  *                     (0 = use currently active device)
 1240  *---Program variables-----------
 1250  BRUNFX .EQ $08      Holds the DOS stack pointer
 1260  TOFARL .EQ $09      Adrs of 1st opcode in col 2;
 1270  TOFARH .EQ $0A      1st column ends just before it.
 1280  TCSWL  .EQ $0B      Holds the 'other' CSWL address
 1290  TCSWH  .EQ $0C
 1300  COLCNT .EQ $0D      Current column
 1310  TEMPL  .EQ $0E      Temporary storage
 1320  TEMPH  .EQ $0F          "        "
 1330  *---Monitor variables------------
 1340  FORMAT .EQ $2E      Holds addressing mode code
 1350  CSWL   .EQ $36      Character Output SWitch Low address
 1360  CSWH   .EQ $37          "       "      "    High   "
 1370  PCL    .EQ $3A      Adrs of opcode currently being
 1380  PCH    .EQ $3B      dis-assembled.
 1390  STKPTR .EQ $AA59    DOS 3.3 stack pointer save loc't'n
 1400  KBD    .EQ $C000    Keyboard
 1410  STROBE .EQ $C010    Clear keyboard strobe
 1420  *---Monitor ROM Subroutines------
 1430  INSDS2 .EQ $F88C    Formats each disassembly line
 1440  INSTDSPA .EQ $F8D3  Print opcode & operand
 1450  PRBL2  .EQ $F94A    Prints (X-reg) many blank spaces
 1460  PCADJ  .EQ $F953    Adjusts A,Y (PCL,H) after each line
 1470  RDKEY  .EQ $FD0C    Get an input character
 1480  CROUT  .EQ $FD8E    Print a <RETURN>
```

```
1490  PRYX2A .EQ $FD99    Print 'adrs-'
1500  COUT   .EQ $FDED    Print Acc as a character
1510  *---Macro definitions------------
1520         .MA CMPD     Double byte CMP
1530         LDA ]1       From the S-C
1540         CMP ]2       MACRO LIBRARY file.
1550         LDA ]1+1
1560         SBC ]2+1
1570         .EM
1580  *
1590         .MA MOVD     Double byte MOV
1600         LDA ]1
1610         STA ]2
1620         LDA ]1+1
1630         LDA ]2+1
1640         .EM
1650  *
1660         .MA MSG      MESSAGE PRINT MACRO
1670         LDX #]1
1680         JSR PRINT.MESSAGE
1690         .EM
1700  *---------------------------------
1710  POLYCOL
1720         LDA STKPTR   Save stack pointer now,
1730         STA BRUNFX   restore it at the end.
1740         LDA SLOT     Send the output to another device?
1750         BEQ .1       No.
1760         ORA #$C0     Use $Cn00 (n=SLOT) so we can simulate a
1770         LDX #0       PR#n when we swap CSWL,H & TCSWL,H.
1780         BEQ .2       This creates a problem if SLOT <> 0 &
1790  .1     LDA CSWH     SLOT contains an 80-col card since PR#
1800         LDX CSWL     can activate card, but not de-activate.
1810  .2     STA TCSWH    No harm done, but it can be confusing.
1820         STX TCSWL
1830         JMP PAUSE2   Start out by waiting for a keypress.
1840  *--------------------------------
1850  STRT   LDA NLPP     'CALC' NLPP lines from STRTL,H.
1860         STA TEMPL    Adrs of the opcode just after the last
1870         LDA #0       one in column one.  Store in TOFARL,H
1880         STA TEMPH    to keep STRTL,H from going beyond it.
1890         JSR CALC
1900         >MOVD PCL,TOFARL
1910  COLM1  LDA #1       Always start in column one.
1920         STA COLCNT   Set COLCNT to 1
1930         >CMPD ENDL,STRTL   Have we finished?
1940         BCS NOESC    No, ENDL,H >= STRTL,H
1950         JSR CROUT    Yes, purge last printed line.
1960  ESC    JSR SWAP     <ESC> brings you here, too.
1970         >MSG M.BYE   Print end message.
1980         LDA BRUNFX   Restore the stack pointer
1990         STA STKPTR
2000         RTS          All done.
2010  NOESC  >CMPD STRTL,TOFARL  About to pass col 2?
2020         BCC NULINE   No, so continue
```

```
2030        LDX NCPP     Yes, so find the new first
2040        JSR MULT     line for the new first column.
2050        JSR CALC
2060        >MOVD PCL,STRTL
2070  NUPAGE LDX NSKP     Page breaks
2080        CPX #$FE
2090        BEQ PAUSE    Pause
2100        BCS FRMFD    Form feed
2110        CPX #0
2120  .1     BEQ STRT     No break - solid listing
2130        JSR CROUT    Yes, print NSKP lines
2140        DEX
2150        JMP .1
2160  *--------------------------------
2170  FRMFD  LDA #$8C
2180        JSR COUT
2190        JMP STRT
2200  *--------------------------------
2210  PAUSE  JSR CROUT    Print a <RETURN>
2220        JSR SWAP     Swap TCSWL,H & CSWL
2230  PAUSE2 >MSG M.PAUSE Print PAUSE msg
2240        JSR RDKEY
2250        JSR SWAP     Swap back
2260        JMP STRT     Do it all again
2270  *--------------------------------
2280  NULINE JSR CROUT    Print a <RETURN>
2290        LDA KBD      A key might have been pressed
2300        EOR #$9B     It might have been <ESC>
2310        BNE OFFSET   It wasn't; continue
2320        BIT STROBE   It was!  ESCape!
2330        JMP ESC
2340  OFFSET LDX COLCNT   Compute which opcode to
2350        JSR MULT     Disassemble next.
2360        JSR CALC
2370        >CMPD ENDL,PCL  Is adrs be beyond ENDL,H?
2380        BCC NEXTOP   Yes, don't bother with it
2390        LDX PCL      No, so disassemble it
2400        LDY PCH
2410        JSR PRYX2A   Print the opcode address
2420        LDX #1
2430        JSR PRBL2    Print 1 blank.  Monitor puts three
2440  *                  here, but if each column is no more
2450  *                  than 34 chars long, can fit 4 columns
2460  *                  onto a printer with 132 chars/line.
2470        JSR INSDS2   Format it
2480        JSR INSTDSPA   Print it
2490        LDA COLCNT   If last column, don't pad.
2500        CMP NCPP
2510        BEQ NXTCOL   It is, get out
2520        LDX #0       Isn't, so pad with blanks so that each
2530  *                  column takes exactly 34 characters.
2540        JSR INSDS2   Calculate the format code
2550        LDX #10      ASSUME 10 SPACES
2560        LDA FORMAT   Get it
```

```
        BEQ SPACE    1 byte code requires 10 spaces
        LDX #7       ASSUME 7 SPACES
        CMP #$81     Z-page
        BEQ SPACE
        DEX          ASSUME 6 SPACES
        CMP #$21     Immediate
        BEQ SPACE
        DEX          ASSUME 5 SPACES
        CMP #$82     Absolute
        BEQ SPACE    5 SPACES
        CMP #$85     Zpage,Y
        BEQ SPACE    5 SPACES
        CMP #$91     Zpage,X
        BEQ SPACE    5 SPACES
        CMP #$9D     Relative
        BEQ SPACE    5 SPACES
        LDX #3       All others
 SPACE  JSR PRBL2    Print (X-reg) many blanks
 NXTCOL INC COLCNT   Go to next column
        LDA NCPP
        CMP COLCNT   Have we gone too far?
        BCS OFFSET   No, do OFFSET
 NEXTOP LDA #1       Jump over the line
        STA TEMPL    just done.
        LDA #0
        STA TEMPH
        JSR CALC
        >MOVD PCL,STRTL    Store it in STRTL,H
        JMP COLM1    And do it all again
 *---------------------------------
 *  CALC returns the opcode adrs that is TEMPL,H
 *       disassembled (!) lines from STRTL,H
 *       It returns this address in PCL,H
 CALC   >MOVD STRTL,PCL   Put STRTL,H into PCL,H for INSDS1
 .1     LDA TEMPL    If TEMPL,H = 0 then done
        ORA TEMPH
        BEQ .3
        LDX #0
        JSR INSDS2   Get end of the next opcode & operand
        JSR PCADJ    Get the new address from PCADJ
        STA PCL      Store the resulting address in PCL,H
        STY PCH
        LDA TEMPL    DEC TEMPL,H - with help
        BNE .2       from the MACRO LIBRARY again!
        DEC TEMPH
 .2     DEC TEMPL
        CLV          Exit from top of loop, not here
        BVC .1       Always taken
 .3     RTS
 *--------------------------------
 *  MULT returns (NLPP * n-1).  N is usually
 *       COLCNT, and as such is usually a small
 *       number (almost always smaller than NLPP).
 *       So MULT simply adds NLPP to itself n times.
```

```
3110  *       Returns with result in TEMPL,H
3120  MULT   LDA #0       Zero TEMPL,H
3130         STA TEMPL
3140         STA TEMPH
3150  .1     CLC
3160  .2     DEX          Exit loop from top, so call with n+1
3170         BEQ .3       Anything times 0 equals 0
3180         LDA TEMPL    Add NLPP to TEMPL,H
3190         ADC NLPP
3200         STA TEMPL
3210         BCC .1       ...NO CARRY, KEEP ADDING
3220         INC TEMPH    ...CARRY
3230         BCS .1       ...ALWAYS
3240  .3     RTS
3250  *--------------------------------
3260  SWAP   LDA CSWL     Swap output device adrses.  They are
3270         LDX TCSWL    the same if SLOT = 0, but swap anyway.
3280         STX CSWL
3290         STA TCSWL
3300         LDA CSWH
3310         LDX TCSWH
3320         STX CSWH
3330         STA TCSWH
3340         RTS
3350  *--------------------------------
3360  PM.1   JSR COUT
3370         INX
3380  PRINT.MESSAGE
3390         LDA MSGS,X
3400         BMI PM.1
3410         RTS
3420  *--------------------------------
3430  MSGS
3440  M.PAUSE    .EQ *-MSGS
3450         .AT -'PRESS A KEY '
3460  M.BYE      .EQ *-MSGS
3470         .AT -'*** END OF LISTING '
3480  *--------------------------------
```

==============================================================
DOCUMENT :AAL-8510:DOS3.3:S.RWTS.SNOOPER.txt
==============================================================

```
 1000  *SAVE S.RWTS.SNOOPER
 1010  *---------------------------------
 1020  PRBYTE .EQ $FDDA
 1030  CROUT  .EQ $FD8E
 1040  COUT   .EQ $FDED
 1050  *---------------------------------
 1060  INSTALL.SNOOPER
 1070         LDX #1
 1080  .1     LDA DRIVER,X
 1090         PHA
 1100         LDA $B091,X
 1110         STA DRIVER,X
 1120         PLA
 1130         STA $B091,X
 1140         DEX
 1150         BPL .1
 1160         RTS
 1170  *---------------------------------
 1180  DRIVER .DA SNOOPER  MODIFIED DURING OPERATION
 1190  *---------------------------------
 1200  SNOOPER
 1210         LDA $778
 1220         STA SAVE778
 1230         LDA $7F8
 1240         STA SAVE7F8
 1250  *---------------------------------
 1260         TSX
 1270         JSR CROUT
 1280         JSR PRADDR   PRINT RETURN ADDR FROM STACK
 1290         JSR PRADDR   AND ANOTHER ONE
 1300  *---------------------------------
 1310         LDA $B7F4    COMMAND
 1320         JSR BYTE
 1330         LDA $B7EC    TRACK
 1340         JSR BYTE
 1350         LDA $B7ED    SECTOR
 1360         JSR BYTE
 1370         LDA $B7F1    BUFFER ADDRESS
 1380         JSR PRBYTE
 1390         LDA $B7F0
 1400         JSR PRBYTE
 1410  *---------------------------------
 1420         LDA SAVE778
 1430         STA $778
 1440         LDA SAVE7F8
 1450         STA $7F8
 1460         LDA $AAC2
 1470         LDY $AAC1
 1480         JMP (DRIVER)
```

```
1490  *--------------------------------
1500  PRADDR
1510         LDA $108,X
1520         JSR PRBYTE
1530         LDA $107,X
1540         DEX          SET UP FOR NEXT ADDRESS
1550         DEX
1560  BYTE   JSR PRBYTE
1570         LDA #"."
1580         JMP COUT
1590  *--------------------------------
1600  SAVEX  .BS 1
1610  SAVEY  .BS 1
1620  SAVE778 .BS 1
1630  SAVE7F8 .BS 1
1640  *--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8510:ProDOS:PRODOS.SNOOPER.txt
==================================================================
```

```
*SAVE PRODOS.SNOOPER
*--------------------------------
       .OR $300
       .OP 65C02    (If you have one)
*--------------------------------
SLOT   .EQ 6
DRIVES .EQ 2
*--------------------------------
BUFFER .EQ $800
*--------------------------------
A300   JMP INSTALL.SNOOPER
A303   JMP DISPLAY
*--------------------------------
INSTALL.SNOOPER
       LDX #1
.1     LDA 2*SLOT+$BF10,X
       PHA          SAVE CURRENT DRIVER ADDRESS
       LDA DRIVER,X INSTALL NEW DRIVER ADDRESS
       STA 2*SLOT+$BF10,X
  .DO DRIVES=2
       STA 2*SLOT+$BF20,X
  .FIN
       PLA          REMEMBER OLD DRIVER
       STA DRIVER,X
       LDA BUFFER.ADDR,X
       STA A+1,X
       DEX
       BPL .1       NOW THE OTHER BYTE
       RTS
*--------------------------------
DRIVER      .DA SNOOPER
BUFFER.ADDR .DA BUFFER
*--------------------------------
SNOOPER
       PHA
       PHY      (If no 65C02 use TYA, PHA)
       PHX      (If no 65C02 use TXA, PHA)
       TSX
       LDA $104,X   LO-BYTE OF RETURN ADDR
       JSR STORE.BYTE
       LDA $105,X   HI-BYTE OF RETURN ADDR
       JSR STORE.BYTE
       LDX #0       $42...47
.1     LDA $42,X    WHICH ARE THE PARAMETERS
       JSR STORE.BYTE     FOR THE CALL
       INX
       CPX #6
       BCC .1
       PLX      (If no 65C02 use PLA, TAX)
```

```
1500         PLY      (If no 65C02 use PLA, TAY)
1510         PLA
1520         JMP (DRIVER) CONTINUE IN DRIVER
1530  *---------------------------------
1540  STORE.BYTE
1550  A      STA BUFFER   THIS ADDRESS IS MODIFIED
1560         INC A+1      BUMP PNTR TO NEXT ADDRESS
1570         BNE .1
1580         INC A+2
1590  .1     RTS
1600  *---------------------------------
1610  COUT   .EQ $FDED
1620  CROUT  .EQ $FD8E
1630  PRBYTE .EQ $FDDA
1640  PNTR   .EQ $00,01
1650  *---------------------------------
1660  DISPLAY
1670         LDA #BUFFER  SET UP PNTR INTO BUFFER
1680         STA PNTR
1690         LDA /BUFFER
1700         STA PNTR+1
1710  *---CHECK IF FINISHED------------
1720  .1     LDA PNTR
1730         CMP A+1
1740         LDA PNTR+1
1750         SBC A+2
1760         BCC .2
1770         RTS
1780  *---DISPLAY NEXT 8 BYTES---------
1790  .2     LDY #1
1800         JSR WORD     DISPLAY RETURN ADDRESS
1810         LDA #":"     "XXXX:"
1820         JSR COUT
1830         JSR BYTE     DISPLAY ($42)=OPCODE
1840         JSR BYTE     DISPLAY ($43)=UNIT NUMBER
1850         INY
1860         JSR WORD     DISPLAY ($44,45)=BUFFER ADDR
1870         JSR DOT
1880         JSR WORD     DISPLAY ($46,47)=BLOCK NUMBER
1890         JSR CROUT    CARRIAGE RETURN
1900         LDA PNTR     ADVANCE PNTR TO NEXT
1910         CLC               GROUP OF 8 BYTES
1920         ADC #8
1930         STA PNTR
1940         BCC .1
1950         INC PNTR+1
1960         BNE .1       ...ALWAYS
1970  *---------------------------------
1980  WORD   LDA (PNTR),Y      DISPLAY HI-BYTE
1990         JSR PRBYTE
2000         DEY               DISPLAY LO-BYTE
2010         LDA (PNTR),Y
2020         INY
2030         INY               ADVANCE INDEX
```

```
2040        JMP PRBYTE
2060  *--------------------------------
2070  BYTE   LDA (PNTR),Y      DISPLAY BYTE
2080         JSR PRBYTE
2090  DOT    LDA #"."          PRINT "."
2100         INY               ADVANCE INDEX
2110         JMP COUT
2120  *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8511:Articles:Front.Page.txt
==========================================================================

$1.80

Volume 6 -- Issue 2                    November, 1985

In This Issue...



Programming the 65816

Last month we expected to have ready for this issue a review of David Eyes' new book on programming the 65816 microprocessor.  Well the books still haven't arrived, despite the passing of two promised shipping dates, so we're still waiting to see when they will really be available and what they come out like.  We are accepting orders (about 20 so far!) and will send out the books and publish a review as soon as they arrive from Prentice-Hall.

quikLoading AppleWorks

For you quikLoader owners who are also using AppleWorks (or for you AppleWorks enthusiasts who want your computer to instantly start up in AppleWorks), Southern California Research Group can now produce a set of quikLoader EPROMs from your configured AppleWorks program disks. The price for the EPROMs and the programming service is $89.50.  For more information call SCRG at (805) 529-2082.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8511:Articles:Kablit.txt
==============================================================================

Kablit Security System

After three burglaries or attempts here at the office, and four at Bill's house, we have been looking into ways to make our Apples a little more secure.  There are a variety of products available these days, some involving special furniture that locks up around computer, and others consisting of brackets that lock the equipment to the desk top.  These solutions seem too expensive and limiting for out purposes.  We are always shifting the computers and monitors around to install or remove cards or to connect or disconnect some accessory. And four computers here in the office and two more at our houses mean that the system had better be inexpensive.

Well we have found what looks to be the answer:  the Kablit Security System, from Secure-It, Inc.  This is 10 feet of 3/16" steel cable with a high-quality padlock-type lock and an assortment of special hardware to attach the cable to your computer, monitor, disk drives, printer, or whatever.  The connectors attach using the normal case screws of your equipment, so in most cases there is no need to drill holes or otherwise tear things up.  There are specific kits for the Apple //c and the Macintosh.

The list price of the Kablit Security System is $49.95; we will be offering them for $45 + shipping.

=============================================================================
DOCUMENT :AAL-8511:Articles:Merging.txt
=============================================================================

Two Ways to Merge Fields in a Byte.........Bob Sander-Cederlof

One of the advantages of assembly language is that data can be manipulated easily at the bit and byte level. This leads to efficiencies in both speed and memory usage which cannot be matched with most higher-level languages.

We can pack more than one data item into the same byte. For example, I may use the first three bits of a byte to indicate which of eight colors to use, and the other five bits to indicated position on a 32-pixel line. There are endless examples. Since we need to be able to store into and retrieve from bit-fields within bytes, all of the microprocessors include opcodes which make it possible.

To merge two values together which already are "clean", we simply use the ORA opcode. For example, if I have data for field A in VAL.A as xxx00000 and data for field B in VAL.B as 000xxxxx, I merge them like this:

```
       LDA VAL.A
       ORA VAL.B
```

By "clean" I mean that all the bits in VAL.A and VAL.B which are not part of the field values are already zero. If they are not, then we must first strip out those bits with the AND opcode:

```
       LDA VAL.A
       AND #$E0
       STA TEMP
       LDA VAL.B
       AND #$1F
       ORA TEMP
```

There is another way, which is shorter and faster and does not need TEMP. However, it is harder to figure out why it works.

```
       LDA VAL.A
       EOR VAL.B
       AND #$1F
       EOR VAL.A
```

Can you explain it? I was so unsure of myself when I first ran into this technique that I devised a test program. My test tries all 256 values of VAL.A and VAL.B, with all possible contiguous fields from 1 bit for VAL.A to 7 bits for VAL.A. Probably overkill, but it runs in a few seconds.

My program prints out the two field masks for each of the seven field sizes, so that I can tell it is running. If the two methods for merging get the same results, that is the only output. If they do not, indicating that one method or the other does not work, I print out more data.

While I was writing the program I tried several variations, such as printing all the results whether they agreed or not. In order to be able to look at that volume of output reasonably, I added a PAUSE subroutine which enabled me to stop the output by tapping any key, restart it the same way, and abort by tapping the RETURN key.

The code for the first merging method is in lines 1310-1380; that for the second at lines 1400-1450.

The test was conclusive. I tried every possible combination, and both methods always give the same results. Looking back, I can see that the whole test was unnecessary; the second method will OBVIOUSLY produce the same results. Now I see it. Do you?

```
==================================================================
DOCUMENT :AAL-8511:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.......................................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $32 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers...........................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *
Switch-a-Slot (SCRG)..........................................($190) $175 *
Extend-a-Slot (SCRG)...........................................($35)  $32 *
Kablit Security System (Secure-It, Inc.)....................($49.95)  $45 *

"Programming the 65816", Eyes...............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
```

```
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"6502 Subroutines", Leventhal...............................($18.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
              Foreign customers inquire for postage needed.
        Texas residents please add 6 1/8 % sales tax to all orders.

         *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
         ***                 (214) 324-2050                   ***
         *** We accept Master Card, VISA and American Express ***
```

================================================================================
DOCUMENT :AAL-8511:Articles:Object.Vector.txt
================================================================================

Using the Object Vector in S-C Macro Assembler......Bill Morgan

Sometimes we want to do something special with the object code generated by the S-C Macro Assembler. Maybe write it directly into an EPROM programmer, send it out through a serial port, or store it into some special device. One such device is the Douglas Electronics Writable ROM Board, which appears to the Apple as 2K of RAM at $C800 but brings out a cable that plugs right into a 2716 EPROM socket. With this card we can test the assembled code instantly in the target machine, without the delay and hassle of programming and transferring an EPROM.

There are a couple of hitches along the way. The assembler normally protects everything above $BFFF from code storage, and we need some special code because we have to temporarily switch off any other card using $C800, switch on the WROM Board, write a byte, and switch the WROM Board off again.

Fortunately, Version 2.0 of the S-C Macro Assembler has some special features for cases just like this. There are parameters at the beginning of the assembler to unprotect a specified area of memory, and each byte generated is passed through an Object Vector on its way to storage, so we can intercept the byte and do our memory switching before passing it back to the assembler.

Since the object code is going to be stored in successive memory locations pointed to by the Target Address, we can just use the Macro Assembler's normal STORE.OBJECT.BYTE routine. The address of STORE.OBJECT.BYTE is in the JMP instruction at OBJECT.VECTOR, so it's easy to get that address, plug it into our code, and then install our address in OBJECT.VECTOR. If we needed to do something different with the object code, like storing each byte into the same hardware register, we would do that instead at the line labelled CALL.

Writable ROM Board, by Douglas Electronics, 718 Marina Blvd., San Leandro, CA 94577. (415) 483-8770. $95.

================================================================
DOCUMENT :AAL-8511:Articles:PDos.Quit.Code.txt
================================================================

Commented Listing of ProDOS QUIT Code......Bob Sander-Cederlof

After reading Mark Jackson's article on improving the ProDOS QUIT code, I though it would be nice to have a commented listing of that program. The listing which follows is just that.

The ProDOS QUIT code is booted into $D100-D3FF in the alternate $D000 bank (the one you get by diddling $C083). Normally ProDOS MLI stays in the $C08B side. When a program issues the QUIT call (MLI code $65), the contents of $D100-D3FF are copied to $1000-12FF; then ProDOS jumps to $1000.

If you BLOAD the SYS file named PRODOS from a bootable ProDOS 1.1.1 disk, and examine it, you will find that it is laid out in eight parts. The first part is a relocator, which copies the other seven parts into their normal homes. Like this:

| Position as loaded | Position copied to | |
|---|---|---|
| 2000-29FF | --- | Relocator |
| 2A00-2BFF | Aux 200-3FF | /RAM/ driver |
| 2C00-2C7F | FF00-FF7F | /RAM/ driver |
| 2C80-2CFF | nowhere | All zeroes |
| 2D00-4DFF | D000-F0FF | MLI Kernel |
| 4E00-4EFF | BF00-BFFF | System Global Page |
| 4F00-4F7F | D742-D7BD | Thunderclock driver |
| 4F80-4FFF | FF80-FFFF | Interrupt Code |
| 5000-56FF | F800-FEFF | Device Drivers |
| 5700-59FF | D100-D3FF(alt) | QUIT Code |
| zeroes | F100-F7FF | |

The part I am interested in right now is the QUIT code, which is at $5700-$59FF in the PRODOS file.

The QUIT code is not written very efficiently. For some reason, there are two completely separate editing programs: one for the prefix, and another for the pathname. (And as Mark points out, neither one is very handy.) Even the code that initializes the BITMAP is inefficient.

```
================================================================
DOCUMENT :AAL-8511:Articles:ProDOS.Quit.txt
================================================================
```

An Easier QUIT from ProDOS.........................Mark Jackson
Chicago, IL

When using a hard disk with ProDOS it is often useful to use the MLI QUIT call to go from one application to another. However, if you are deep within a subdirectory the QUIT code makes you retype the entire Prefix if you want to shorten it. To allow the use of the right arrow during the QUIT call do the following:

```
UNLOCK PRODOS
BLOAD PRODOS,A$2000,TSYS
CALL-151
5764:75      (for ProDOS 1.1.1 -- use 5964 for 1.0.1)
BSAVE PRODOS,A$2000,TSYS
LOCK PRODOS
```

This changes the input call to $FD75 which allows right arrow input. There is one drawback: now to restore the prompted prefix you must press ESCape when asked for the Pathname of the next application.

==============================================================================
DOCUMENT :AAL-8511:Articles:Puzzle.Solves.txt
==============================================================================

Solutions to Adam Levin's Painting Puzzle....Adam Levin, et al

The puzzle, published last month, was to write a program which would fill all RAM from $0000 through $BFFF with the same value. What value is your choice.

The listing of my solution follows. It executes at $9966, which is inside the middle DOS buffer. To get it there, you can BLOAD or BRUN it. A few seconds after the screen fill's up with "Y" characters, the program has completely filled RAM from $0000 through $BFFD with $99.

Lines 1080-1200 fill all the RAM not occupied by my program (addresses $0000-$98FF and $99C8-$BFFF) with $99. I first fill the RAM from $99C8 up, and then from $0000 up through $98FF. You have to forgive the self-modifying code in a puzzle solution like this.

Lines 1210-1280 store a NOP and a JMP $0000 at the end of RAM. Lines 1320-1350 store $99 into $9900-$9999. It's getting hot in here!

Lines 1390-1590 get executed more than once. The first time, they store $99 into $999B-$99A3, and $99A5. By this time every byte from $0000 through $99A5 is set $99. All those bytes can be executed as "STA $9999,Y" instructions, and the JMP $0000 we placed at the end of RAM will do just that. When we get back up to line 1430, at $99A9, we start moving Y again and store $99 into $99A6-99AC and $99AE. It progressively keeps covering itself up, and eventually it is all gone:

```
       999B
       999C
       999D 99A6
       999E 99A7 99AF
       999F 99A8 99B0
       99A0 99A9 99B1 99B7
       99A1 99AA 99B2 99B8
       99A2 99AB 99B3 99B9 99BD
       99A3 99AC 99B4 99BA 99BE 99C1
       ...
       99A5 99A3 99B6 99BC 99C0 99C3 99C5 99C6
```

<<<<code here >>>.

Bob S-C's solution

The program loads at $800, but actually executes at $100. Lines 1030-1080 move the filler program down to $100 and jump to it. This solution fills all of RAM from $0000-BFFF with $48, which is a "PHA" instruction.

To keep from running off the end of RAM into the I/O space, I took advantage of the fact that the keyboard register can be read at both $C000 and $C001. Lines 1140-1160 wait until you type a zero key ("0"). The ASCII code for "0" is $B0. Two $B0 values in a row at $C000 and $C001 will dis-assemble as a BCS to $BFB2. Hence my solution finishes with an infinite loop running from $BFB2 to $C001.

Lines 1170-1290 fill RAM from $200-$BFFF with $48's, which are "PHA" opcodes. Lines 1300-1330 do the same with page zero.

Line 1350 jumps to $200, which means that the PHA opcodes start being executed. Since the stack is only 256 bytes long, and since the stack pointer wraps around, by the time the PHA at $2FF has executed all of page 1 will have been filled with $48. Since carry is set, when execution reaches $C000 the processor will go into that infinite loop I mentioned above.

<<<<code here>>>>

David Johnson's solution

My solution uses the power of the 65802. There was no restriction to the 6502 mentioned in the puzzle last month. All 49152 locations of motherboard RAM are filled with $DB, which happens to be the opcode value for the "STP" opcode. STP means "stop the processor", so once all RAM is filled it quits!

I use the MVP instruction to do the actual filling. The MVP instruction is located at $0000. I first put $DB into $BFFF. Then I set up the registers so that MVP will copy $BFFF into $BFFE, then $BFFE into $BFFD, and so on down to copying $0001 into $0000. By this time the MVP runs out, and the processor executes the STP opcode at $0003.

The 2nd and 3rd bytes of the MVP opcode specify which 64K memory banks to use; on a 65802 these don't do anything, because the bank addresses don't get out of the chip. On a 65816 my program won't work correctly, because the bank bytes will be changed at the end. First the Source bank address will be changed, so that a byte will be copied from $DB.0002 into $00.0001. Now the Destination Bank Address is changed, to we don't know what: we will finally copy $DB.0001 into $xx.0000. That last byte-move could be catastrophic (who knows, since we don't have any 65816-based systems yet?). Anyway, my program works fine in an Apple equipped with a 65802.

==========================================================================
DOCUMENT :AAL-8511:Articles:RAMDisk.txt
==========================================================================

Little DOS RAM Disk in Language Card.......Bob Sander-Cederlof

For some reason, we have until now avoided this subject. Many versions of RAM disks have been created and published in various magazines. The programs always seemed to me to be rather long and involved for what they really had to do. Recently a friend typed one in from Nibble, prompting me to try my hand.

The so-called "language card" is really the 16K RAM area. In //e and //c computers it is not a separate card at all, just the top 16K of the motherboard RAM. It received the monicker of "language card" because it was first sold as a separate card with the Pascal language system. The RAM in this area is not directly addressable, because the top 16K of Apple's address space is normally allocated to I/O ($C000-CFFF) and ROM ($D000-FFFF).

By flipping a few software-controlled switches the address range from $D000 through $FFFF can be made to point at the 16K RAM instead of ROM. Furthermore, the addresses from $D000 through $DFFF can be pointed at either of two 4K banks. If you have an Apple II or II+ with a 16K RAM card you already know this, of course.

Some programs use the language card under DOS, and some do not. Some which do are Integer BASIC, S-C Macro Assembler, Visicalc, Magicalc, Big Mac, and Merlin. If you are just using Applesoft to run your own programs, the language card is not used.

If the card is otherwise idle, that RAM could be used to simulate a small disk drive. My program sets it up as a 64 sector drive, with 60 sectors available for files. One sector is used for the VTOC, and three sectors are used for the catalog. You can save up to 21 files into the disk, or one file of up to 60 sectors.

One of the first questions I had to answer was where to put the program. Naturally, it ended up at $300. This is almost always my first choice, because it is so easy. If I find some substantial reasons, I try harder and find some other place in RAM for my programs. The ramdisk code could be placed inside DOS itself, on top of the RWTS format code. Another choice might be to use up one page of the language card for the bulk of the code, using only a few lines of code inside RWTS to switch it on and off. I like this idea, but it does deprive me of one sector out of 60. Anyway, for now let's just leave it at $300.

Another choice to be made is how to link into DOS. Many hard disks and other ramdisks do it by placing a JMP or JSR instruction at the beginning of RWTS ($BD00-BD02). This works very well, but it would be nice to be able to use both our ramdisk and any hard disk also. Therefore, I figured out a way to chain my ramdisk together with my

Sider hard disk. The method should be compatible with all the ramdisks and hard disks which patch in at $BD00.

The program is broken into two parts. The first part installs the ramdisk, and the second part performs the reads and writes. The installer loads and executes at $4000, but of course you could change it to whatever you wish.

I use six page zero locations. These are all locations which are used by regular RWTS, so it is all right for me to use them. I don't even need to save the original data and restore it when I am finished.

Lines 1090-1150 copy the read/write part down to $300-3B4. I actually copy a few extra bytes, but no harm done. I do have to be careful not to write any bytes above $3CF, because $3D0-3FF is already used by DOS and the monitor.

Lines 1160-1230 save the current contents of $BD00-BD02, and place a JMP to my ramdisk code there. Any future calls to RWTS will be vectored to my code down in page 3.

Lines 1250 and 1260 may look ridiculous, if you have not tried programming the language card before. The software-controlled switches ("soft switches") in the Apple are designed so that you have to make two references to address $C083 to turn it on and un-protect it. Two references to $C08B turn on the card also, but with the other 4K bank at $D000.

Lines 1270-1340 store zeroes in every byte from $D000-D3FF. In my scheme, those four pages are equivalent to four sectors (track $11, sectors 0-3). Now that I have mentioned that, why not tell you how I have laid out the whole 16K?

| Bank | Addresses | Trk | Sectors |
|---|---|---|---|
| C083 | D000-D3FF | $11 | 0-3 |
| C083 | D400-DFFF | $01 | 4-F |
| C08B | D000-DFFF | $02 | 0-F |
| | E000-EFFF | $03 | 0-F |
| | F000-FFFF | $04 | 0-F |

Lines 1350-1420 chain the three catalog sectors together. I have set up track $11 sector 3 as the first catalog sector, sector 2 as the second, and sector 1 as the third and last. This is the same kind of chain DOS makes on a real disk, but shorter.

Lines 1430-1500, together with the two data lines at 1550 and 1560, fill in the non-zero bytes in the VTOC sector. This table driven technique takes somewhat fewer bytes than direct code. I know, because the first time I wrote it the direct way: LDA, STA, LDA, STA, etc. The code as it now is plus the tables takes 45 bytes. The other way it takes 42 bytes just for the STA instructions. If I use LDA #$xx for each of the different values, that is another 16 bytes. So, I saved about 13 bytes. The TBLX line gives the offsets into the

$D000 page, and the TBLA line gives the data value which should be stored at each one. I use a 00 offset to indicate the end of the list.

Line 1580 tells the assembler to start assembling code to be executed at $300, but to keep putting the object code bytes in a continuous stream. Since we are writing the code on a target file (see line 1030), the whole program is on one file. RAMDISK.IMAG gets the value $4076, which is what the program counter is BEFORE the .PH directive takes effect. At line 1590 RAMDISK.REAL gets the value $300.

When a program calls RWTS, it is usually through as JSR $B7B5 instruction. The code at $B7B5 disables the interrupts and then does a JSR $BD00. We put our hook at $BD00, so the code jumps to $306, my label LITTLE.RAM.DISK. Lines 1650 and 1660 are the code which normally is executed at $BD00-BD03. They store the IOB address.

Lines 1670-1700 pick up the slot number out of the IOB. This is actually the slot number times 16. If the caller has specified slot 3, he wants to read or write the ramdisk. Any other slot, we need to let regular RWTS do the work. Lines 1710-1750 copy the original contents back to $BD00-BD02. Then I can call RWTS again, and this time it won't come back until it has done its job. Lines 1760-1780 restore Y and A as they were before we got involved, and re-call RWTS. When RWTS is finished, lines 1790-1830 put my hook back into $BD00-BD02. You might wonder if I should be saving and restoring the Y- and A-registers here. I originally did, saving them before line 1790 and restoring them before 1840. Then I realized that the normal contents of Y and A after visiting RWTS are not meaningful. Only the carry status bit is important, as it signifies whether there was an error or not.

If the caller specified slot 3, he wants to talk to our ramdisk. Lines 1860-1900 check to make sure he specified drive 1. If not, we call it an error. I funneled all of the messages through .99, setting the error byte in the IOB to $40. This causes DOS to say there was an I/O error.

I used an EOR #1 rather than CMP #1 at line 189~ so that if the drive was correct, we would also have 0 in the A-register. At some point I need to store 0 into RAMP, and this saves me a LDA #0 instruction. Then line 1910 can set RAMP to 0.

Lines 1930-1970 pick up the sector number the caller specified, and checks it for proper range. It must be from 0 to 15 to be valid. For the time being I save it in a handier location, RAMP+1.

Lines 1980-2020 and 2110-2120 check the track value. I will accept tracks 1-4 and $11, but no others. I have to accept $11, because that is where DOS always expects the VTOC to be, and where the catalog almost always is. The other four tracks could be anything I want, just so they are not $11. Since I am only using 4 sectors of track 11 for VTOC and catalog, I want the others to be usable for files. DOS refuses to allocate any sectors to files in track 11 unless we patch some code in the file manager, so I just put the rest of that bank of ram in another track.

Lines 2040-2060 make sure that if the caller wants track $11, his sector number is not bigger than 3. Lines 2130-2170 make sure that if the caller wants track 1, his sector number is not less than 4. If the track is either $11 or 1, lines 2070-2090 set us up to use the $C083 bank at $D000, with the sector specifying which page in that bank to use.

If the caller wants track 2, 3, or 4 then lines 2250-2310 set up the $C08B side, and compute the page number according to the table given above.

All this may be academic, because we have yet to look at the opcode. We are only implementing read and write, so if the opcode is something else we give an error. Lines 2340-2390 check the opcode, and also set the carry status for read or clear carry for write.

Lines 2400-2420 write enable the ramcard and select the proper $D000 bank. The value in the X-register is either 0 or 8, so we are either addressing $C083 or $C08B twice. We don't really need to write enable it unless the opcode was WRITE, but it doesn't hurt anything.

Lines 2430-2460 clear the error byte in the IOB. I could save two bytes by doing this above, just after line 1910.

Lines 2470-2530 pick up the caller's buffer address and store it in a pointer in page zero. I don't do any range checking on the buffer address, but then neither does RWTS.

Lines 2540-2550 set Y=0 to start the read or write loop, and then branch to the read loop if carry was set. Lines 2570-2610 comprise the write loop, and lines 2620-2660 the read loop.

Finally, line 2670 turns the language card back off. Then we clear carry status to indicate no errors, and return.

And that is how you make a ramdisk. If you have a bigger RAM card, it probably came with a ramdisk program. But if not, you ought to be able to see how to extend this program to handle larger amounts of memory.

==================================================================
DOCUMENT :AAL-8511:Articles:SathersComments.txt
==================================================================

Comments on O'Ryan's 65C02 Mod for Apple II.........Jim Sather

William O'Ryan's method (October 1985 AAL) of modifying old Apples to accept 65C02s looks like a very reliable fix. I notice no negative consequences in RAM or video timing. I do however recommend switching to 150 nanosecond motherboard RAM.

Apple motherboard RAM read access is CAS' limited, meaning TCAC (delay from CAS' falling to read data valid) is the critical RAM chip specification. In an Apple with O'Ryan's fix, RAM chips have 140 nsec minus 74LS139 pin 1 to pins 4,5,6 high/low propagation delay to get RAM read data valid after CAS' falls at the RAM chips. This means TCAC needs to be 119 nsec or less with a typical LS139. TCAC specifications are 100 nsec for 150 nsec RAM and 135 nsec for 200 nsec RAM, so 150 nsec or faster chips should be installed to be within RAM chip specifications with O'Ryan's fix.

A given Apple II may work with O'Ryan's fix and 200 nsec RAM chips, but operation may not be reliable over a wide range of room temperatures. Again I say, O'Ryan's fix calls for 150 nsec RAM chips. To operate with slower chips is asking for trouble.

Incidentally, 16K RAM chips don't cost as much as they used to. The cheapest 150 nsec 16K RAM chips I can find in my current mail order catalogs are 45 cents apiece at Jameco Electronics, 1355 Shoreway Rd., Belmont, CA 94002. [Slower ones were $65.00 apiece in 1978!]

As an alternative to replacing slow motherboard RAM chips, one can replace the 74LS139 at F2 with a 74S139. This changes the TCAC requirement with O'Ryan's fix to 133 nsec for a typical S139, and to 130 nsec for a worst case S139. These are barely less than the 135 nsec specification of 200 nsec RAM, so operation with 200 nsec RAM is probably reliable.

```
=============================================================================
DOCUMENT :AAL-8511:Articles:Words.On.MacAsm.txt
=============================================================================
```

Note on Mainstay MACASM for the Macintosh

We still have a small supply of the original release of this highly-praised development tool for the Macintosh. (Even Jerry Pournelle had good words for it.) I say original edition, because they are now at version 1.2, with 1.3 scheduled in January.

Mainstay has told us that there is little real difference in the various versions, not enough to influence your decision as to where to buy. And they also have a policy that they will provide your first upgrade absolutely free. All you need to do is fill in your registration card, make a backup copy of MacASM to use in the interim, and send them your original MacASM disk.

Their current end-user price is $125. Note that ours are still being sold at the intordunctory price of $100. Wow! It's a steal!

==============================================================================
DOCUMENT :AAL-8511:DOS3.3:DJohnsonsFiller.txt
==============================================================================

```
 1000  *SAVE DAVID JOHNSON'S FILLER
 1010  *-------------------------------
 1020  *      SOLUTION TO PUZZLE BY DAVID C. JOHNSON
 1030  *-------------------------------
 1040         .OP 65802    I got mine!
 1050  *-------------------------------
 1060         .OR $00
 1070  *-------------------------------
 1080  paint  mvp 0,0      fill $BFFE-$0000 from $BFFF
 1090  *-------------------------------
 1100  START  LDA #$DB     "STP" OPCODE
 1110         STA $BFFF    SEED FOR THE "MVP" INSTRUCTION
 1120         CLC          GET INTO NATIVE MODE
 1130         XCE
 1140         REP #$30     16-BIT REGISTERS
 1150         LDX ##$BFFF  Source Address = $BFFF
 1160         TXY
 1170         DEY          Destination Address = $BFFE
 1180         TYA          # Bytes -1 to be "moved"
 1190         BRA paint    MVP must be at $0000
 1200  *-------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8511:DOS3.3:LittleRamDisk.txt
==============================================================================

    1    .LIF
 1000  *SAVE S.LITTLE RAM DISK
 1010  *--------------------------------
 1020         .OR $4000
 1030         .TF B.LITTLE RAM DISK
 1040  *--------------------------------
 1050  RAMP   .EQ $3C,3D
 1060  BUFP   .EQ $3E,3F
 1070  IOB    .EQ $48,49
 1080  *--------------------------------
 1090  INSTALL
 1100         LDY #0       COPY CODE TO PAGE 3
 1110  .0     LDA RAMDISK.IMAG,Y
 1120         STA RAMDISK.REAL,Y
 1130         INY
 1140         CPY #$D0     NOT PAST $3CF
 1150         BCC .0
 1160  *---INSTALL DOS HOOK-------------
 1170         LDY #2
 1180  .1     LDA $BD00,Y
 1190         STA OLD.BD00,Y
 1200         LDA NEW.BD00,Y
 1210         STA $BD00,Y
 1220         DEY
 1230         BPL .1
 1240  *---INIT VTOC & CATALOG----------
 1250         LDA $C083
 1260         LDA $C083
 1270         INY          Y=0
 1280         TYA
 1290  .2     STA $D000,Y  CLEAR VTOC
 1300         STA $D100,Y  CLEAR THREE CATALOG PAGES
 1310         STA $D200,Y  ...ROOM FOR 21 FILES
 1320         STA $D300,Y
 1330         INY
 1340         BNE .2
 1350  *---CATALOG CHAIN----------------
 1360         LDA #$11     SIMULATED TRACK 11
 1370         STA $D201
 1380         STA $D301
 1390         INY          Y=1
 1400         STY $D202    POINT TO 3RD CATALOG SECTOR
 1410         INY          Y=2
 1420         STY $D302    POINT TO 2ND CATALOG SECTOR
 1430  *---FINISH THE VTOC-------------
 1440         LDY #0       USE TABLES FOR VTOC
 1450  .3     LDX TBLX,Y   INDEX INTO VTOC
 1460         BEQ .4       ...FINISHED
 1470         LDA TBLA,Y
```

```
1480        STA $D000,X
1490        INY
1500        BNE .3        ...ALWAYS
1510 *--------------------------------
1520 .4     LDA $C082     BACK TO MOTHERBOARD ROM
1530        RTS
1540 *--------------------------------
1550 TBLX   .HS 01.02.27.34.35.37.3C.3D.40.41.44.45.48.49.00
1560 TBLA   .HS 11.03.7A.23.10.01.FF.F0.FF.FF.FF.FF.FF.FF
1570 *--------------------------------
1580 RAMDISK.IMAG .PH $300
1590 RAMDISK.REAL
1600 *--------------------------------
1610 OLD.BD00   .BS 3
1620 NEW.BD00   JMP LITTLE.RAM.DISK
1630 *--------------------------------
1640 LITTLE.RAM.DISK
1650        STY IOB
1660        STA IOB+1
1670        LDY #1        LOOK AT SLOT NUMBER
1680        LDA (IOB),Y
1690        CMP #$30      RAMDISK IN SLOT 3
1700        BEQ RAM.DISK.SELECTED
1710        LDY #2
1720 .1     LDA OLD.BD00,Y
1730        STA $BD00,Y
1740        DEY
1750        BPL .1
1760        LDY IOB
1770        LDA IOB+1
1780        JSR $BD00
1790        LDY #2
1800 .2     LDA NEW.BD00,Y
1810        STA $BD00,Y
1820        DEY
1830        BPL .2
1840        RTS
1850 *--------------------------------
1860 RAM.DISK.SELECTED
1870        INY           LOOK AT DRIVE
1880        LDA (IOB),Y
1890        EOR #1        MUST BE DRIVE 1
1900        BNE .99       ...NOT DRIVE 1, ERROR
1910        STA RAMP      LO-BYTE OF RAMPAGE
1920 *--------------------------------
1930        LDY #5        GET SECTOR #
1940        LDA (IOB),Y
1950        CMP #16
1960        BCS .99       BAD T/S
1970        STA RAMP+1
1980        DEY           GET TRACK #
1990        LDA (IOB),Y
2000        BEQ .99       INVALID TRACK #
2010        CMP #$11      IS IT VTOC TRACK?
```

```
2020        BNE .2       NOT TRACK 17
2030 *---TRACK 17--------------------
2040        LDA RAMP+1        GET SECTOR #
2050        CMP #4            MUST BE 0-3
2060        BCS .99           NOT VALID T/S
2070 .1     ORA #$D0     FORM HI-BYTE OF ADDRESS
2080        LDX #0       C083 BANK
2090        BEQ .4       ...ALWAYS
2100 *---TRACK 1-4-------------------
2110 .2     CMP #5       OTHERWISE MUST BE TRACK 1-4
2120        BCS .99      NOT VALID T/S
2130        CMP #1       TRACK 1?
2140        BNE .3       ...NO
2150        LDA RAMP+1        GET SECTOR #
2160        CMP #4            MUST BE 4-F
2170        BCS .1            ...GOOD
2180 *---ERROR-----------------------
2190 .99    LDY #13
2200        LDA #$40
2210        STA (IOB),Y
2220        SEC
2230        RTS
2240 *-------------------------------
2250 .3     ASL          CHANGE 2,3,4 TO 20,30,40
2260        ASL
2270        ASL
2280        ASL
2290        ADC #$B0     ... TO D0,E0,F0
2300        ORA RAMP+1        MERGE SECTOR
2310        LDX #8       C08B BANK
2320 .4     STA RAMP+1
2330 *-------------------------------
2340        LDY #12      LOOK AT OPCODE
2350        LDA (IOB),Y
2360        BEQ .99      ...NOT RD OR WRT
2370        CMP #3       IS IT RD OR WRT?
2380        BCS .99      ...NO, IGNORE
2390        LSR          SET CARRY IF READ, CLR IF WRT
2400 *---SELECT RAMCARD BANK----------
2410        LDA $C083,X
2420        LDA $C083,X
2430 *---CLEAR ERROR CODE-------------
2440        LDY #13
2450        LDA #0
2460        STA (IOB),Y
2470 *---GET BUFFER ADDRESS-----------
2480        LDY #8
2490        LDA (IOB),Y
2500        STA BUFP
2510        INY
2520        LDA (IOB),Y
2530        STA BUFP+1
2540        LDY #0
2550        BCS .6       ...READ
```

```
2560  *---WRITE A SECTOR---------------
2570  .5     LDA (BUFP),Y
2580         STA (RAMP),Y
2590         INY
2600         BNE .5
2610         BEQ .7       ...ALWAYS
2620  *---READ A SECTOR----------------
2630  .6     LDA (RAMP),Y
2640         STA (BUFP),Y
2650         INY
2660         BNE .6
2670  .7     LDA $C082    BACK TO MOTHERBOARD ROM
2680         CLC
2690         RTS
2700  *--------------------------------
2710         .EP
2720         .LIF
```

```
===============================================================================
DOCUMENT :AAL-8511:DOS3.3:MergeFieldByte.txt
===============================================================================

 1000  *SAVE MERGE FIELDS IN A BYTE
 1010  *-------------------------------
 1020  CROUT  .EQ $FD8E
 1030  PRBYTE .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  *-------------------------------
 1060  FIELD.A    .EQ $00
 1070  FIELD.B    .EQ $01
 1080  VAL.A      .EQ $02
 1090  VAL.B      .EQ $03
 1100  MERGE.1    .EQ $04
 1110  MERGE.2    .EQ $05
 1120  *-------------------------------
 1130  T
 1140  *---FOR FIELD= 80,7F TO 7F,80----
 1150         LDA #$7F     DEFINE FIELDS AS 1,7
 1160         STA FIELD.B
 1170         LDA #$80
 1180         STA FIELD.A
 1190  *---FOR A=0 TO MAX VAL-----------
 1200  .1     LDA #0
 1210         STA VAL.A
 1220         JSR CROUT
 1230         LDA FIELD.A
 1240         JSR PRBYTESP
 1250         LDA FIELD.B
 1260         JSR PRBYTE
 1270  *---FOR B=0 TO MAX VAL-----------
 1280  .2     LDA #0
 1290         STA VAL.B
 1300
 1310  *---MERGE FIRST METHOD-----------
 1320  .3     LDA VAL.A
 1330         AND FIELD.A
 1340         STA MERGE.1
 1350         LDA VAL.B
 1360         AND FIELD.B
 1370         ORA MERGE.1
 1380         STA MERGE.1
 1390
 1400  *---MERGE SECOND METHOD----------
 1410         LDA VAL.A
 1420         EOR VAL.B
 1430         AND FIELD.B
 1440         EOR VAL.A
 1450         STA MERGE.2
 1460
 1470  *---PRINT RESULTS, IF NOT EQUAL--
 1480         CMP MERGE.1
```

```
1490        BEQ .4
1500        JSR CROUT
1510        LDA FIELD.A
1520        JSR PRBYTESP
1530        LDA VAL.A
1540        JSR PRBYTESP
1550        LDA VAL.B
1560        JSR PRBYTESP
1570        LDA MERGE.1
1580        JSR PRBYTESP
1590        LDA MERGE.2
1600        JSR PRBYTE
1610        JSR PAUSE
1620 *---NEXT B----------------------
1630 .4     INC VAL.B
1640        BNE .3
1650 *---NEXT A----------------------
1660        INC VAL.A
1670        BNE .2
1680 *---NEXT FIELD------------------
1690        SEC
1700        ROR FIELD.A
1710        LSR FIELD.B
1720        BNE .1        CONTINUE
1730        RTS           FINISHED
1740 *--------------------------------
1750 PRBYTESP
1760        JSR PRBYTE
1770        LDA #$A0
1780        JMP COUT
1790 *--------------------------------
1800 PAUSE  LDA $C000
1810        BPL .3
1820        STA $C010
1830        CMP #$8D
1840        BNE .2
1850 .1     PLA
1860        PLA
1870        RTS
1880 .2     LDA $C000
1890        BPL .2
1900        STA $C010
1910        CMP #$8D
1920        BEQ .1
1930 .3     RTS
1940 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8511:DOS3.3:S.RAMFill.Adam.txt
==============================================================================

 1000  *SAVE S.RAMFILL ADAM
 1010  *--------------------------------
 1020  *   ADAM LEVIN'S SOLUTION TO THE PUZZLE
 1030  *--------------------------------
 1040         .OR $9966    MUST START HERE
 1050         .TF B.PAINTER
 1060  *--------------------------------
 1070  PAINTER
 1080         LDY #END-1
 1090         LDA #$99     STORE $99 FROM END OF PROGRAM
 1100  COAT1  STA $9900,Y  THROUGH $BFFF
 1110         INY
 1120         BNE COAT1
 1130         INC COAT1+2  NEXT PAGE
 1140         LDX COAT1+2
 1150         CPX #$C0     REACHED $BFFF YET?
 1160         BNE .2       ...NOT YET
 1170         LDX #0       WRAP AROUND AND STORE FROM
 1180         STX COAT1+2  $0000 THRU $HERE
 1190  .2     CPX #$99     HAVE WE COME FULL CIRCLE?
 1200         BNE COAT1    ...NO, KEEP PAINTING
 1210         LDY #$EA     ...YES, NOW PATCH END OF RAM
 1220         STY $BFFB    FOR WRAPPING AROUND
 1230         STY $BFFC
 1240         LDY #$4C     NOP, JMP $0000
 1250         STY $BFFD
 1260         LDY #0
 1270         STY $BFFE
 1280         STY $BFFF
 1290  *--------------------------------
 1300  *   PAINT $9900-HERE
 1310  *--------------------------------
 1320  COAT2  STA $9900,Y
 1330         INY
 1340         CPY #COAT2+2
 1350         BCC COAT2
 1360  *--------------------------------
 1370  *   TRY TO GET OUT WITHOUT LEAVING FOOTPRINTS!
 1380  *--------------------------------
 1390         LDY #2       SET INDEX TO POINT TO $999B
 1400         STA $9999,Y
 1410         INY          $999C
 1420         STA $9999,Y
 1430         INY          $999D
 1440         STA $9999,Y
 1450         INY          $999E
 1460         STA $9999,Y
 1470         INY          $999F
 1480         STA $9999,Y
```

```
1490        INY          $99A0
1500        STA $9999,Y
1510        INY          $99A1
1520        STA $9999,Y
1530        INY          $99A2
1540        STA $9999,Y
1550        INY          $99A3
1560        STA $9999,Y
1570        INY          $99A4
1580        INY          $99A5
1590 END    STA $9999,Y
1600 *--------------------------------
```

```
=======================================================================
DOCUMENT :AAL-8511:DOS3.3:S.RAMFILL.RBSC.txt
=======================================================================

 1000  *SAVE S.RAMFILL RBSC
 1010  *--------------------------------
 1020         .OR $800
 1030  MOVER  LDY #LENGTH       MOVE "FILLER" PROGRAM
 1040  .1     LDA MY.FILLER,Y   TO EXECUTION AREA
 1050         STA FILLER,Y      AT $100...
 1060         DEY
 1070         BPL .1
 1080         JMP FILLER        NOW START FILLING!
 1090  *--------------------------------
 1100  *    FOLLOWING CODE EXECUTES AT $100...
 1110  *--------------------------------
 1120  MY.FILLER .PH $100
 1130  FILLER
 1140  .1     LDA $C000    WAIT UNTIL "0" TYPED
 1150         CMP #$B0     ($B0 IS ALSO BCS OPCODE)
 1160         BNE .1
 1170  *---FILL $200-$BFFF--------------
 1180         LDY #0
 1190         STY 0
 1200         LDA #2
 1210         STA 1
 1220         LDA #$48     PHA OPCODE
 1230  .2     STA (0),Y
 1240         INY
 1250         BNE .2
 1260         INC 1
 1270         LDX 1
 1280         CPX #$C0     UNTIL $BFFF
 1290         BCC .2
 1300  *---FILL PAGE ZERO---------------
 1310  .3     STA 0,Y
 1320         INY
 1330         BNE .3
 1340  *---FILL PAGE ONE----------------
 1350         JMP $200
 1360  LENGTH .EQ *-FILLER
 1370         .EP
 1380  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8511:DOS3.3:S.WROMWRITE.txt
==============================================================================

 1000  *SAVE S.WROMWRITE
 1010  *--------------------------------
 1020  LC       .EQ 1      1 if $D000 assembler
 1030  WROMSLOT .EQ 7
 1040  *--------------------------------
 1050       .DO LC
 1060  OBJECT.VECTOR  .EQ $D012
 1070  UNPROTECT.LOW  .EQ $D024
 1080  UNPROTECT.HIGH .EQ $D026
 1090       .ELSE
 1100  OBJECT.VECTOR  .EQ $1012
 1110  UNPROTECT.LOW  .EQ $1024
 1120  UNPROTECT.HIGH .EQ $1026
 1130       .FIN
 1140
 1150  WRITECARD   .EQ WROMSLOT*$10+$C080
 1160  CARDOFF     .EQ WROMSLOT*$10+$C081
 1170  C800.OFF    .EQ $CFFF
 1180
 1190  TARGET.LOW  .EQ $C800
 1200  TARGET.HIGH .EQ $CFFF
 1210  *--------------------------------
 1220         .OR $300
 1230  *      .TF WROMWRITE
 1240  INSTALL
 1250       .DO LC
 1260         BIT $C083
 1270         BIT $C083
 1280       .FIN
 1290         LDA /TARGET.LOW
 1300         STA UNPROTECT.LOW+1
 1310         LDA #TARGET.LOW
 1320         STA UNPROTECT.LOW
 1330         LDA /TARGET.HIGH
 1340         STA UNPROTECT.HIGH+1
 1350         LDA #TARGET.HIGH
 1360         STA UNPROTECT.HIGH
 1370         LDA OBJECT.VECTOR+2
 1380         STA CALL+2
 1390         LDA OBJECT.VECTOR+1
 1400         STA CALL+1
 1410         LDA /CARDON
 1420         STA OBJECT.VECTOR+2
 1430         LDA #CARDON
 1440         STA OBJECT.VECTOR+1
 1450       .DO LC
 1460         BIT $C080
 1470       .FIN
 1480         RTS
```

```
1490  *--------------------------------
1500  CARDON BIT C800.OFF
1510         BIT WRITECARD
1520  CALL   JSR $FFFF
1530         BIT CARDOFF
1540         RTS
```

```
================================================================================
DOCUMENT :AAL-8511:ProDOS:S.PRODOS.QUIT.txt
================================================================================

 1000  *SAVE S.PRODOS.QUIT
 1010  *--------------------------------
 1020  CH     .EQ $24
 1030  CV     .EQ $25
 1040  ERRCOD .EQ $DE
 1050  *--------------------------------
 1060  BUF    .EQ $0280
 1070  *--------------------------------
 1080  SYSTEM .EQ $2000
 1090  *--------------------------------
 1100  MLI    .EQ $BF00
 1110  BITMAP .EQ $BF58
 1120  *--------------------------------
 1130  KEY    .EQ $C000
 1140  S80STOREOFF .EQ $C000
 1150  S80OFF .EQ $C00C
 1160  SALTON .EQ $C00F
 1170  STROBE .EQ $C010
 1180  ROM    .EQ $C082
 1190  *--------------------------------
 1200  HOME   .EQ $FC58
 1210  CLREOL .EQ $FC9C
 1220  RDKEY  .EQ $FD0C
 1230  CROUT  .EQ $FD8E
 1240  COUT   .EQ $FDED
 1250  SETKBD .EQ $FE89
 1260  SETVID .EQ $FE93
 1270  BELL   .EQ $FF3A
 1280  *--------------------------------
 1290         .MA MLI
 1300         JSR MLI
 1310         .DA #$]1,]2
 1320         .EM
 1330  *--------------------------------
 1340         .OR $1000
 1350         .TA $5700
 1360  *--------------------------------
 1370  PRODOS.QUIT
 1380         LDA ROM      TURN ON THE MONITOR ROM
 1390         JSR SETVID   GET BACK TO GOOD OLD-FASHIONED
 1400         JSR SETKBD        DOWN-HOME 40 COLUMN DISPLAY
 1410         STA S80OFF
 1420         STA SALTON   Know what I mean, Vern?
 1430         STA S80STOREOFF
 1440  *---PREPARE BITMAP---------------
 1450         LDX #$17
 1460         LDA #1     Mark $BFxx in use
 1470         STA BITMAP,X
 1480         DEX
```

```
1490        LDA #0       Most pages are free
1500 .1     STA BITMAP,X
1510        DEX
1520        BPL .1
1530        LDA #$CF     $0000-01FF, $0400-07FF in use
1540        STA BITMAP
1550 *---DISPLAY PREFIX---------------
1560 GET.PREFIX
1570        JSR HOME
1580        JSR CROUT
1590        LDA #Q.PRFX
1600        STA MSG.ADDR
1610        LDA /Q.PRFX
1620        STA MSG.ADDR+1
1630        JSR PRINT.MESSAGE
1640        LDA #3       VTAB 4
1650        STA CV
1660        JSR CROUT    MAKE IT 5
1670        >MLI C7,PREFIX.PARM
1680        LDX BUF      # CHARS IN PREFIX
1690        LDA #0       MARK END OF PREFIX WITH 00
1700        STA BUF+1,X  SO OUR MESSAGE PRINTER WILL
1710        LDA #BUF+1       PRINT IT.
1720        STA MSG.ADDR
1730        LDA /BUF+1
1740        STA MSG.ADDR+1
1750        JSR PRINT.MESSAGE
1760 *---GET NEW PREFIX---------------
1770        LDX #0
1780        DEC CV       MOVE CURSOR TO BEGINNING OF LINE
1790        JSR CROUT
1800 NEXT.PREFIX.CHAR
1810        JSR RDKEY
1820        CMP #$8D
1830        BEQ SET.NEW.PREFIX   ...ACCEPT WHAT IS ON SCREEN
1840        PHA          ERASE PREFIX FROM SCREEN
1850        JSR CLREOL
1860        PLA
1870        CMP #$9B     IS CHAR <ESCAPE>?
1880        BEQ GET.PREFIX   ...YES, START ALL OVER
1890        CMP #$98     IS CHAR CTRL-X?
1900 START.PREFIX.OVER
1910        BEQ GET.PREFIX    ...START ALL OVER
1920        CMP #$89     IS CHAR <TAB>?
1930        BEQ .3       ...YES, RING BELL
1940        CMP #$88     IS CHAR BACKSPACE?
1950        BNE .2       ...NO, APPEND TO LINE
1960        CPX #0       ...BACKSPACE, UNLESS AT BEGINNING
1970        BEQ .1       AT BEGINNING ALREADY
1980        DEC CH       BACK UP
1990        DEX
2000 .1     JSR CLREOL   CHOP OFF AFTER CURSOR
2010        JMP NEXT.PREFIX.CHAR
2020 .2     BCS .4       OTHER CONTROL CHAR < $88
```

```
2030  .3     JSR BELL
2040         JMP NEXT.PREFIX.CHAR
2050  .4     CMP #"Z"+1
2060         BCC .5       ...NOT LOWER CASE
2070         AND #$DF     CONVERT LOWER CASE TO UPPER
2080  .5     CMP #"."     ALLOW PERIOD, SLASH, DIGITS
2090         BCC .3       ...TOO SMALL
2100         CMP #"Z"+1   ALLOW LETTERS
2110         BCS .3       ...TOO LARGE
2120         CMP #"9"+1
2130         BCC .6       ...PERIOD, SLASH, OR DIGIT
2140         CMP #"A"
2150         BCC .3       ...NOT A LEGAL CHARACTER
2160  .6     INX
2170         CPX #$27
2180         BCS START.PREFIX.OVER   ...TOO LONG
2190         STA BUF,X
2200         JSR COUT     ECHO THE CHARACTER
2210         JMP NEXT.PREFIX.CHAR
2220  *--------------------------------
2230  SET.NEW.PREFIX
2240         CPX #0       DID WE CHANGE IT?
2250         BEQ GET.PATHNAME   ...NO
2260         STX BUF      ...YES, SO TELL SYSTEM
2270         >MLI C6,PREFIX.PARM
2280         BCC GET.PATHNAME   ...NO ERRORS
2290         JSR BELL     DING, DONG!
2300         LDA #0       SET .EQ. STATUS
2310  PFXOVR BEQ START.PREFIX.OVER    ...ALWAYS
2320  *--------------------------------
2330  GET.PATHNAME
2340         JSR HOME
2350  START.PATHNAME.OVER
2360         JSR CROUT
2370         LDA #Q.PATH
2380         STA MSG.ADDR
2390         LDA /Q.PATH
2400         STA MSG.ADDR+1
2410         JSR PRINT.MESSAGE
2420         LDA #3       VTAB 4
2430         STA CV
2440         JSR CROUT    MAKE IT 5
2450         LDX #0
2460  NEXT.PATHNAME.CHAR
2470         LDA #$FF     CURSOR CHARACTER
2480         JSR COUT
2490         DEC CH       BACK UP OVER CURSOR
2500  .1     LDA KEY
2510         BPL .1       ...WAIT TILL KEY PRESSED
2520         STA STROBE
2530         CMP #$9B     <ESCAPE>?
2540         BNE .2       ...NO
2550         LDA CH       IF AT BEGINNING, GET PREFIX OVER
2560         BNE GET.PATHNAME      ...ELSE GET PATHNAME OVER
```

```
2570          BEQ PFXOVR
2580  .2      CMP #$98     CONTROL-X?
2590  .3      BEQ GET.PATHNAME
2600          CMP #$89     TAB KEY?
2610          BEQ .5       ...YES
2620          CMP #$88     BACKSPACE?
2630          BNE .4       ...NO
2640          JMP BACKSPACE.IN.PATHNAME
2650  *--------------------------------
2660  .4      BCS .6
2670  .5      JSR BELL     ...INVALID CHAR, RING BELL
2680          JMP NEXT.PATHNAME.CHAR
2690  *--------------------------------
2700  .6      CMP #$8D
2710          BEQ SET.NEW.PATHNAME
2720          CMP #"Z"+1
2730          BCC .7
2740          AND #$DF     CHANGE LOWER CASE TO UPPER
2750  .7      CMP #"."     ACCEPT DOT, SLASH, OR DIGIT
2760          BCC .5       ...TOO SMALL
2770          CMP #"Z"+1   ACCEPT LETTERS
2780          BCS .5       ...TOO LARGE
2790          CMP #"9"+1
2800          BCC .8       ...DOT, SLASH, OR DIGIT
2810          CMP #"A"
2820          BCC .5       ...NOT A VALID CHARACTER
2830  .8      PHA          CLEAR BEYOND THIS POINT
2840          JSR CLREOL
2850          PLA
2860          JSR COUT     ECHO THE NEW CHARACTER
2870          INX
2880          CPX #$27
2890          BCS .3       ...NAME TOO LONG
2900          STA BUF,X    APPEND CHAR TO NAME
2910          JMP NEXT.PATHNAME.CHAR
2920  *--------------------------------
2930  SET.NEW.PATHNAME
2940          LDA #" "
2950          JSR COUT
2960          STX BUF
2970          >MLI C4,FILE.INFO.PARM
2980          BCC .1       ...NO ERRORS
2990          JMP PROCESS.ERROR
3000  *--------------------------------
3010  .1      LDA FILTYP   FILE.INFO.PARM+4
3020          CMP #$FF
3030          BEQ .2       "SYS" FILE
3040          LDA #1
3050          JMP PROCESS.ERROR
3060  *--------------------------------
3070  .2      LDA #0
3080          STA CL.REF   CLOSE.PARM+1, REF NO.
3090          >MLI CC,CLOSE.PARM
3100          BCC .3       ...NO ERROR
```

```
3110        JMP PROCESS.ERROR
3120 *--------------------------------
3130 .3     LDA ACBITS   FILE.INFO.PARM+3
3140        AND #1
3150        BNE .4       ...OKAY TO READ IT
3160        LDA #$27
3170        JMP PROCESS.ERROR
3180 *--------------------------------
3190 .4     >MLI C8,OPEN.PARM
3200        BCC .5       ...NO ERRORS
3210        JMP PROCESS.ERROR
3220 *--------------------------------
3230 .5     LDA OP.REF   OPEN.PARM+5, REF NO.
3240        STA RD.REF   READ.PARM+1, REF NO.
3250        STA EF.REF   EOF.PARM+1, REF NO.
3260        >MLI D1,EOF.PARM
3270        BCC .6       ...NO ERRORS
3280        JMP PROCESS.ERROR
3290 *--------------------------------
3300 .6     LDA FIL.SZ+2    EOF.PARM+4
3310        BEQ .7       ...NOT TOO LONG
3320        LDA #$27
3330        JMP PROCESS.ERROR
3340 *--------------------------------
3350 .7     LDA FIL.SZ      EOF.PARM+2
3360        STA READ.PARM+4
3370        LDA FIL.SZ+1    EOF.PARM+3
3380        STA READ.PARM+5
3390        >MLI CA,READ.PARM
3400        PHP
3410        >MLI CC,CLOSE.PARM
3420        BCC .9
3430        PLP
3440 .8     JMP PROCESS.ERROR
3450 *--------------------------------
3460 .9     PLP
3470        BCS .8
3480        JMP SYSTEM
3490 *--------------------------------
3500 BACKSPACE.IN.PATHNAME
3510        LDA CH       UNLESS ALREADY AT BEGINNING
3520        BEQ .1       ...WE WERE
3530        DEX
3540        LDA #" "
3550        JSR COUT
3560        DEC CH
3570        DEC CH
3580        JSR COUT
3590        DEC CH
3600 .1     JMP NEXT.PATHNAME.CHAR
3610 *--------------------------------
3620 PRINT.MESSAGE
3630        LDX #0
3640 MSG.LP LDA MSG.LP,X
```

```
3650  MSG.ADDR   .EQ *-2
3660         BEQ .1
3670         ORA #$80
3680         JSR COUT
3690         INX
3700         BNE MSG.LP
3710  .1     RTS
3720  *---------------------------------
3730  PROCESS.ERROR
3740         STA ERRCOD
3750         LDA #12      VTAB 13
3760         STA CV
3770         JSR CROUT    MAKE IT 14
3780         LDA ERRCOD
3790         CMP #1
3800         BNE .1
3810         LDA #ERQT.1
3820         STA MSG.ADDR
3830         LDA /ERQT.1
3840         STA MSG.ADDR+1
3850         BNE .3
3860  .1     CMP #$40
3870         BEQ .2
3880         CMP #$44
3890         BEQ .2
3900         CMP #$45
3910         BEQ .2
3920         CMP #$46
3930         BEQ .2
3940         LDA #ERQT.2
3950         STA MSG.ADDR
3960         LDA /ERQT.2
3970         STA MSG.ADDR+1
3980         BNE .3       ...ALWAYS
3990  .2     LDA #ERQT.3
4000         STA MSG.ADDR
4010         LDA /ERQT.3
4020         STA MSG.ADDR+1
4030  .3     JSR PRINT.MESSAGE
4040         LDA #0       VTAB 1
4050         STA CV
4060         JMP START.PATHNAME.OVER
4070  *---------------------------------
4080  Q.PRFX .AS -/ENTER PREFIX (PRESS "RETURN" TO ACCEPT)/
4090         .HS 00
4100  Q.PATH .AS -/ENTER PATHNAME OF NEXT APPLICATION/
4110         .HS 00
4120  ERQT.1 .HS 87
4130         .AS -/NOT A TYPE "SYS" FILE/
4140         .HS 00
4150  ERQT.2 .HS 87
4160         .AS -"I/O ERROR            "
4170         .HS 00
4180  ERQT.3 .HS 87
```

```
4190        .AS -"FILE/PATH NOT FOUND  "
4200        .HS 00
4210 *--------------------------------
4220 FILE.INFO.PARM
4230        .DA #10
4240        .DA BUF
4250 ACBITS .HS 00
4260 FILTYP .HS 00
4270        .BS 13
4280 *--------------------------------
4290 OPEN.PARM
4300        .DA #3
4310        .DA BUF
4320        .DA $1800    BUFFER ADDR
4330 OP.REF .BS 1        REF NO.
4340 *--------------------------------
4350 CLOSE.PARM
4360        .DA #1
4370 CL.REF .BS 1        REF NO.
4380 *--------------------------------
4390 READ.PARM
4400        .DA #4
4410 RD.REF .BS 1        REF NO.
4420        .DA $2000    BUFFER ADDR
4430        .BS 2        # BYTES TO READ
4440        .BS 2        # ACTUALLY READ
4450 *--------------------------------
4460 EOF.PARM
4470        .DA #2
4480 EF.REF .BS 1        REF NO.
4490 FIL.SZ .BS 3        EOF POSITION
4500 *--------------------------------
4510 PREFIX.PARM
4520        .DA #1
4530        .DA BUF
4540 *--------------------------------
4550        .LIF
```

```
=====================================================================
DOCUMENT :AAL-8512:Articles:Day.Of.Week.txt
=====================================================================
```

Computing the Day of Week...............Bob Sander-Cederlof

Within reasonable limits, it should be possible for a clock/ calendar card to automatically set the day-of-week number when given the year, month, and day. The algorithm for deriving day-of-week from the date is simple enough. However, as the algorithm is stated in all my reference material, it involves multiplication and division by numbers that are not simple powers of two.

I have simplified the algorithm so that it will work over the range from March 1, 1984 through December 31, 2083. That should be an adequate range for any Apple products!

Years evenly divisible by 4 are leap years, having 366 days. The years ending in 00 are exceptions, unless they are divisible by 400. Thus 1900 was not a leap year, 2100 will not be a leap year, but 2000 is a leap year.

My algorithm started out as a method for converting a Y-M-D date to a Julian date, which is a unique number that was 0 several thousand years ago. I could get the remainder after dividing the Julian date by 7, and use it for a day-of-week index. However, the numbers get rather large; they won't fit in one byte.

By converting all the intermediate values to their modulo 7 equivalents, I can keep the result down to byte-size. Here is an Applesoft program which implements my algorithm:

```
100 DIM MD(11),D$(6)
110 DATA 3,6,1,4,6,2,5,0,3,5,1,4
120 DATA SUN,MON,TUES,WEDNES,THURS,FRI,SATUR
130 FOR I=0 TO 6 : READ M : MD(I)=M : NEXT
140 FOR I=0 TO 11: READ D$: D$(I)=D$: NEXT
200 INPUT Y,M,D
210 M = M-3
220 IF M<0 THEN M=M+12 : Y=Y-1
230 Y=Y-1984
240 W = Y + INT(Y/4) + MD(M) + D
250 IF W>6 THEN W=W-7 : GO TO 250
260 PRINT D$(W)"DAY"
270 GO TO 200
```

Lines 100-140 build two arrays. The MD array holds a modulo 7 number for the number of days preceding each month in a normal year (not leap year). The D$ array holds the names of the days, shortened by the last three letters.

Line 200 waits for you to type in the year, month, and day as three numbers. I did not add any error testing, but I expect the year to be from 1984 up. The month should be a number from 1 to 12, and the day from 1 to 31.

Lines 210-220 adjust the month number. I move January and February to the end of the previous year, like it must have been in the olden days. That makes leap day the last day of the year, where it belongs. It also makes the month names for Sept-, Oct-, Nov-, and Dec-ember make linguistic sense! March becomes the first month, December the tenth, and so on. Internally, the value of the variable M will be a number from 0 to 11.

Line 230 adjusts the year to start at 1984. Line 240 adds up the various day-values. We add Y, the number of the years since 1984, because 365 = 1 mod 7. We add INT(Y/4) to get the leap days. MD(M) adds in the bias for the number of days beyond an integral number of weeks to the end of the previous month. D adds in the day number. Altogether we have a number which is still less than 256, and fits in one byte in a machine language version of the algorithm.

Line 250 subtracts 7 (whole weeks) until we get to a number less than 7. The result is the day number in a week with 0 meaning Sunday, 1 meaning Monday, and so on. Line 260 prints the day name, and line 270 lets us try another date.

After making sure of my method with the Applesoft program, I coded it in assembly language. The program which follows is set up to be used from inside Applesoft, and I also list here the Applesoft driver. I did it this way to make it easy to test my assembly language code. Later I will probably put the code inside a larger package which sets the time and day on my clock card. Once it is in there, I can forever forget about the need to tell the card what day of week it is.

Lines 1020-1050 are the variables used to communicate with the Applesoft test program, by way of PEEKs and POKEs. The program assumes that only the last two digits of the year are used, so that YEAR is a number from 84 to 99 for 1984 to 1999; values from 0 to 83 signify years from 2000 to 2083.

Lines 1080-1130 change the year number, which runs 84...99 and 00...83 to a value based at 1984, running from 00 to 99. 00 means 1984, 99 means 2083.

Lines 1150-1210 are equivalent to the Applesoft lines 210 and 220 in the first program above. Lines 1220-1290 are equivalent to the Applesoft line 240. Lines 1300-1340 reduce the result to a modulo 7 remainder. The final value, a number from 0 to 6, is stored in line 1350 where an Applesoft driver can find it by PEEK(771).

Here is my Applesoft test program. This time I went in for a little range checking on the input values for year, month, and day.

```
100 DIM D$(6)
110 DATA SUN,MON,TUES,WEDNES,THURS,FRI,SATUR
120 FOR I=0 TO 6 : READ M : MD(I)=M : NEXT
```

```
200 INPUT "YEAR (1984-2083):  ";Y
210 IF Y<1984 OR Y>2083 THEN 200
220 Y = Y - INT(Y/100)*100
230 POKE 768,Y
300 INPUT "    MONTH (1-12):  ";M
310 IF M<1 OR M>12 THEN 300
320 POKE 769,M
400 INPUT "      DAY (1-31):  ";D
410 IF D<1 OR D>31 THEN 400
420 POKE 770,D
500 CALL 772
510 W = PEEK(771)
600 PRINT D$(W)"DAY"
610 GO TO 200
```

=================================================================
DOCUMENT :AAL-8512:Articles:Front.Page.txt
=================================================================

$1.80

Volume 6 -- Issue 3 December, 1985

In This Issue...



Little RAM Disk Bug

Does that mean a Bug in the Little RAM Disk, or a Little Bug in the RAM Disk? Actually, both. Several of you have called or written to point out a problem in Bob's program last month.

The TAY instruction at line 1280 (on page 8) should be a TYA. It does seem pointless to force Y to zero and then immediately clobber it with whatever the processor read out of $C083. This code worked when Bob tested it on his //e, because that computer does return a zero when you read $C083. My ][+, on the other hand, returns a byte of video data, usually $A0, and that really makes a mess out of the VTOC and Catalog sectors.

There's one other glitch in that article as well. In the fifth paragraph on page 5 there is a reference to line number "189~". I bet you can guess that's really supposed to be "1890".

So, thanks to all of you who caught us on this one! It's nice to know you're keeping an eye on us.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).



=================================================================
DOCUMENT :AAL-8512:Articles:Kashmarek.Trace.txt
=================================================================

**ProDOS MLI Tracing................................Ken Kashmarek**
**Eldridge, Iowa**

I took Bob S-C's work with ProDOS Snooper (October 1985 AAL) one step further: I added MLI calls to the information that is collected in the trace table. By combining the MLI call data with the device driver data, we get a better idea of what is happening.

The entries below all come from slot 6 drive 1. MLI calls are tagged with an "M" after the hex data. To support both the MLI calls and device driver calls, the hex output provides the data as it exists in memory without taking into account whether a set of bytes is a two byte memory pointer or a single data byte.

For all calls, the return address is still shown as hi-byte first before the colon. Data for the device driver parameter is still from $42-$47. For MLI calls, the return address is to the program that called the routine in the BASIC.SYSTEM global page. All BASIC.SYSTEM calls go to the $BE00 global page and then to the $BF00 ProDOS global page. MLI data is the MLI call number followed by the first five bytes of the parameter list (some bytes do not apply if the list is shorter).

The volume in question is labeled /TEST and has one file, ABC, in the root directory.

First of all, issue: CAT,S6

```
A6E9:C7 BC BC 02 BC BC M GET PREFIX
A85F:C5 60 01 02 00 03 M ON LINE CALL   + Not used when
EC0C:01 50 00 DC 02 00   READ BLOCK 2   + CAT /TEST entered
A825:C4 BC BC C3 0F 00 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:01 60 00 DC 06 00   READ Bit Map
B1B9:C8 BC BC 00 8A 01 M OPEN FILE
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EE85:01 60 00 8A 02 00   READ BLOCK 2
B175:CA 01 59 02 2B 00 M READ FILE
B201:CE 01 2B 00 00 03 M SET FILE MARK  + Appears for each
B208:CA 01 59 02 27 00 M READ FILE      + file in directory
B0A5:CC 01 00 C3 CF D0 M CLOSE FILE
B0FB:C5 60 BD BC 00 03 M ON LINE CALL
EC0C:01 50 00 DC 02 00   READ BLOCK 2
B10F:C4 BC BC C3 0F 18 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:01 60 00 DC 06 00   READ Bit Map
```

For this simple operation, there are ten MLI calls and eight device driver calls (disk I/O operations). I do not understand the reason for the Get Prefix call at the beginning. It would appear that the On Line call and the Get File Info call at the end are unnecessary (we will be checking this out as we go). On Line returns the volume name, but this should already be available through the prefix or pathname of the directory. Get File Info information should already be available from the previous call, and the bit map was already read in once. However, this is a simple catalog operation and may be indicative of some of the steps necessary for more complex catalog operations.

Carrying this one step further, I issued CAT /TEST/DIR. In this case, the first read of the bit map is not performed. Next, the former apparently duplicate read of block 2 now turns into a read of block 7, the key block for subdirectory DIR (in /TEXT/DIR; the device driver return address is $EE85, the buffer address is $8A00). Note: block 2 is the key block of the root directory.

A Get File Info call for a volume name (/TEST) always reads the bit map. Therefore, this call is repeated when cataloging a volume, but not when cataloging a subdirectory. As to the On Line call, it is used to get volume name for the Get File Info call for the free space information for the volume, since the initial catalog command may have been for a subdirectory. This explains (only partially) what appeared to be duplicate reads of the same information.

Now, let's try loading an Applesoft file: LOAD ABC,S6

```
A85F:C5 60 01 02 00 03 M ON LINE CALL   + Not used for
EC0C:01 60 00 DC 02 00   READ BLOCK 2   + LOAD /TEST/ABC
A825:C4 BC BC E3 FC 01 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
AC00:CC 00 00 C3 CF D0 M CLOSE ALL FILES
B1B9:C8 BC BC 00 8A 01 M OPEN FILE
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EE85:01 60 00 8A 07 00   READ BLOCK 7
AC22:D1 01 01 02 00 03 M GET FILE EOF
AC4B:CA 01 01 08 09 00 M READ FILE
AC50:CC 01 00 C3 CF D0 M CLOSE FILE
```

The loaded program is less than 512 bytes in length, so the key block read is the only data I/O operation. As with the catalog operation, the Get File Info call is used to verify the file type. Close All Files is used in case the previous program left any open. Note the Get File EOF call which is used to get the length for the Read File call (which performs the entire load operation). This example is relatively simple. Let's check what happens when we create an Applesoft file that is just over 512 bytes in length (changing our seedling file into a sapling file, which requires an index block and two data blocks).

We'll lengthen the program, and then type: SAVE /TEST/ABC.3

```
A825:C4 BC BC C3 0F 18 M GET FILE INFO
EC0C:01 60 00 DC 02 00   READ BLOCK 2
ACDC:C0 BC BC C3 FC 01 M CREATE FILE
EC0C:01 60 00 DC 02 00   READ BLOCK 2
F477:00 60 00 DC 00 00   STATUS S6,D1
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DC 07 00   WRITE BLOCK 7
EC0C:01 60 00 DC 02 00   READ BLOCK 2
```

```
EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
B1B9:C8 BC BC 00 8A 01 M OPEN FILE CALL
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EE85:01 60 00 8A 07 00   READ BLOCK 7
AD0A:CB 01 01 08 5B 02 M WRITE FILE CALL
F477:00 60 01 08 00 00   STATUS S6,D1
EE85:02 60 00 8A 07 00   WRITE BLOCK 7
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
EC0C:01 60 00 DA 06 00   READ BIT MAP
AD11:D0 01 5B 02 00 03 M SET FILE E0F CALL
AD16:CC 01 00 C3 CF D0 M CLOSE FILE CALL
EE85:02 60 00 8A 09 00   WRITE BLOCK 9
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
```

This sequence has the same number of MLI calls for a seedling or a sapling file. The big difference is allocating the index block (block number 8) and additional data blocks. This also generates additional calls to read and write the bit map.

If the file already exists, and the SAVE command does not change the length, then the Create File call is not executed, there are no accesses to the bit map (block 6), and the index block does not change. If the file length changes sufficiently to add or delete blocks, then the bit map is updated and the index block is rewritten (this is forced by the Set File EOF call which adjusts the file length).

Interesting note: whenever a file is opened, the first data block is always read in, even if the file will subsequently be written to. Likewise, when a new file is allocated, the first data block is allocated and written, even if no data is placed in the block.

In the above sequence, what appears to be a duplicate read of block 2 (return address $EC0C) is actually a read to separate blocks if the SAVE command was to a subdirectory. It turns out to be duplicate reads to the subdirectory block, write to the subdirectory, then read and write the root directory. Sigh.

LOAD /TEST/ABC.3 is similar to the previous load operation, except that we must also read the index block before reading the data blocks, and there are two data blocks rather than one.

Finally, let's try deleting this file: DELETE /TEST/ABC.3

```
A825:C4 BC BC E3 04 00 M GET FILE INFO CALL
EC0C:01 60 00 DC 02 00   READ BLOCK 2
9AD7:C1 BC BC 02 BC BC M DESTROY FILE CALL
```

```
EC0C:01 60 00 DC 02 00   READ BLOCK 2
F477:00 60 00 DC 00 00   STATUS S6,D1
EC0C:01 60 00 DC 08 00   READ BLOCK 8 (index block)
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DC 08 00   WRITE INDEX BLOCK (zeroed)
EC0C:01 60 00 DC 07 00   READ BLOCK 7
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
```

Again, use Get File Info for file type and status call to see if the disk can be written to. The bit map is read and written to reflect the freed blocks. Block 8, the former file index block, is trashed. I don't know why block 7 is read in. Trashing the index block makes it very hard to reconstruct a DELETEd file.

At this point, we get a feel for what is happening between the MLI calls and the device driver calls. Consider how extensive these simple examples become on a hard disk if working down three or four directory levels and at the second, third, or fourth block in each directroy, and the hard disk has five blocks for the bit map (and we need the fifth block because the disk is almost full). Ouch!

I performed one more test case, far too long to list here. It involved adding a record to a new sparse random access file. The new record caused the file to grow to a tree file. The program used was:

```
10 D$ - CHR$(4)
20 PRINT D$"OPEN /TEST/NAMES,L140"
30 PRINT D$"WRITE/TEST/NAMES,R936"
40 PRINT "XXX ... XXX": REM 120 X's
50 PRINT D$"CLOSE/TEST/NAMES"
```

This sequence produced eight MLI calls and 29 device driver calls to perform I/O (there were three status calls). The file ended up with six blocks (master index block, two index blocks, and three data blocks) which generated 12 accesses to read and write the bit map.

A 32 megabyte hard disk, the maximum size supported by ProDOS, requires 16 blocks for the free space bit map. Obviously, such a disk would suffer quite a performance impact when allocating new files, or adding space to existing files, if the hard disk were more than half full.

## Ohio Systems Kache Card....................Bob Sander-Cederlof

After reading Ken's article, I came to the conclusion that the Kache Card or something like it is a MUST for users of large hard disks.

The Kache Card has 256K RAM and a controlling Z-80 on it. As far as the Apple is concerned, it is just a hard disk controller. It replaces the controller card which came with your Sider. But it is smarter.

The Kache card remembers the most recently read or most frequently read data blocks. Over 2000 of them. You can see that the entire bit map and at least all the directory blocks associated with the currently used pathnames would stay in RAM on the card. When ProDOS issues a READ command, the DMA interface on the Kache Card simply transfers the block, without doing anything to the hard disk.

When you write to the hard disk, the Kache Card sends it to the hard disk as well as updating its RAM-based copy. You can write to the Kache Card faster than the Kache Card writes to the disk, and your program keeps chugging along while the Kache Card spins out the data to the drive.

The Kache Card is expensive ($695), at least relative to the price of a Sider. A 10-meg Sider is currently $595, and a 20-meg Sider is currently $895. Nevertheless, if you are using 20 megabytes or more you really need a caching system of some kind.

Of course, you could implement caching inside the operating system. ProDOS could be modified (perish the thought) to use about 16K RAM from the //e's auxiliary memory to cache the bit map, root directory, and other frequently used blocks, for each on-line hard disk. (It does not seem profitable to try to cache blocks from floppies, because you can too easily mess things up by removing one floppy and inserting another.)

Like I said, you COULD do it this way. However, it would be very difficult to make it work with the variety of peripherals available to Apple owners. It seems much more reasonable to include caching on the controller card, or even inside the hard disk box itself. I think 256K is probably overkill, 64K per hard drive should be plenty.

My first brush with the Kache Card was not pleasant. I ended up returning the card with a list of complaints. They called me about a month later with the news that they had taken my compaints seriously, and rectified the problems I had pointed out. Or at least most of them.

If you are interested in the Kache Card, contact Ohio Kache Systems Corporation, 75 Tahlequah Trail, Springboro, Ohio 45066. Or call them at (513)746-9160. Tell them where you read this.

=============================================================================
DOCUMENT :AAL-8512:Articles:More.Pzl.Solves.txt
=============================================================================

More Puzzle Solutions...............Bruce Love & Charles Putney

It takes a little longer for the mail to carry our messages overseas, so these solutions missed the November issue.

Bruce Love (from Hamilton, New Zealand) uses the power of the 65802 in a different manner than David Johnson did last month. Remember that David used the MVP instruction to fill all RAM with the STP opcode. Bruce uses a combination of a loop and the PHA instruction to fill all of RAM with $4C, which is a JMP opcode.

If you disassemble a series of $4C's, you will see JMP $4C4C. Therefore Bruce positioned his code so that the last byte to be filled is at $4C4C.

The loop in lines 1160-1200 fills all RAM below $4C4C with the $4C value. After finishing, it jumps back to $4C4C where a two-line loop pushes the A-register on the stack. The trick here is that the stack pointer in the 65802 is 16-bits long. Bruce starts it at $BFFF, and each PHA lowers it by one location. The last location to be changed is $4C4C itself, and after that it loops endlessly executing JMP $4C4C at $4C4C.

Bruce points out that you can test the effectiveness of his program (if you have a 65802 in your Apple) by changing lines 1130 and 1160 to LDX ##$4FFF and LDX ##$4000 respectively. Then it will fill the range from $4000 through $4FFF with $4C, and you can examine it to be sure it did.

Charles Putney (from Shankill, Dublin, Ireland) fills RAM with $48, using the normal 6502 instruction set. Charlie used a combination similar to Bob S-C's solution last month. The final loop resides inside the stack page, and the infinite series of PHA's fills the stack. The difference is that Charlie has the user type an "L" key, which loads the keyboard register with $CC. Then he clears the strobe, which changes it to $4C. Since the locations $C000 through $C002 will all read back as $4C, the cpu will execute JMP $4C4C.

```
=================================================================
DOCUMENT :AAL-8512:Articles:My.Ad.txt
=================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM......................................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code).......................$50
DP18 Source and Object...............................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $32 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers...........................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *
Switch-a-Slot (SCRG)......................................($179.50) $170 *
Extend-a-Slot (SCRG)..........................................($35)  $32 *
Kablit Security System (Secure-It, Inc.)...................($49.95)  $45 *

"Programming the 65816", Eyes..............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
```

```
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
              Foreign customers inquire for postage needed.
        Texas residents please add 6 1/8 % sales tax to all orders.

         *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
         ***                 (214) 324-2050                   ***
         *** We accept Master Card, VISA and American Express ***
```

================================================================================
DOCUMENT :AAL-8512:Articles:PQRS.txt
================================================================================

## S-C Macro Assembler Quick Reference Booklet

We have a new Quick Reference Booklet for the S-C Macro Assembler! With all the new features of the Version 2.0 Macro Assemblers, including the 65C02 and 65816 in both DOS and ProDOS, we have outgrown the old Programmer Reference Card. Taking its place is our new Programmer Reference, a 14-page booklet containing even more information on the S-C Macro Assembler, even more information on the 6502/65C02/65802/65816 processors, and even more information on the Apple II, II+, //e, and //c computers.

All this new reference information is organized into an easy-to-read 14-page booklet, with the S-C Macro Assembler commands at the beginning and the 65XXX opcode tables in the center spread, so it will be as easy as possible to flip right to those important items.

These are the major subject headings covered in the new Programmer Reference:

S-C Macro Assembler Commands
Shorthand Commands
EDIT Mode Commands
DOS Commands Relevant to S-C Macro Assembler
ProDOS Commands Active under S-C Macro Assembler
S-C Macro Assembler Directives
Operand Expressions
6502/65C02 Instructions with Opcode and Execution Cycles
65802/65816 Instructions
Status Registers
Interrupt Vectors
Page Three Locations
Apple Monitor Commands
Apple Monitor Entry Points
S-C Macro Assembler Memory Maps
Source File Formats
S-C Macro Assembler Parameters
Sweet-16 Opcodes
//e and //c Bank Switches
ASCII Chart
Apple II I/O Addresses

As you see, we've packed just about all of the important assembler, processor and computer information you need into this convenient 5 1/2 x 8 1/2 inch package.

The new S-C Macro Assembler Programmer Reference is only $3.00 (plus $1.00 postage for foreign orders).

=============================================================================
DOCUMENT :AAL-8512:Articles:PseudoVariables.txt
=============================================================================

Using Pseudo-Variables in Machine Language...........John Oakey
227 Creekstone Bend, Peachtree City, GA

A couple of years ago I got a bright idea. I was working on an Applesoft program that required knowing what files were on the disk in a given disk drive. By creating binary "images" of Applesoft variables, I was able to hook into DOS 3.3 and employ Applesoft routines to convert the information DOS 3.3 prints to the screen into regular Applesoft variables.

The whole thing worked beautifully and was printed in the last of only four "Second Grade Chats" ever published in Softalk Magazine -- in the very last issue. ("Sorree -- your number has been dis-co-nected.") I never did get paid. ($!#%~&*)

Oh, well. We Apple owners mainly do it for the love of the little machine anyway. Right? Since that time I have realized that the most important thing which I did in that article was to discover the technique of creating pseudo-variables for use in an applications program which can make available all the subroutines already written in the Applesoft ROMs.

It doesn't require a long explanation. Just one example should be enough, and it so happens that one is printed below. This short program, when called from an Applesoft program, will "poll" an Applied Engineering TIMEMASTER H.O. card from 80-column mode without affecting the screen and move the ASCII string which the time card places in the input buffer into the Applesoft variable TIME$. It not only makes getting the time while in 80-column mode possible without blowing away the screen, but it also is a great deal faster than trying to use an Applesoft interface.

This routine should also work with ThunderClock and other compatible clock cards. Permission is granted to reprint this article and to use the copyrighted program below for non-commercial applications. Have a good TIME$!

And, as usual, Bob couldn't resist squeezing out a few bytes:

==============================================================================
DOCUMENT :AAL-8512:Articles:RAMDisk.Bug.txt
==============================================================================

Little RAM Disk Bug

Does that mean a Bug in the Little RAM Disk, or a Little Bug in the RAM Disk?  Actually, both.  Several of you have called or written to point out a problem in Bob's program last month.

The TAY instruction at line 1280 (on page 8) should be a TYA.  It does seem pointless to force Y to zero and then immediately clobber it with whatever the processor read out of $C083.  This code worked when Bob tested it on his //e, because that computer does return a zero when you read $C083.  My ][+, on the other hand, returns a byte of video data, usually $A0, and that really makes a mess out of the VTOC and Catalog sectors.

There's one other glitch in that article as well.  In the fifth paragraph on page 5 there is a reference to line number "189~".  I bet you can guess that's really supposed to be "1890".

So, thanks to all of you who caught us on this one!  It's nice to know you're keeping an eye on us.

```
=======================================================================
DOCUMENT :AAL-8512:DOS3.3:S.DAY.OF.WEEK.txt
=======================================================================

 1000  *SAVE S.DAY OF WEEK
 1001         .OR $300
 1010  *--------------------------------
 1020  YEAR   .BS 1        84-99 MEANS 1984-1999; 0-83 MEANS 2000-2083
 1030  MONTH  .BS 1        1...12 FOR JAN...DEC
 1040  DAY    .BS 1        1...31
 1050  W      .BS 1
 1060  *--------------------------------
 1070  DOW
 1080         LDA YEAR     NORMALIZE YEAR TO 1984
 1090         SEC          SO IT RUNS 1...99
 1100         SBC #84      (MAR 1, 1984 THROUGH DEC 31, 2083)
 1110         BCS .1       WAS 1984-1999
 1120         ADC #100     WAS 2000-2083
 1130  .1     STA W
 1140  *--------------------------------
 1150         LDA MONTH    ADJUST MONTH SO FEBRUARY IS END OF YEAR
 1160         SEC
 1170         SBC #3
 1180         BCS .2
 1190         DEC W
 1200         ADC #12
 1210  .2     TAX
 1220  *--------------------------------
 1230         LDA W        YEAR
 1240         LSR
 1250         LSR
 1260         CLC          + INT (YEAR/4)
 1270         ADC W
 1280         ADC MD,X     + MD(ADJ.MONTH)
 1290         ADC DAY      + DAY
 1300  *--------------------------------
 1310         SEC
 1320  .3     SBC #7       MOD 7
 1330         BCS .3
 1340         ADC #7
 1350         STA W
 1360         RTS
 1370  *--------------------------------
 1380  MD     .DA #3,#6,#1,#4,#6,#2,#5,#0,#3,#5,#1,#4
 1390  *--------------------------------
```

```
================================================================================
DOCUMENT :AAL-8512:DOS3.3:S.RAMFIll.BLove.txt
================================================================================

 1000  *SAVE S.RAMFILL BRUCE LOVE
 1010  *--------------------------------
 1020         .OP 65802
 1030         .OR $4C49
 1040  *--------------------------------
 1050  PAINT  JMP .2
 1060  *--------------------------------
 1070  .1     PHA           PUSH FROM $BFFF DOWN
 1080         JMP .1        (NOTE = 4C4C4C)
 1090  *--------------------------------
 1100  .2     CLC           TURN ON 65802 MODE
 1110         XCE
 1120         REP #$10      X=16 BIT, A=8 BIT
 1130         LDX ##$BFFF   POINT STACK TO TOP OF RAM
 1140         TXS
 1150         LDA #$4C      FILL VALUE
 1160         LDX ##0       POINT TO BOTTOM OF RAM
 1170  .3     STA >0,X      FILL FROM $0000 TO $4C4B
 1180         INX
 1190         CPX ##$4C4C
 1200         BCC .3
 1210         BCS .1        BACK TO FILL FROM TOP DOWN
 1220  *--------------------------------
```

```
=========================================================================
DOCUMENT :AAL-8512:DOS3.3:S.RAMFILLPutney.txt
=========================================================================
```

```
 1000  *SAVE S.RAMFILL PUTNEY
 1010  *--------------------------------
 1020  *  BY  CHARLES H. PUTNEY
 1030  *      18 QUINNS ROAD
 1040  *      SHANKILL
 1050  *      CO. DUBLIN
 1060  *      IRELAND
 1070  *--------------------------------
 1080         .OR $803     NORMAL PLACE
 1090  *--------------------------------
 1100  PNTR   .EQ $06      BLOCK MOVE POINTER
 1110  *--------------------------------
 1120  KEYBD  .EQ $C000    KEYBOARD DATA
 1130  KEYSTB .EQ $C010    KEYBOARD STROBE
 1140  VIDOUT .EQ $FDF0    VIDEO OUTPUT ROUTINE
 1150  CROUT  .EQ $FD8E    SEND A RETURN
 1160  *--------------------------------
 1170  WIPE   JSR CROUT    START A NEW LINE
 1180         LDX #$00
 1190  .1     LDA MESS,X   TELL HIM WHAT KEY TO PUSH
 1200         JSR VIDOUT   SEND IT
 1210         INX          NEXT CHAR
 1220         TAY          CHECK IF LAST ONE
 1230         BMI .1       NO
 1240         JSR CROUT    SEND A RETURN
 1250         LDX #$00
 1260  .2     LDA IMAGE,X  RELOCATE CODE TO PAGE ONE
 1270         STA $200-CODEND+CODE,X
 1280         INX
 1290         CPX #CODEND-CODE
 1300         BNE .2
 1310  .3     BIT KEYBD    KEY PRESSED ?
 1320         BPL .3       WAIT UNTIL PUSHED ?
 1330         LDA KEYSTB   RESET STROBE
 1340         LDA KEYBD    MAKE SURE ITS THE RIGHT KEY
 1350         CMP #$4C     IS IT L ? (JMP OPCODE)
 1360         BNE WIPE     TELL HIM AGAIN
 1370         JMP CODE     WIPE OUT !
 1380  *--------------------------------
 1390  *   THIS CODE IS RELOCATED TO PAGE ONE
 1400  *--------------------------------
 1410  IMAGE  .PH $1E1
 1420  CODE   LDA #$00     INITIALIZE POINTER
 1430         STA PNTR
 1440         LDA #$02
 1450         STA PNTR+1   START AT PAGE TWO
 1460         LDA #$48     GET A PHA OPCODE
 1470         LDY #$00     INIT Y REG
 1480  .1     STA (PNTR),Y SAVE PHA OPCODE
```

```
1490         INY          NEXT
1500         BNE .1       FULL PAGE DONE ?
1510         INC PNTR+1   NEXT PAGE
1520         LDX PNTR+1   CHECK IF DONE
1530         CPX #$C0     AT I/O AREA ?
1540         BNE .1       NOT YET
1550  .2     STA $00,Y    SET PAGE ZERO TO $48
1560         INY          NEXT
1570         BNE .2       FULL PAGE WIPED ?
1580  *   FALL INTO PAGE 2 PHA'S
1590  CODEND .EP
1600  *--------------------------------
1610  MESS   .AT -/TYPE UPPER CASE L TO SET MEMORY TO $48  /
1620  *--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8512:DOS3.3:S.READ.TIME.txt
==================================================================

 1000  *SAVE S.READ.TIME
 1010  *--------------------------------
 1020  *   READ TIMEMASTER H.O. CARD, PUTTING
 1030  *      TIME INTO APPLESOFT STRING TI$.
 1040  *--------------------------------
 1050  *   ORIGINAL BY JOHN OAKEY, 11-22-85
 1060  *              (c) 1985
 1070  *
 1080  *   MODIFIED BY BOB SANDER-CEDERLOF
 1090  *--------------------------------
 1100  FORPNT .EQ $85,86
 1110  TXTPTR .EQ $B8,B9
 1120  *--------------------------------
 1130  WBUF   .EQ $200
 1140  *--------------------------------
 1150  SLOT   .EQ 5      <<<BE SURE TO PUT YOUR SLOT HERE>>>
 1160  *--------------------------------
 1170  AS.GDBUFS  .EQ $D539     MARK END, CLEAR HI-BITS
 1180  AS.SAVD    .EQ $DA9A     FINISH INSTALLING STRING
 1190  AS.PTRGET  .EQ $DFE3     PARSE STRING NAME
 1200  AS.STRLIT  .EQ $E3E7     BUILD STRING DESCRIPTOR
 1210  *--------------------------------
 1220         .OR $300     (WHERE ELSE!)
 1230  *--------------------------------
 1240  RDTIME LDA TXTPTR   SAVE CURRENT TEXT PNTR
 1250         PHA
 1260         LDA TXTPTR+1
 1270         PHA
 1280  *---READ TIME INTO BUFFER--------
 1290         LDA #"%"     MODE: "FRI JAN 03 10:11:32 AM"
 1300         JSR SLOT*256+$C00B
 1310         JSR SLOT*256+$C008  READ TIME STRING
 1320  *---PREPARE STRING FOR A/S-------
 1330         LDX #23      LENGTH OF STRING
 1340         JSR AS.GDBUFS     CLEAR HI-BITS AND MARK END
 1350  *---SETUP TI$ VARIABLE-----------
 1360         LDA #VARNAM
 1370         STA TXTPTR
 1380         LDA /VARNAM
 1390         STA TXTPTR+1
 1400         JSR AS.PTRGET
 1410         STA FORPNT
 1420         STY FORPNT+1
 1430  *---MOVE TIME INTO TI$-----------
 1440         LDA #WBUF+1  SKIP OVER LEADING QUOTE
 1450         LDY /WBUF+1
 1460         JSR AS.STRLIT
 1470         JSR AS.SAVD
 1480  *---RESTORE TXTPTR, RETURN-------
```

```
1490        PLA
1500        STA TXTPTR+1
1510        PLA
1520        STA TXTPTR
1530        RTS
1540 *--------------------------------
1550 VARNAM .AS /TI$/
```

```
===================================================================
DOCUMENT :AAL-8512:DOS3.3:S.READTIMEPLUS.txt
===================================================================

 1000  *SAVE S.READ.TIME+
 1010  *---------------------------------
 1020  *   READ TIMEMASTER H.O. CARD, PUTTING
 1030  *      TIME INTO APPLESOFT STRING TI$.
 1040  *---------------------------------
 1050  *   BY BOB SANDER-CEDERLOF
 1060  *---------------------------------
 1070  VARNAM .EQ $81,82
 1080  FORPNT .EQ $85,86
 1090  *---------------------------------
 1100  WBUF   .EQ $200
 1110  *---------------------------------
 1120  SLOT   .EQ 5      <<<BE SURE TO PUT YOUR SLOT HERE>>>
 1130  *---------------------------------
 1140  AS.GDBUFS  .EQ $D539     MARK END, CLEAR HI-BITS
 1150  AS.SAVD    .EQ $DA9A     FINISH INSTALLING STRING
 1160  AS.PTRGET9 .EQ $E04F     FIND OR MAKE VARIABLE
 1170  AS.STRLIT  .EQ $E3E7     BUILD STRING DESCRIPTOR
 1180  *---------------------------------
 1190         .OR $300     (WHERE ELSE!)
 1200  *---------------------------------
 1210  RDTIME
 1220         LDA #"%"     MODE: "FRI JAN 03 10:11:32 AM"
 1230         JSR SLOT*256+$C00B
 1240         JSR SLOT*256+$C008  READ TIME STRING
 1250  *---PREPARE STRING FOR A/S-------
 1260         LDX #23      LENGTH OF STRING
 1270         JSR AS.GDBUFS     CLEAR HI-BITS AND MARK END
 1280  *---SETUP TI$ VARIABLE-----------
 1290         LDA #'T'     HI-BIT OFF FOR STRING VARIABLE
 1300         STA VARNAM
 1310         LDA #"I"     HI-BIT ON FOR STRING VARIABLE
 1320         STA VARNAM+1
 1330         JSR AS.PTRGET9
 1340         STA FORPNT
 1350         STY FORPNT+1
 1360  *---MOVE TIME INTO TI$-----------
 1370         LDA #WBUF+1  SKIP OVER LEADING QUOTE
 1380         LDY /WBUF+1
 1390         JSR AS.STRLIT
 1400         JMP AS.SAVD  CLEAN UP STRINGS AND RETURN
 1410  *---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8512:DOS3.3:Test.DayWeek.1.txt
==============================================================================

dÜMD(11),D$(6)0nÉ3,6,1,4,6,2,5,0,3,5,1,4YxÉSUN,MON,TUES,WEDNES,THURS,
FRI,SATURnÇÅI–0¡11:áMD(I):ÇÇåÅI–0¡6:áD$(I):Çç»ÑY,M,Dó“M–M…3≠‹≠M—0ƒM–
M»12:Y–Y…1∫ÊY–Y…1984—W–Y»”(YÀ4)»MD(M)»DÂ˙≠Wœ6ƒW –
W…7:´250ı∫D$(W)"DAY"˛´200
```

```
================================================================================
DOCUMENT :AAL-8512:DOS3.3:Test.DayWeek.2.txt
================================================================================
```

```
dÜD$(6)5nÉSUN,MON,TUES,WEDNES,THURS,FRI,SATURIxÅI–0¡6:áD$(I):Çf»Ñ"YEAR
(1984-2083):  ";Y}“≠Y—1984ŒYœ2083ƒ200í‹Y–Y…”(YÀ100) 100ùÊπ768,Y∫,Ñ"
MONTH (1-12):  ";MÃ6≠M—1ŒMœ12ƒ300◊@π769,MÙêÑ"      DAY (1-31):
";D\ ö≠D—1ŒDœ31ƒ400       §π77,D     Ùå772&     ˛W–,(771)6
     X∫D$(W)"DAY"?   b´200
```

==============================================================================
DOCUMENT :AAL-8601:Articles:Browns.Mover.txt
==============================================================================

RAMWORKS Compatible Auxiliary MOVE Routine.....Harvey Brown
Spirit River, Alberta, CANADA

The MOVE routine inside the Apple //c and //e ROM transfers data conveniently to and from the auxiliary 48K area, but it does not work with the upper 16K area. Also, it does not work with the extra banks of RAM available in cards such as the RAMWORKS from Applied Engineering.

I needed that ability, so I wrote my own MOVE subroutine. Mine uses the page at $200 in main RAM as a buffer, to simplify the movement code. If you want to move from an arbitrary bank to another arbitrary bank, my program will require you to use $200 in main RAM as an intermediate buffer. (Somewhat like stopping at Chicago on your way from Toronto to Dallas.) My program also assumes you are always moving exactly 256 bytes (one page). This simplifies the code and the calling sequence, and is probably a reasonable restriction.

The program begins by copying itself into every bank you are using. The bank numbers must be assembled in to the list in lines 1800-1870. Notice that I use bank number $FF to signal the main RAM, and banks from $00 up to signal the banks of Auxiliary RAM. This code needs to reside in the same location in every bank that will be switched on, because when you move from an auxiliary bank to the main RAM that auxiliary bank will be set up so that all RAM reads come from it. This includes reads for the program, so the program had better be there.

Once the program has been initialized, you can JSR MOVE (or JSR $C03 if you want to use a "frozen" entry point) to move a page. At the time of the JSR MOVE, you should have the high byte of the Auxiliary RAM address in the A-register, and the bank number in the X-register. Set carry (SEC opcode) to indicate moving from main $200, or clear carry (CLC opcode) to indicate moving into main $200. Thinking in terms of a ramdisk application, SEC for a write or CLC for a read.

Warning: my program assumes you are calling from a program that runs with the Applesoft ROM selected (see line 1780). If you plan to run it with RAM selected in the upper 16K, you will have to make appropriate changes. You could save the status of the LCRAM and LCBANK soft switches ($C012 and $C011 respectively) before changing them. These partially indicate the status of the $C08x switches. You can tell whether RAM or ROM was selected, and restore the proper one after MOVE is finished. You can also tell which $D000 bank was selected. However, you cannot tell whether the RAM was write-enabled or not; also, you cannot tell if it was in the special mode in which you read ROM and write RAM.

==============================================================================
DOCUMENT :AAL-8601:Articles:Correx.DblInit.txt
==============================================================================

Correction to DOS/ProDOS Double Init.......Bob Sander-Cederlof

The Sep 85 (V5N12) issue of AAL included an article and program to initialize a disk with both DOS and ProDOS catalogs in separate halves of the disk.  After trying to use Catalog Arranger on a disk we made with DOUBLE.INIT, we discovered that program has a bug.

The DOS catalog is written in track $11, starting with sector 15 and going backwards to sector 1.  The second and third bytes in each catalog sector are supposed to point to the next catalog sector, with the exception of those bytes in the LAST catalog sector.  In the last catalog sector, the link bytes should both be $00, to signal to anyone who tries to read the catalog that this is indeed the last sector. DOUBLE.INIT stored $11 in the first link byte, and so some catalog reading programs such as Catalog Arranger get very confused.

The fix is to add the following lines to the program, where the line numbers correspond to those in the printed listing in AAL:

```
       2201      BNE .5
       2202      STY C.TRACK   (Y=0)
```

Add the label ".5" to line 2210, so that it reads:

```
       2210 .5   JSR CALL.RWTS
```

If you have already created some disks with DOUBLE.INIT, we suggest you use a program such as Bag of Tricks, CIA, or some other disk zap program to clear the second byte of track $11, sector $01 on those disks.

```
==============================================================================
DOCUMENT :AAL-8601:Articles:Front.Page.txt
==============================================================================
```

$1.80

Volume 6 -- Issue 4                    January, 1986

In This Issue...



New Low Diskette Price

When we first started offering blank diskettes for sale, back in May of 1981, we were able to sell them for the then below retail price of only $50 for 20 diskettes.  There have been quite a few changes in this industry in the last five years, and prices have continued to fall.  We have a new supplier and can now offer you quality diskettes for only $20 per package of 20, a reduction of 37% since last month.

New Monthly Disks

Since diskette prices have fallen so far, we are now planning to send out disks containing the source code from AAL on a monthly basis, instead of quarterly as we have in the past.  See the note on page 11 for the details.

Discover

Sears Financial Network is launching a new credit card this year, called Discover.  They offer lower interest rates and no yearly charge to the consumer, and better rates to the merchants as well, so we are pleased to be able to accept this card now.  You can use it just like any other card for your phone and mail orders.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8601:Articles:Lawries.Notes.txt
==============================================================================

A Question About BRUN

Mike Lawrie, a reader in South Africa, reports that he tried our prime benchmark (Sep 85 AAL) in a Titan Accelerator card equipped with a 65802. It ran 1000 times in 41 seconds, which correlates very nicely with my predictions in the article. The Titan card runs at 3.58 MHz, and I predicted .35 seconds for 10 repetitions at 4 MHz.

Mike also asked an interesting question, which has been asked by a lot of you at one time or another. Why is it that some assembly language programs can be BLOADed and CALLed, but not BRUN? Even the following very simple program will not return from a BRUN, while it will from a BLOAD followed by a CALL:

```
       JSR $FF3A   Ring the bell
       RTS
```

The problem is inside the DOS BRUN command. This command does not use a JSR command to jump to the binary code just loaded. Rather, it uses a JMP command. No return address is left on the stack. When the RTS at the end of the program is executed, it pops garbage off the stack and returns wherever that garbage indicates. What will happen is rather unpredictable.

The Applesoft CALL command does use JSR, and so it works. So does the monitor G command, and the S-C Macro Assembler MGO command. In ProDOS, the BRUN processor works correctly, using a JSR.

This leaves the question: How should a BRUNnable program end under DOS 3.3? If it is to return to the command prompt (] for Applesoft) then the last line should be JMP $3D0. If the BRUN command came from a machine language program (unlikely) then the called program should end with a JMP to a known entry point in the calling program. The most likely case is an Applesoft program that uses a machine language routine. The best way to handle this is to use BLOAD and CALL.

=============================================================================
DOCUMENT :AAL-8601:Articles:Lores2Hires.txt
=============================================================================

Convert Lo-Res Pictures to Hi-res....Bob Sander-Cederlof

Most Apple dot-matrix printer interfaces now include the firmware to print hi-res graphics pictures directly from a screen image. However, most do not provide any way to print lo-res graphics pictures. With the program presented here you can convert a lo-res graphics image into a hi-res picture, ready to be printed by your interface firmware.

Even if you don't ever plan to do such a thing, there are some neat coding tricks in the following program, which you might be able to apply to other hi-res programs.

Lines 1070-1200 demonstrate the use of my lo- to hi-res conversion program. Notice that I started with the label "T". I find I am using that label all the time, lately. I think I started using it as a short mnemonic for "TEST". It is convenient, because in the S-C Macro Assembler environment I can test the program I just assembled by typing "MGO" and the label I want to start at. I find my fingers can now type "MGOT" without my brain even realizing it happened.

The first thing my demo does is call PLOT to create a lo-res picture. I didn't have any real lo-res art around, so I simply drew a 4-by-4 pattern showing all 16 lo-res colors. PLOT fills 16 (4x4) pixels with color 0, 16 with color 1, and so on:

```
lo-res    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

   0      0  0  0  0  4  4  4  4  8  8  8  8  C  C  C  C
   1      0  0  0  0  4  4  4  4  8  8  8  8  C  C  C  C
   2      0  0  0  0  4  4  4  4  8  8  8  8  C  C  C  C
   3      0  0  0  0  4  4  4  4  8  8  8  8  C  C  C  C
    ----
  15      3  3  3  3  7  7  7  7  B  B  B  B  F  F  F  F
```

The rest of the lo-res screen I did not change, so it will normally show the lo-res equivalent of whatever text was on the screen. Of course if you were really trying to use my CONVERT program you would draw your real lo-res picture.

Lines 1090-1120 turn on the lo-res graphics display, and line 1130 waits until I press a key on the keyboard. After running this much of the program, and studying the dot patterns on the screen, I realized that it is not possible to exactly reproduce the lo-res colors on the hi-res screen (unless I used //e or //c double hi-res). However, by mixing various patterns of dots within the 28 dots (7x4) each lo-res pixel maps to, I can come close to the same color. I don't really know how close I came, because I do not have a color monitor. However, I can at least tell by inspection that all 16 colors map to different dot patterns that will be distinguishable colors.

The PAUSE.FOR.ANY.KEY subroutine will return EQ status if the key I press is RETURN, and NE status if it is any other key. Line 1140 will terminate my test program if RETURN was typed. If it was not RETURN, line 1150 turns on hi-res graphics and line 1160 calls the convert program. Then line 1170 waits for another keypress. Again, RETURN will terminate the test, and any other key will flop back to let me see the lo-res display again. Line 1190 turns the text display back on.

CONVERT is very straightforward. The outer loop, using the X-register, runs from 23 down to 0. This corresponds to the 24 text lines on the screen, or 48 lo-res rows. If your lo-res picture does not use the bottom 4 lines (8 rows), change line 1300 to "LDX #19".

The inner loop, using the Y-register, runs from 39 down to 0, corresponding to the 40 columns of lo-res pixels. Each of the 960 bytes addressed by X and Y contains two lo-res pixels. The top lo-res row (HLIN 0,39 AT 0) is in the low-order nybble of each of 40 bytes starting at $400. The second row (HLIN 0,39 AT 1) is in the high-order nybble of the same 40 bytes. The third and fourth rows are in the 40 bytes starting at $480, and so on. The starting addresses for each row-pair are exactly the same as those for the 24 lines of the text screen. They also happen to be very closely related to the starting addresses for the corresponding rows on the hi-res screen.

I stored the 24 starting addresses in two tables, LOL and LOH. LOL contains the low-half of each address, and LOH the high. Lines 1320-1360 pick up the base address for the current row- pair and put it in pointer LBAS. Lines 1340 and 1370-1380 set up a similar pointer for the hi-res screen. Note that the only difference is that the lo-res screen starts at $400, and the hi-res screen starts at $2000. This address points at the first byte (first seven dots) of the top line of the eight hi- res lines that are in the same position as the lo-res row-pair.

Each lo-res pixel will be mapped to four lines on the hi-res screen, and will be seven dots (or one byte) wide. Each of the 960 lo-res bytes has two pixels, so each byte uses eight lines on the hi-res screen. The right lo-res nybble will be the top four lines, and the left nybble will be the bottom four. After studying the tables of hi-res addresses, I noticed that each set of eight lines follow a very regular pattern. Given the address for the leftmost byte of the top line of a set of eight lines, I can compute the addresses for the next seven lines by successively adding 4 to the high byte of the address. Thus the base addresses for the first eight lines are $2000, $2400, $2800, $2C00, $3000, $3400, $3800, and $3C00. I can always get the base address for the first of the eight by subtracting $400 and adding $2000 to the corresponding lo-res pointer. Line 1370 does that operation in one step with "EOR #$24".

Lines 1400-1480 pick up the current lo-res byte and feed first the right nybble and then the left nybble to PROCESS.NYBBLE. For indexing purposes I multiply the nybble by 8, so that the lo-res color is in the A-register like this: xCCCCxxx. More on that later. Lines 1490-1530 are the south ends of the two nested loops, equivalent to NEXT Y and NEXT X. By the way, please don't get confused by the terms Y and X. They refer in my program to 6502 registers, not Cartesian coordinates. Just to keep your minds nimble, I use the Y-register for the X-coordinate. The X-register is half the lo-res Y-coordinate.

I mentioned above the problem of coming up with patterns of 28 dots to approximate the lo-res colors. There are only six solid hi-res colors, which correspond exactly to lo-res colors 0, 3, 6, 9, 12, and 15. The other 10 lo-res colors take double the normal hi-res resolution to reproduce exactly. However, as Don Lancaster explains in detail in his "Enhancing Your Apple II -- Volume I", you can produce thousands of shades in hi-res by using dot patterns. I picked 12 of his patterns based on the names he gave them, since I did not have a color monitor. His patterns fit in a 28-dot by two line array. Since each byte stores seven dots, it takes 28 dots before the some of the patterns repeat. Using two lines with different or offset patterns gives even more variety.

The table SHADES in lines 1900-2050 give sixteen patterns. The first four bytes of each color are for the first line of 28 dots, and the other four bytes give the second line of 28 dots. Each lo-res pixel will use only one pair of bytes from the set of eight, depending on which column it is in. The last two bits of the lo-res column number (in the Y-register) select which byte pair to use.

Lines 1650-1700 build an index to the byte pair by merging those two bits with the color*8. Then by addressing "SHADE,X" I get the first byte of a pair, and by using "SHADE+4,X" I get the second one. Each lo-res pixel will use the four hi-res bytes by repeating the pair selected from SHADES.

The rest of the code in PROCESS.NYBBLE involves putting the selected color bytes into the hi-res area. HBAS points at the top line of the four to be stored into, and the Y-register points to the byte on that line; so "STA (HBAS),Y" will store into that byte. COMMON.CODE (so named because of a lack of creativity on my part this morning when I discovered that the same eight lines appeared twice) gets and puts two color bytes. The first byte goes into (HBAS),Y; then I add 4 to the high byte of HBAS (since I KNOW it is zero, ORA can be used to add the bit) and store the second byte at the new (HBAS),Y. The "EOR #$0C" at line 1720 changes $24 to $28 or $34 to $38. Similarly, the "EOR #$1C" at line 1750 changes $2C to $30 or $3C to $20. This last possibility leaves HBAS prepared for the next column, automatically!

Some of the same tricks could be used in writing a program to copy text from the text screen to the hi-res graphics screen, or for a general purpose routine to write characters onto the hi-res screen. Instead of using a color map, we would need a table of dot-matrix characters. Maybe this is just how everyone does it, but I don't recall seeing all of these tricks in any previous code. Especially the idea of getting the hi-res base pointer by merely toggling two bits in the equivalent text pointer, and the idea of generating successive hi-res pointers by merely adding 4 to that base pointer.

When I wrote this program I wasn't really worrying about speed or space. Nevertheless, as you can see, it is fairly compact. As for speed, it takes less than a second.

==============================================================================
DOCUMENT :AAL-8601:Articles:Monthly.Disks.txt
==============================================================================

Monthly AAL Source Disk Subscriptions Now Available

We have always made the source code of all the programs published in Apple Assembly Line available on disks. We have collected the programs from three issues together on Quarterly Disks, priced at $15 each or $45/year.

Now that diskettes are so much less expensive, we have decided to try another approach. For those who are interested in getting the source code on disk, we would like to send the source disk along with each newsletter. We will still collect three issues onto Quarterly Disks, for late comers. But those of you who have Quarterly Disk subscriptions will start getting the Monthly Disks. We will send the disk and newsletter in the same envelope, First Class Mail.

The price for combined newsletter/disk subscriptions will be $64 in the USA, Canada, and Mexico. For other countries the postage is higher, so the fee will be $87.

If you want to synchronize your newsletter and Monthly Disk subscriptions, you can pro-rate the Monthly Disk at $3.75 per month ($4.75 for overseas). You can check the length remaining on your current newsletter subscription by looking at the mailing label: the number in the upper-right corner is the year and month of the last issue of your current subscription.

If you currently are receiving the Quarterly Disk by an automatic charge to your credit card each quarter there will be no change: you will still get the Quarterly Disk rather than the monthly disk.

==============================================================================
DOCUMENT :AAL-8601:Articles:Multiplying.txt
==============================================================================

Fast 6502 & 65802 Multiply Routines........Bob Sander-Cederlof

Since multiplication is not a built-in function in the 6502, 65C02, or 65802, many of us have written our own subroutines for the purpose. I will present some efficient subroutines here, to handle the 8-bit and 16-bit cases.

I will assume both arguments are the same length (either 8-bits or 16-bits) and that we want the full product. If the arguments are only 8-bits long, the product will by 16-bits long. If the arguments are 16-bits long, the product will be 32-bits long. I will also assume the arguments are unsigned values. Thus $FF times $FF will be $FE01 (in decimal, 255x255 = 65025).

Way back in February, 1981, I published an article with a Brooke Boering's fast 16-bit multiplication subroutine. His subroutine duplicated the functions of the subroutine in the original Apple Monitor ROM, but was nearly twice as fast. Brooke's programs were originally published in the December, 1980, Micro magazine (now defunct). He included an 8-bit multiply subroutine with an average time of only 192 cycles.

Damon Slye wrote an article for Call APPLE, published June, 1983. He introduced some coding tricks which allow an 8-bit multiply in an average of 160 cycles. I have reproduced Damon's program below, in lines 1010-1300. His trick involves eliminating a CLC opcode from the loop in lines 1210-1260. Ordinarily you would need a CLC before the ADC instruction; Damon decremented the multiplicand by one before starting the loop, so that adding with carry set works. He does the decrementing in lines 1130-1160. Note that if the original multiplicand was zero, he skips all the rest of the code and just returns the answer: 0.

I had to go at least one step faster, so I partially "un-wrapped" the 8-step loop. I changed it to loop only four times, but handled two bits of the multiplier each time. This runs an average time of 140 cycles. You could unwrap it all the way, writing out the BCC-ADC-ROR-ROR lines a total of 8 times, and cut the average time down to only 111 cycles.

Let me stop here and say what I mean by average time. I am stating time in terms of "cycles", rather than seconds or microseconds. The Apple two different cycle times, depending on the video timing logic. The average Apple speed is 1020488 cycles per second. The multiply algorithms will vary in speed depending on the number of bits in the multiplier which equal "1". If the multiplier = $FF (all ones) the algorithm will take the maximum time. If the multiplier is $00, it will take the minimum time. On the average for random arguments, the multiplier will have four zeroes and four ones, so the average time is equal to the average of the minimum and maximum times. For all of the subroutines, I included the cycles for a JSR to call them, and for the RTS at the end.

I programmed an 8-bit multiply using 65802 opcodes, as shown below in lines 1560-1790. The program is slightly shorter (one byte), but that really isn't a fair comparison. The arguments and product are handled differently, and so the effort to call the program may be more or less than that for the 6502 version. Rather than passing the multiplicand in the X-register, I have it in the A-register. I pass the multiplier in the high byte of the A-register. Since X is not used for passing any values, I saved and restored it (lines 1620 and 1770). I assumed the program would be called from the 6502 mode, which of course it was as long as I was testing it. In "real life" it might be written to be called from Native 65802 mode, since the larger program it was a part of would also be taking advantage of all the 65802 features.

I used a couple of tricks to save space and time. One you may justly complain about is that I store the multiplicand directly into the operand field of the ADC instruction at line 1720. This definitely saves time, but it also could have serious drawbacks. (For example, it would not work if executed from ROM.) Since I enter in 6502 Emulation mode, line 1640 only loads 0 into the low byte of the A-register. Lines 1650-1660 enter the 65802 Native mode. Line 1680 sets the A-register to 16-bit mode.

In line 1690 I form the inverse (one's complement) of the multiplier. This is just another way of eliminating the CLC from the loop. Note that the multiplier is in the high byte of A, and the product is going to be accumulated in the low byte. The loop runs from line 1700 through line 1740. Line 1700 shifts to the left both the partial product and what remains of the multiplier, putting the highest remaining bit of the multiplier into the carry status bit. If that bit = 1, then the original bit in the multiplier before complementing was a zero, so we do not add the multiplicand to the current partial product. As we continue through the loop, the bits of the multiplier keep shifting out just ahead of the ever-growing partial product, until finally we have the answer.

Lines 1750-1780 restore the machine state to the 6502 Emulation mode and restore the original X-register value. The full product is now in the A-register. If I wanted to print out the product, I might do it like this:

```
       XBA             GET HIGH BYTE INTO LOW-A
       JSR $FDDA       MONITOR PRINT-BYTE SUBROUTINE
       XBA             GET LOW BYTE INTO LOW-A
       JMP $FDDA
```

Here is a summary of the execution times (in cycles) for the three 8-bit multiply subroutines:

| | Minimum | Maximum | Average |
|---|---|---|---|
| Slye | 152 | 168 | 160 |

```
        RBSC    132       148      140
        65802   119       135      127
```

The 65802 version would be seven cycles faster if we did not require saving and restoring the X-register. If you want to change the 65802 version for calling from Native mode, delete lines 1650, 1660, 1750, and 1760. Then insert the following:

```
       1612         PHP
       1614         SEP #$30
       ...
       1772         PLP
```

These changes add one cycle to the time.

<<<8x8 listings here>>>

I will also show three sample 16-bit multiply subroutines....no, four. The first one is a copy of Brooke Boering's code. The second is a direct conversion of Brooke's code to 65802 code, with emphasis on space. The third modifies the second with the tricks of Damon Slye; it takes more space, but it is faster.

The first three of these subroutines are modeled after the code in the original Apple monitor ROM. The arguments are expected in page zero locations, low-byte first. The result will also be in page zero locations. The function performed is actally a little more than just multiplication, because it is possible to specify an addend as well. The final result will be PRODUCT = ADDEND + (MULTIPLIER * MULTIPLICAND). PRODUCT is stored in four consecutive bytes, backwards. The highest byte is at PRODUCT+1, the next at PRODUCT, the next at PLIER+1, and the lowest at PLIER. The fourth subroutine differs in that the product does not overlap the multiplier.

Looking at Brooke's version (lines 1000-1270) you can see that the loop contains a 16-bit addition (lines 1130-1190). There are also two 16-bit ROR shifts, at lines 1200-1230. These are the likely candidates for shortening via 65802 code. My first version for the 65802 made no other changes in the loop. I merely prefixed Brooke's code with CLC-XCE-REP to get into the 16-bit Native mode, and suffixed it with SEC-XCE to get back to Emulation mode. Then I noticed another shortcut, and the result is in lines 1300-1480.

By moving the LDA PRODUCT up before the BCC opcode in lines 1370-1380, I was able to change a ROR PRODUCT to a simple ROR on the A-register followed by a STA PRODUCT. This saves a net six cycles when the multiplier bit is "1", and costs two cycles when the multiplier bit is "0". The average savings for random multipliers is four cycles, inside a loop that runs 16 times.

The faster version, in lines 1500-1780, merely implements Damon Slye's trick of pre-decrementing the multiplicand so as to avoid an explicit CLC opcode inside the 16-time loop. It costs 12 cycles for the extra setup, but it saves two cycles for each one-bit in the multiplier.

The fourth version, in the separate listing as lines 1000-1430, uses the trick of splitting the multiplier in half. In effect, two parallel 8-bit by 16-bit multiplies are accomplished, with the result usually taking less time than any of the other algorithms. By deleting line 1130 (which shaves off another four cycles) the feature of allowing an addend can be included.

Here is a summary of the execution cycles for the four 16-bit multiply subroutines:

| | Minimum | Maximum | Average |
|---|---|---|---|
| Boering | 541 | 845 | 693 |
| Smaller | 519 | 599 | 559 |
| Faster | 531 | 579 | 555 |
| Fourth | 332 | 684 | 508 (usually fastest) |

Note that the third subroutine also goes even faster when the multiplicand = zero, because the bulk of the code is skipped.

These are pretty good subroutines, but I have no doubt that they can be improved upon. Why not try your hand? If you can significantly improve either space or time or features, send your code to AAL. We'll publish the best ones, and help advance the state of the art. And if you have some classy division subroutines, they are welcome too!

<<<listings of 16x16 routines>>>>

```
==================================================================
DOCUMENT :AAL-8601:Articles:My.Ad.txt
==================================================================

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.......................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM......................................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code).......................$50
DP18 Source and Object...............................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $20 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers...........................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)  $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *
Switch-a-Slot (SCRG)......................................($179.50) $170 *
Extend-a-Slot (SCRG)..........................................($35)  $32 *

"Programming the 65816", Eyes..............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little.............................($19.95)  $18 *
"Inside the Apple //e", Little.............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Beneath Apple DOS", Worth & Lechner.......................($19.95)  $18 *
```

```
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($12.95)  $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)  $16 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
              Foreign customers inquire for postage needed.
        Texas residents please add 6 1/8 % sales tax to all orders.
```

==============================================================================
DOCUMENT :AAL-8601:Articles:Parker.Trivia.txt
==============================================================================

An Interesting Bit of Trivia........................Bill Parker

Some time ago I asked Bob S-C if he knew the origin of the term "6502". Why was this particular number chosen? Bob didn't know, but referred me to Bill Mensch at Western Design Center.

Bill worked at Motorola and was on the design team that created the 6800, which later led to the development of the 68000. He left Motorola with a few others and formed MOS Technology (now absorbed into Commodore), where they developed a new micro- processor which was supposed to be an improved version of the 6800. Hence the decision was made to use a number in the 6000 series. As for the hundreds digit, Commodore already had chips that used just about every digit, except "5". Thus, the "6500" series of chips was born.

As history tells us, the first chip in the series, the 6501, was too close to Motorola's design, and had to be revised. The result was the 6502.

==============================================================================
DOCUMENT :AAL-8601:Articles:Potts.TxtCopy.txt
==============================================================================

Text File Transfer Using DOS 3.3 File Manager.....Bob Potts

Transferring text files from one drive to another can be frustrating and time consuming. The standard procedure is to read from the file on the source drive and write to the file on the target drive. One possible solution is to use FID, but you must BRUN FID and cannot use it from inside an Applesoft program.

With this in mind, I set out to write a utility which will transfer a text file using the DOS 3.3 File Manager (FM) routines. FM has been a part of every release of DOS, but very little documentation has been written about these powerfull routines. While RWTS concerns itself with tracks and sectors, FM deals with whole files, be they binary, text, or Applesoft. I recalled that a couple of years ago, Bob Sander-Cederlof had assisted me with a communications program and had used the FM routines to read and store the file. I located the listing we used, analyzed the code, and here is the result.

The entire program could have been written in assembler, but since most of my programs are in Applesoft (with machine language support routines), I decided to write it as simple as possible. The name of the file to be transferred, the OPEN, READ, WRITE, and CLOSE commands are all obtained through a short Applesoft front end program. The machine language portion is broken down as follows.

Lines 1130-1150 are simply easily accessible jump vectors to the two routines which will be CALLed from inside an Applesoft driver.

Lines 1190-1320 clear the buffer, in this case $2000-95FF, to zeroes. This gives us a buffer of 30,208 bytes, which should be large enough for most text files. (This is 118 sectors.) Lines 1330-1340 reset the base buffer address, for use later to find the end of the data in the buffer.

Lines 1360-1460 load the file that has been OPENed by the Applesoft driver. The process of setting up a FM parameter block is simplified by using a preset data area called RD.BLK, lines 1790-1800. Calling FM.SETUP sets up the Y- and A-registers properly, and then calling FM.ENTRY reads the text file.

Lines 1500-1580 search through the data buffer for the first occurrence of a 00 byte, which will signal the end of data. By subtracting the base buffer address in lines 1660-1710 we get the actual length of the data. Lines 1600-1650 copy in the initial parameter values for writing, and lines 1660-1710 set up the length.

Lines 1720-1740 call on FM to actually write the data on the file that has been opened in the Applesoft driver.

The time saving using this transfer is significant. A text file containing 8000 bytes took 49 seconds to read and write using pure Applesoft. Using the FM the same operation was accomplished in only 17 seconds.

Since the program is only 120 bytes long, it can be placed almost anywhere there is free space, especially on page 3. If you are working from a larger Applesoft program, the starting point for the buffer could be moved as needed to load your text file.

==================================================================
DOCUMENT :AAL-8601:DOS3.3:BrownMoveProg.txt
==================================================================

```
 1000  *SAVE BROWN'S MOVE PROGRAM
 1010  *--------------------------------
 1020  * MOVE by H. Brown
 1030  * Jan 18/86
 1040  *--------------------------------
 1050  PTR    .EQ $00,01
 1060  BUFFER .EQ $200
 1070  RAMRD  .EQ $C002
 1080  RAMWRT .EQ $C004
 1090  ALTZP  .EQ $C008
 1100  BNKSEL .EQ $C073    RAMWORKS BANK SELECT REGISTER
 1110  ROM    .EQ $C082
 1120  RAM1   .EQ $C08B
 1130  RAM2   .EQ $C083
 1140  *--------------------------------
 1150         .OR $C00     ORG  AT BEGINNING OF A PAGE
 1160  *--------------------------------
 1170  COMMONPG
 1180         JMP INIT     BRUN OR JSR TO INITIALIZE
 1190         JMP MOVE     NORMAL ENTRY
 1200  *--------------------------------
 1210  *   INIT copies COMMONPG to all 64K banks
 1220  *--------------------------------
 1230  INIT   LDX BANKS    START WITH LAST 64K BANK
 1240  .1     LDA BANKS,X  GET  BANK #
 1250         STA BNKSEL   SELECT 64K BANK
 1260         STA RAMWRT+1 CHOOSE TO WRITE
 1270         LDY #0       COPY PAGE
 1280  .2     LDA COMMONPG,Y
 1290         STA COMMONPG,Y
 1300         INY
 1310         BNE .2       LOOP TO END OF PAGE
 1320         DEX
 1330         BNE .1       LOOP TO START OF TABLE
 1340         BEQ EXIT     RESTORE STANDARD MEMORY
 1350  *--------------------------------
 1360  *   enter MOVE with A = page (CX for 2nd DX)
 1370  *                   X = 64K bank #
 1380  *                   Carry SET for write, CLEAR for read
 1390  *--------------------------------
 1400  MOVE   BCS .3       BRANCH IF WRITING
 1410         CMP #$C0
 1420         BCS .1       BRANCH IF UPPER 16K
 1430         CPX #$FF     ---  READ 48K ---
 1440         BEQ .2       SKIP IF MAIN 64K
 1450         STA RAMRD+1  READ FROM AUX 48K
 1460         STX BNKSEL   SELECT 64K BANK
 1470         BNE .2
 1480  .1     JSR SEL16K   ---  READ 16K ---
```

```
1490        CPX #$FF
1500        BEQ .2       SKIP IF MAIN 64K
1510        STX BNKSEL   SELECT 64K BANK
1520        STA ALTZP+1  SELECT AUX 16K
1530 .2     CLC
1540        JSR COPYPAGE
1550        BEQ EXIT
1560 *--------------------------------
1570 *   WRITING
1580 *--------------------------------
1590 .3     CMP #$C0
1600        BCS .4       BRANCH IF UPPER 16K
1610        CPX #$FF     ---  WRITE 48K ---
1620        BEQ .5       SKIP IF MAIN 64K
1630        STX BNKSEL
1640        STA RAMWRT+1 WRITING TO AUX 48K
1650        BNE .5
1660 .4     JSR SEL16K   ---  WRITE 16K ---
1670        CPX #$FF
1680        BEQ .5
1690        STX BNKSEL
1700        STA ALTZP+1
1710 .5     SEC
1720        JSR COPYPAGE
1730 *--------------------------------
1740 EXIT   STY BNKSEL   RESORE STD 64K FOR VIDEO
1750        STA RAMWRT   MAIN 48K
1760        STA RAMRD
1770        STA ALTZP    MAIN 16K
1780        LDA ROM
1790        RTS
1800 *--------------------------------
1810 *   BANKS is a table of 64K bank #'s, where
1820 *      FF = main 64k, 00 = alt 64K when no RAMWORKS
1830 *      00,04,08,0C = banks of a 256K RAMworks
1840 *      1st entry is # of banks
1850 *--------------------------------
1860 BANKS  .HS 05       Five banks all told
1870        .HS FF.00.04.08.0C
1880 *--------------------------------
1890 *   COPYPAGE copies 256 bytes
1900 *      from (PTR) in specified bank to motherboard $200
1910 *   or from motherboard $200 to (PTR) in specified bank
1920 *--------------------------------
1930 COPYPAGE
1940        STA PTR+1
1950        LDY #0
1960        STY PTR
1970        BCS .2
1980 .1     LDA (PTR),Y
1990        STA BUFFER,Y
2000        INY
2010        BNE .1
2020        RTS
```

```
.2     LDA BUFFER,Y
       STA (PTR),Y
       INY
       BNE .2
       RTS
*--------------------------------
*   SEL16K selects the appropriate bank in 16K area
*--------------------------------
SEL16K CMP #$D0
       BCS .1
       LDY RAM2     C0   -> AUX D0
       LDY RAM2
       ADC #$10
       RTS
.1     LDY RAM1     SELECT RD/WRT RAM
       LDY RAM1
       RTS
*--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8601:DOS3.3:POTTS.A:S.TRANSFER.txt
==============================================================================

'
≤ PROGRAM TO TRANSFER A TEXT FILEK≤ FROM DRIVE 1 TO DRIVE 2 USINGo-≤
DOS 3.3 FILE MANAGER ROUTINESù(≤---BY BOB POTTS, LOUISVILLE, KENTUCKY-
--∂c≤-------------------/d£8192:≤ $2000-95FF IS MY BUFFERÊnD$ –
Á(4)˚x∫D$"NOMON I,O,C"    Ç∫D$"BLOAD TEXT.TRANSFER.OBJ"5 «≤-----------
-------[  »RF–768:≤CALL ADDRESS TO READ FILEÇ “WF–771:≤CALL ADDRESS TO
WRITE FILEÆ    ‹RC–226:≤PEEK ADDRESS FOR FM RETURN CODE      ,≤------
---------------Ë    6â:ó:∫"TEXT FILE TRANSFER"
@∫"------------------"
JÑ"ENTER FILE NAME:  ";F$;
ê≤READ FILE FROM DRIVE 1O
ö∫D$"OPEN"F$",D1_
§∫D$"READ"F$g
ÆåRFx
∏≠,(RC)–5ƒ500á
¬∫D$"CLOSE"¨
Ã∫"RETURN CODE NOT 'END OF DATA'"≤
÷≥Õ
Ù≤WRITE FILE TO DRIVE 2‹
˛∫D$"CLOSE"
∫D$"OPEN"F$",D2
∫D$"WRITE"F$
åWF
&≠,(RC)–0ƒ600)
0∫D$"CLOSE"F
:∫"RETURN CODE WAS ",(RC)L
D≥Z
X≤FINISHEDi
b∫D$"CLOSE"Ç
l∫"TRANSFER COMPLETE"à
vÄ
```

```
=============================================================================
DOCUMENT :AAL-8601:DOS3.3:PottsTextCopier.txt
=============================================================================

 1000  *SAVE POTTS TEXT COPIER
 1010  *---------------------------------
 1020         .OR $300
 1025         .TF TEXT.TRANSFER.OBJ
 1030  *---------------------------------
 1040  MY.BUFFER  .EQ $2000
 1050  *---------------------------------
 1060  BUFFER     .EQ $E0,E1     POINT TO FILE BUFFER
 1070  RESULT     .EQ $E2        FILE MANAGER RETURN CODE
 1080  *---------------------------------
 1090  FM.SETUP   .EQ $3DC       INITIALIZE Y & A
 1100  FM.ENTRY   .EQ $3D6       FILE MANAGER ENTRY POINT
 1110  FM.BLK     .EQ $B5BB      FILE MANAGER PARM LIST
 1120  *---------------------------------
 1130  *   SET UP JUMP VECTORS
 1140         JMP INITIALIZE.AND.READ
 1150         JMP FIND.END.AND.WRITE
 1160  *---------------------------------
 1170  INITIALIZE.AND.READ
 1180  *---------------------------------
 1190  INITIALIZE.THE.BUFFER
 1200         LDA #MY.BUFFER
 1210         STA BUFFER         LSB
 1220         LDA /MY.BUFFER
 1230         STA BUFFER+1       MSB
 1240         LDY #0
 1250  .1     LDA #0             CLEAR BUFFER UP TO $95FF
 1260  .2     STA (BUFFER),Y
 1270         INY                NEXT BYTE IN THIS PAGE
 1280         BNE .2             ...STILL IN THE PAGE
 1290         INC BUFFER+1       NEXT PAGE
 1300         LDA BUFFER+1
 1310         CMP #$96           AT END OF STORAGE?
 1320         BNE .1             ...NO, KEEP CLEARING
 1330         LDA /MY.BUFFER     RESET BUFFER POINTER
 1340         STA BUFFER+1
 1350  *---------------------------------
 1360  READ.THE.FILE
 1370         LDX #9             10 BYTES
 1380  .1     LDA RD.BLK,X
 1390         STA FM.BLK,X
 1400         DEX
 1410         BPL .1
 1420         JSR FM.SETUP
 1430         JSR FM.ENTRY
 1440         LDA FM.BLK+10      GET RETURN CODE
 1450         STA RESULT         SAVE FOR APPLESOFT PEEK
 1460         RTS                RETURN TO APPLESOFT
 1470  *---------------------------------
```

```
FIND.END.AND.WRITE
*--------------------------------
FIND.END.OF.BUFFER
       LDY #0             SEARCH FOR 00 BYTE
.1     LDA (BUFFER),Y
       BEQ .2             ...FOUND END
       INY
       BNE .1             ...NEXT BYTE IN SAME PAGE
       INC BUFFER+1       NEXT PAGE
       BNE .1             ...ALWAYS
.2     STY BUFFER         LSB OF EOF BYTE
*--------------------------------
WRITE.FILE
       LDX #9             10 BYTES
.1     LDA WR.BLK,X
       STA FM.BLK,X
       DEX
       BPL .1
       LDA BUFFER         LSB
       STA FM.BLK+6       LSB OF FILE LENGTH
       SEC
       LDA BUFFER+1
       SBC /MY.BUFFER
       STA FM.BLK+7       MSB OF FILE LENGTH
       JSR FM.SETUP
       LDX #1             IF NO FILE, ALLOCATE ONE
       JSR FM.ENTRY       WRITE THE FILE
       LDA FM.BLK+10      RETURN CODE
       STA RESULT         SAVE FOR APPLESOFT PEEK
       RTS                RETURN TO APPLESOFT
*--------------------------------
RD.BLK .HS 03.02.0000.0000
       .DA $9600-MY.BUFFER,MY.BUFFER
WR.BLK .HS 04.02.0000.0000
       .DA $9600-MY.BUFFER,MY.BUFFER
*--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8601:DOS3.3:S.Lores2Hires.txt
==================================================================

 1000  *SAVE S.LORES TO HIRES
 1010  *--------------------------------
 1020  LBAS   .EQ $26,27
 1030  HBAS   .EQ $2A,2B
 1040  SAVEX  .EQ $2E
 1050  COLOR  .EQ $30
 1060  *--------------------------------
 1070  T
 1080         JSR PLOT
 1090         LDA $C050    GRAPHICS
 1100         LDA $C052    SOLID (40 BY 48 PIXELS)
 1110         LDA $C054    PRIMARY PAGE
 1120  .1     LDA $C056    LO-RES
 1130         JSR PAUSE.FOR.ANY.KEY
 1140         BEQ .2       ...<RETURN>
 1150         LDA $C057    HIRES
 1160         JSR CONVERT
 1170         JSR PAUSE.FOR.ANY.KEY
 1180         BNE .1       ...NOT <RETURN>
 1190  .2     LDA $C051    TEXT
 1200         RTS
 1210  *--------------------------------
 1220  PAUSE.FOR.ANY.KEY
 1230  .1     LDA $C000    WAIT FOR ANY KEY
 1240         BPL .1       ...NOT YET
 1250         STA $C010    CLEAR STROBE
 1260         CMP #$8D     SET .EQ. IF <RETURN>
 1270         RTS
 1280  *--------------------------------
 1290  CONVERT
 1300         LDX #23      OR #19 IF MIXED MODE
 1310  .1     LDY #39      COLUMNS 0...39
 1320         LDA LOL,X    SET UP BASE POINTER FOR LINE
 1330         STA LBAS
 1340         STA HBAS     SAME FOR HI-RES
 1350         LDA LOH,X
 1360         STA LBAS+1
 1370         EOR #$24     SHIFT FROM $400 TO $2000 FOR HI-RES
 1380         STA HBAS+1
 1390         STX SAVEX    SAVE X-REG
 1400  .2     LDA (LBAS),Y GET TWO LO-RES PIXELS
 1410         PHA          SAVE FOR LOWER ONE
 1420         ASL          UPPER PIXEL * 8
 1430         ASL
 1440         ASL
 1450         JSR PROCESS.NYBBLE
 1460         PLA          GET LOWER PIXEL
 1470         LSR          TIMES 8
 1480         JSR PROCESS.NYBBLE
```

```
1490        DEY          NEXT COLUMN, SCANNING RIGHT TO LEFT
1500        BPL .2       ...ANOTHER ONE
1510        LDX SAVEX    RESTORE X-REG
1520        DEX          NEXT LINE, SCANNING BOTTOM TO TOP
1530        BPL .1       ...ANOTHER ONE
1540        RTS          FINISHED!
1550  *--------------------------------
1560  LOH    .HS 04.04.05.05.06.06.07.07  HIGH BYTES
1570         .HS 04.04.05.05.06.06.07.07  OF SCRN PNTRS
1580         .HS 04.04.05.05.06.06.07.07  (TEXT OR LO-RES)
1590  *--------------------------------
1600  LOL    .HS 00.80.00.80.00.80.00.80  LOW BYTES
1610         .HS 28.A8.28.A8.28.A8.28.A8  OF SCRN PNTRS
1620         .HS 50.D0.50.D0.50.D0.50.D0
1630  *--------------------------------
1640  PROCESS.NYBBLE
1650         AND #$78     MASK THE SHIFTED NYBBLE
1660         STA COLOR
1670         TYA          LO-RES COLUMN
1680         AND #3       LOW 2 BITS
1690         ORA COLOR         0CCCC0YY
1700         TAX
1710         JSR COMMON.CODE
1720         EOR #$0C          3RD LINE OF 4
1730         STA HBAS+1
1740         JSR COMMON.CODE
1750         EOR #$1C          NEXT LINE
1760         STA HBAS+1
1770         RTS
1780  *--------------------------------
1790  COMMON.CODE
1800         LDA SHADES,X      EVEN LINE
1810         STA (HBAS),Y
1820         LDA HBAS+1
1830         ORA #4
1840         STA HBAS+1
1850         LDA SHADES+4,X    ODD LINE
1860         STA (HBAS),Y
1870         LDA HBAS+1
1880         RTS
1890  *--------------------------------
1900  SHADES .HS 00.00.00.00.00.00.00.00  0--BLACK
1910         .HS AA.D5.AA.D5.55.2A.55.2A  1--MAGENTA
1920         .HS 91.A2.C4.88.C4.88.91.A2  2--DARK BLUE
1930         .HS 11.22.44.08.44.08.11.22  3--PURPLE
1940         .HS 2A.55.2A.55.2A.55.2A.55  4--DARK GREEN
1950         .HS 33.66.4C.19.4C.19.33.66  5--GRAY 1
1960         .HS D5.AA.D5.AA.D5.AA.D5.AA  6--MEDIUM BLUE
1970         .HS DD.BB.F7.EE.F7.EE.DD.BB  7--LIGHT BLUE
1980         .HS A2.C4.88.91.88.91.A2.C4  8--BROWN
1990         .HS AA.D5.AA.D5.AA.D5.AA.D5  9--ORANGE
2000         .HS B3.E6.CC.99.CC.99.B3.E6  A--GRAY 2
2010         .HS D5.AA.D5.AA.AA.D5.AA.D5  B--PINK
2020         .HS 6E.5D.3B.77.3B.77.6E.5D  C--LIGHT GREEN
```

```
              .HS 2A.55.2A.55.AA.D5.AA.D5  D--YELLOW
              .HS 2A.55.2A.55.D5.AA.D5.AA  E--AQUAMARINE
              .HS 7F.7F.7F.7F.7F.7F.7F.7F  F--WHITE
*--------------------------------
*   FILL CORNER WITH SAMPLES OF EACH COLOR
*--------------------------------
PLOT   LDY #0
       STY COLOR
.1     LDX #3
.2     LDA COLOR    00, 44, 88, CC
       STA $400,Y   GR ROWS 0-3
       STA $480,Y
       CLC
       ADC #$11     11, 55, 99, DD
       STA $500,Y   GR ROWS 4-7
       STA $580,Y
       ADC #$11     22, 66, AA, EE
       STA $600,Y   GR ROWS 8-11
       STA $680,Y
       ADC #$11     33, 77, BB, FF
       STA $700,Y   GR ROWS 12-15
       STA $780,Y
       INY
       DEX
       BPL .2
       ADC #$11     .., 44, 88, CC, END
       STA COLOR
       BCC .1       ...MORE
       RTS
*--------------------------------
```

=================================================================
DOCUMENT :AAL-8601:DOS3.3:S.M1616.802.EF.txt
=================================================================

```
 1000  *SAVE S.MUL.16X16.65802.EVEN.FASTER
 1010  *---------------------------------
 1020         .OP 65802
 1030  *---------------------------------
 1040  A      .EQ 0,1
 1050  B      .EQ 2,3
 1060  P      .EQ 4,5,6,7
 1070  *---------------------------------
 1080  MUL.EVEN.FASTER
 1090         CLC
 1100         XCE          ENTER NATIVE MODE
 1110         REP #$20     16-BIT A-REGISTER
 1120         STZ P+2      MAKE SURE NO ADDEND IN HI-16
 1130         STZ P   (DELETE IF WANT AN ADDEND IN LO-16)
 1140         LDX #8
 1150         BRA .2       ...HOP OVER SHIFTS
 1160  *---------------------------------
 1170  .1     ASL P        DOUBLE THE PRODUCT
 1180         ROL P+2
 1190  .2     LDA A
 1200         AND ##$0080  LOOK AT SIGN OF LO-BYTE
 1210         BEQ .3       ...DON'T ADD MULTIPLICAND
 1220         CLC
 1230         LDA P
 1240         ADC B
 1250         STA P
 1260         BCC .3
 1270         INC P+2      ADD CARRY TO HI-16
 1280  *---------------------------------
 1290  .3     ASL A        SHIFT MULTIPLIER
 1300         BCC .4
 1310         CLC
 1320         LDA P+1      ADD TO MIDDLE OF PRODUCT
 1330         ADC B
 1340         STA P+1
 1350         BCC .4
 1360         INC P+3      (NEVER BOTHERS P+4)
 1370  *---------------------------------
 1380  .4     DEX
 1390         BNE .1
 1400         SEC
 1410         XCE
 1420         RTS
 1430  *---------------------------------
 1440  T
 1450         JSR MUL.EVEN.FASTER
 1460         LDA P+3
 1470         JSR PRB
 1480         LDA P+2
```

```
1490        JSR PRB
1500        LDA P+1
1510        JSR PRB
1520        LDA P+0
1530 PRB    JMP $FDDA
1540 *--------------------------------
1550        .LIF
```

==============================================================================
DOCUMENT :AAL-8601:DOS3.3:S.Mult.16.16.txt
==============================================================================

```
1000  *SAVE S.MULTIPLY 16X16
1010  *--------------------------------
1020  PLICAND    .EQ $00,01    MULTIPLICAND
1030  PLIER      .EQ $02,03    MULTIPLIER, LO-16 OF PRODUCT
1040  PRODUCT    .EQ $04,05    HI-16 OF PRODUCT
1050  *--------------------------------
1060         .OP 6502
1070  *--------------------------------
1080  MULTIPLY.16X16.6502
1090         LDX #16
1100  .1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
1110         LSR
1120         BCC .2       ...DON'T ADD MULTIPLICAND
1130         CLC
1140         LDA PRODUCT
1150         ADC PLICAND
1160         STA PRODUCT
1170         LDA PRODUCT+1
1180         ADC PLICAND+1
1190         STA PRODUCT+1
1200  .2     ROR PRODUCT+1
1210         ROR PRODUCT
1220         ROR PLIER+1
1230         ROR PLIER
1240         DEX
1250         BNE .1
1260         RTS
1270  *--------------------------------
1280         .OP 65802
1290  *--------------------------------
1300  MULTIPLY.16X16.65802.SMALLER
1310         CLC
1320         XCE          NATIVE MODE
1330         REP #$20     A-REG 16-BITS
1340         LDX #16      LOOP 16 TIMES
1350  .1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
1360         LSR
1370         LDA PRODUCT  GET HI-16 OF PRODUCT
1380         BCC .2       ...DO NOT NEED TO ADD
1390         CLC
1400         ADC PLICAND
1410  .2     ROR
1420         STA PRODUCT
1430         ROR PLIER    USE FOR LO-16 OF PRODUCT
1440         DEX
1450         BNE .1
1460         SEC
1470         XCE          BACK TO EMULATION MODE
1480         RTS
```

```
*--------------------------------
MULTIPLY.16X16.65802.FASTER
       CLC
       XCE          NATIVE MODE
       REP #$20     A-REG 16-BITS
       LDA PLICAND
       BEQ .3       0*ANYTHING=0
       DEC
       STA PLICAND
       LDX #16      LOOP 16 TIMES
.1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
       LSR
       LDA PRODUCT  GET HI-16 OF PRODUCT
       BCC .2       ...DO NOT NEED TO ADD
       ADC PLICAND
.2     ROR
       STA PRODUCT
       ROR PLIER    USE FOR LO-16 OF PRODUCT
       DEX
       BNE .1
       SEC
       XCE          BACK TO EMULATION MODE
       RTS
.3     LDA PRODUCT  INITIAL ADDEND
       STA PLIER    LOW 16 OF PRODUCT
       STZ PRODUCT  HIGH 16 OF PRODUCT
       SEC
       XCE          BACK TO EMULATION MODE
       RTS
*--------------------------------
       .LIF
```

```
==============================================================================
DOCUMENT :AAL-8601:DOS3.3:S.MULTIPLY.8X8.txt
==============================================================================
```

```
1000  *SAVE S.MULTIPLY 8X8
1010  *--------------------------------
1020  CAND   .EQ 2
1030  PLIER  .EQ 3
1040  PROD   .EQ 4,5
1050  *--------------------------------
1060  *   FAST 6502 MULTIPLICATION, BY DAMON SLYE
1070  *   CALL APPLE, JUNE 1983, P45-48.
1080  *      (A-REG) = MULTIPLIER
1090  *      (X-REG) = MULTIPLICAND
1100  *      RETURNS PRODUCT IN A,X (X=LO-BYTE)
1110  *--------------------------------
1120  FAST.8X8.SLYE
1130         CPX #0
1140         BEQ .3        A*0=0
1150         DEX           DECR. CAND TO AVOID
1160         STX CAND         THE CLC BEFORE ADC CAND
1170         LSR           PREPARE FIRST BIT
1180         STA PLIER
1190         LDA #0
1200         LDX #8
1210  .1     BCC .2        NO ADD
1220         ADC CAND
1230  .2     ROR
1240         ROR PLIER
1250         DEX
1260         BNE .1
1270         LDX PLIER
1280         RTS
1290  .3     TXA
1300         RTS
1310  *--------------------------------
1320  FAST.8X8.RBSC
1330         CPX #0
1340         BEQ .3        A*0=0
1350         DEX           DECR. CAND TO AVOID
1360         STX CAND         THE CLC BEFORE ADC CAND
1370         LSR           PREPARE FIRST BIT
1380         STA PLIER
1390         LDA #0
1400         LDX #4
1410  .1     BCC .2        NO ADD
1420         ADC CAND
1430  .2     ROR
1440         ROR PLIER
1450         BCC .25       NO ADD
1460         ADC CAND
1470  .25    ROR
1480         ROR PLIER
```

```
1490        DEX
1500        BNE .1
1510        LDX PLIER
1520        RTS
1530 .3     TXA
1540        RTS
1550 *--------------------------------
1560        .OP 65816
1570 *--------------------------------
1580 *      MULTIPLIER IN A(15-8), MULTIPLICAND IN A(7-0)
1590 *      RETURN PRODUCT IN A(15-0)
1600 *--------------------------------
1610 MULTIPLY.8X8.65802
1620        PHX
1630        STA .2+1     SAVE MULTIPLICAND
1640        LDA #0
1650        CLC
1660        XCE
1670        LDX #8
1680        REP #$20     A-REG 16 BITS
1690        EOR ##$FF00  COMPLEMENT MULTIPLIER
1700 .1     ASL
1710        BCS .3       ...IF ORIGINAL BIT=0
1720 .2     ADC ##0      ADD MULTIPLICAND
1730 .3     DEX
1740        BNE .1
1750        SEC
1760        XCE
1770        PLX
1780        RTS
1790 *--------------------------------
1800        .LIF
```

```
==============================================================================
DOCUMENT :AAL-8601:DOS3.3:TextTransferObj.txt
==============================================================================

™‡  Ωÿ\ù◊\ËêÛ©†ù◊\`ò™©†ù◊\Ë‡-ê¯` Lçtò™Ë‡-∞Ω◊\Õt–Ûä® ä
 LÕt‰hh≠nLa
```

```
LX†π◊\…†–à^¿»`≠p …
†±ô‘\»¿!ê^`òH≠p …
†π‘\ë»
```

```
==============================================================================
DOCUMENT :AAL-8601:ProDOS:BROWNS.MOVE.txt
==============================================================================

 1000  *SAVE BROWNS.MOVE
 1010  *--------------------------------
 1020  * MOVE by H. Brown
 1030  * Jan 18/86
 1040  *--------------------------------
 1050  PTR    .EQ $00,01
 1060  BUFFER .EQ $200
 1070  RAMRD  .EQ $C002
 1080  RAMWRT .EQ $C004
 1090  ALTZP  .EQ $C008
 1100  BNKSEL .EQ $C073    RAMWORKS BANK SELECT REGISTER
 1110  ROM    .EQ $C082
 1120  RAM1   .EQ $C08B
 1130  RAM2   .EQ $C083
 1140  *--------------------------------
 1150         .OR $C00     ORG  AT BEGINNING OF A PAGE
 1160  *--------------------------------
 1170  COMMONPG
 1180         JMP INIT     BRUN OR JSR TO INITIALIZE
 1190         JMP MOVE     NORMAL ENTRY
 1200  *--------------------------------
 1210  *   INIT copies COMMONPG to all 64K banks
 1220  *--------------------------------
 1230  INIT   LDX BANKS    START WITH LAST 64K BANK
 1240  .1     LDA BANKS,X  GET  BANK #
 1250         STA BNKSEL   SELECT 64K BANK
 1260         STA RAMWRT+1 CHOOSE TO WRITE
 1270         LDY #0       COPY PAGE
 1280  .2     LDA COMMONPG,Y
 1290         STA COMMONPG,Y
 1300         INY
 1310         BNE .2       LOOP TO END OF PAGE
 1320         DEX
 1330         BNE .1       LOOP TO START OF TABLE
 1340         BEQ EXIT     RESTORE STANDARD MEMORY
 1350  *--------------------------------
 1360  *   enter MOVE with A = page (CX for 2nd DX)
 1370  *                   X = 64K bank #
 1380  *                   Carry SET for write, CLEAR for read
 1390  *--------------------------------
 1400  MOVE   BCS .3       BRANCH IF WRITING
 1410         CMP #$C0
 1420         BCS .1       BRANCH IF UPPER 16K
 1430         CPX #$FF     ---  READ 48K ---
 1440         BEQ .2       SKIP IF MAIN 64K
 1450         STA RAMRD+1  READ FROM AUX 48K
 1460         STX BNKSEL   SELECT 64K BANK
 1470         BNE .2
 1480  .1     JSR SEL16K   ---  READ 16K ---
```

```
               CPX #$FF
               BEQ .2       SKIP IF MAIN 64K
               STX BNKSEL   SELECT 64K BANK
               STA ALTZP+1  SELECT AUX 16K
        .2     CLC
               JSR COPYPAGE
               BEQ EXIT
        *-------------------------------
        *   WRITING
        *-------------------------------
        .3     CMP #$C0
               BCS .4       BRANCH IF UPPER 16K
               CPX #$FF     ---  WRITE 48K ---
               BEQ .5       SKIP IF MAIN 64K
               STX BNKSEL
               STA RAMWRT+1 WRITING TO AUX 48K
               BNE .5
        .4     JSR SEL16K   ---  WRITE 16K ---
               CPX #$FF
               BEQ .5
               STX BNKSEL
               STA ALTZP+1
        .5     SEC
               JSR COPYPAGE
        *-------------------------------
        EXIT   STY BNKSEL   RESORE STD 64K FOR VIDEO
               STA RAMWRT   MAIN 48K
               STA RAMRD
               STA ALTZP    MAIN 16K
               LDA ROM
               RTS
        *-------------------------------
        *   BANKS is a table of 64K bank #'s, where
        *      FF = main 64k, 00 = alt 64K when no RAMWORKS
        *      00,04,08,0C = banks of a 256K RAMworks
        *      1st entry is # of banks
        *-------------------------------
        BANKS  .HS 05       Five banks all told
               .HS FF.00.04.08.0C
        *-------------------------------
        *   COPYPAGE copies 256 bytes
        *      from (PTR) in specified bank to motherboard $200
        *   or from motherboard $200 to (PTR) in specified bank
        *-------------------------------
        COPYPAGE
               STA PTR+1
               LDY #0
               STY PTR
               BCS .2
        .1     LDA (PTR),Y
               STA BUFFER,Y
               INY
               BNE .1
               RTS
```

```
2030  .2     LDA BUFFER,Y
2040         STA (PTR),Y
2050         INY
2060         BNE .2
2070         RTS
2080  *--------------------------------
2090  *   SEL16K selects the appropriate bank in 16K area
2100  *--------------------------------
2110  SEL16K CMP #$D0
2120         BCS .1
2130         LDY RAM2     C0   -> AUX D0
2140         LDY RAM2
2150         ADC #$10
2160         RTS
2170  .1     LDY RAM1     SELECT RD/WRT RAM
2180         LDY RAM1
2190         RTS
2200  *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8601:ProDOS:POTTSTEXTCOPIER.txt
==============================================================================

1000  *SAVE POTTS TEXT COPIER
1010  *--------------------------------
1020         .OR $300
1025         .TF TEXT.TRANSFER.OBJ
1030  *--------------------------------
1040  MY.BUFFER  .EQ $2000
1050  *--------------------------------
1060  BUFFER     .EQ $E0,E1    POINT TO FILE BUFFER
1070  RESULT     .EQ $E2       FILE MANAGER RETURN CODE
1080  *--------------------------------
1090  FM.SETUP   .EQ $3DC      INITIALIZE Y & A
1100  FM.ENTRY   .EQ $3D6      FILE MANAGER ENTRY POINT
1110  FM.BLK     .EQ $B5BB     FILE MANAGER PARM LIST
1120  *--------------------------------
1130  *   SET UP JUMP VECTORS
1140         JMP INITIALIZE.AND.READ
1150         JMP FIND.END.AND.WRITE
1160  *--------------------------------
1170  INITIALIZE.AND.READ
1180  *--------------------------------
1190  INITIALIZE.THE.BUFFER
1200         LDA #MY.BUFFER
1210         STA BUFFER        LSB
1220         LDA /MY.BUFFER
1230         STA BUFFER+1      MSB
1240         LDY #0
1250  .1     LDA #0            CLEAR BUFFER UP TO $95FF
1260  .2     STA (BUFFER),Y
1270         INY               NEXT BYTE IN THIS PAGE
1280         BNE .2            ...STILL IN THE PAGE
1290         INC BUFFER+1      NEXT PAGE
1300         LDA BUFFER+1
1310         CMP #$96          AT END OF STORAGE?
1320         BNE .1            ...NO, KEEP CLEARING
1330         LDA /MY.BUFFER    RESET BUFFER POINTER
1340         STA BUFFER+1
1350  *--------------------------------
1360  READ.THE.FILE
1370         LDX #9            10 BYTES
1380  .1     LDA RD.BLK,X
1390         STA FM.BLK,X
1400         DEX
1410         BPL .1
1420         JSR FM.SETUP
1430         JSR FM.ENTRY
1440         LDA FM.BLK+10     GET RETURN CODE
1450         STA RESULT        SAVE FOR APPLESOFT PEEK
1460         RTS               RETURN TO APPLESOFT
1470  *--------------------------------
```

```
FIND.END.AND.WRITE
*--------------------------------
FIND.END.OF.BUFFER
       LDY #0            SEARCH FOR 00 BYTE
.1     LDA (BUFFER),Y
       BEQ .2            ...FOUND END
       INY
       BNE .1            ...NEXT BYTE IN SAME PAGE
       INC BUFFER+1      NEXT PAGE
       BNE .1            ...ALWAYS
.2     STY BUFFER        LSB OF EOF BYTE
*--------------------------------
WRITE.FILE
       LDX #9            10 BYTES
.1     LDA WR.BLK,X
       STA FM.BLK,X
       DEX
       BPL .1
       LDA BUFFER        LSB
       STA FM.BLK+6      LSB OF FILE LENGTH
       SEC
       LDA BUFFER+1
       SBC /MY.BUFFER
       STA FM.BLK+7      MSB OF FILE LENGTH
       JSR FM.SETUP
       LDX #1            IF NO FILE, ALLOCATE ONE
       JSR FM.ENTRY      WRITE THE FILE
       LDA FM.BLK+10     RETURN CODE
       STA RESULT        SAVE FOR APPLESOFT PEEK
       RTS               RETURN TO APPLESOFT
*--------------------------------
RD.BLK .HS 03.02.0000.0000
       .DA $9600-MY.BUFFER,MY.BUFFER
WR.BLK .HS 04.02.0000.0000
       .DA $9600-MY.BUFFER,MY.BUFFER
*--------------------------------
```

```
==============================================================
DOCUMENT :AAL-8601:ProDOS:S.LORESTOHIRES.txt
==============================================================

 1000  *SAVE S.LORES TO HIRES
 1010  *--------------------------------
 1020  LBAS   .EQ $26,27
 1030  HBAS   .EQ $2A,2B
 1040  SAVEX  .EQ $2E
 1050  COLOR  .EQ $30
 1060  *--------------------------------
 1070  T
 1080         JSR PLOT
 1090         LDA $C050    GRAPHICS
 1100         LDA $C052    SOLID (40 BY 48 PIXELS)
 1110         LDA $C054    PRIMARY PAGE
 1120  .1     LDA $C056    LO-RES
 1130         JSR PAUSE.FOR.ANY.KEY
 1140         BEQ .2       ...<RETURN>
 1150         LDA $C057    HIRES
 1160         JSR CONVERT
 1170         JSR PAUSE.FOR.ANY.KEY
 1180         BNE .1       ...NOT <RETURN>
 1190  .2     LDA $C051    TEXT
 1200         RTS
 1210  *--------------------------------
 1220  PAUSE.FOR.ANY.KEY
 1230  .1     LDA $C000    WAIT FOR ANY KEY
 1240         BPL .1       ...NOT YET
 1250         STA $C010    CLEAR STROBE
 1260         CMP #$8D     SET .EQ. IF <RETURN>
 1270         RTS
 1280  *--------------------------------
 1290  CONVERT
 1300         LDX #23      OR #19 IF MIXED MODE
 1310  .1     LDY #39      COLUMNS 0...39
 1320         LDA LOL,X    SET UP BASE POINTER FOR LINE
 1330         STA LBAS
 1340         STA HBAS     SAME FOR HI-RES
 1350         LDA LOH,X
 1360         STA LBAS+1
 1370         EOR #$24     SHIFT FROM $400 TO $2000 FOR HI-RES
 1380         STA HBAS+1
 1390         STX SAVEX    SAVE X-REG
 1400  .2     LDA (LBAS),Y GET TWO LO-RES PIXELS
 1410         PHA          SAVE FOR LOWER ONE
 1420         ASL          UPPER PIXEL * 8
 1430         ASL
 1440         ASL
 1450         JSR PROCESS.NYBBLE
 1460         PLA          GET LOWER PIXEL
 1470         LSR          TIMES 8
 1480         JSR PROCESS.NYBBLE
```

```
        DEY          NEXT COLUMN, SCANNING RIGHT TO LEFT
        BPL .2       ...ANOTHER ONE
        LDX SAVEX    RESTORE X-REG
        DEX          NEXT LINE, SCANNING BOTTOM TO TOP
        BPL .1       ...ANOTHER ONE
        RTS          FINISHED!
*--------------------------------
LOH     .HS 04.04.05.05.06.06.07.07  HIGH BYTES
        .HS 04.04.05.05.06.06.07.07  OF SCRN PNTRS
        .HS 04.04.05.05.06.06.07.07  (TEXT OR LO-RES)
*--------------------------------
LOL     .HS 00.80.00.80.00.80.00.80  LOW BYTES
        .HS 28.A8.28.A8.28.A8.28.A8  OF SCRN PNTRS
        .HS 50.D0.50.D0.50.D0.50.D0
*--------------------------------
PROCESS.NYBBLE
        AND #$78     MASK THE SHIFTED NYBBLE
        STA COLOR
        TYA          LO-RES COLUMN
        AND #3       LOW 2 BITS
        ORA COLOR         0CCCC0YY
        TAX
        JSR COMMON.CODE
        EOR #$0C          3RD LINE OF 4
        STA HBAS+1
        JSR COMMON.CODE
        EOR #$1C          NEXT LINE
        STA HBAS+1
        RTS
*--------------------------------
COMMON.CODE
        LDA SHADES,X      EVEN LINE
        STA (HBAS),Y
        LDA HBAS+1
        ORA #4
        STA HBAS+1
        LDA SHADES+4,X    ODD LINE
        STA (HBAS),Y
        LDA HBAS+1
        RTS
*--------------------------------
SHADES  .HS 00.00.00.00.00.00.00.00  0--BLACK
        .HS AA.D5.AA.D5.55.2A.55.2A  1--MAGENTA
        .HS 91.A2.C4.88.C4.88.91.A2  2--DARK BLUE
        .HS 11.22.44.08.44.08.11.22  3--PURPLE
        .HS 2A.55.2A.55.2A.55.2A.55  4--DARK GREEN
        .HS 33.66.4C.19.4C.19.33.66  5--GRAY 1
        .HS D5.AA.D5.AA.D5.AA.D5.AA  6--MEDIUM BLUE
        .HS DD.BB.F7.EE.F7.EE.DD.BB  7--LIGHT BLUE
        .HS A2.C4.88.91.88.91.A2.C4  8--BROWN
        .HS AA.D5.AA.D5.AA.D5.AA.D5  9--ORANGE
        .HS B3.E6.CC.99.CC.99.B3.E6  A--GRAY 2
        .HS D5.AA.D5.AA.AA.D5.AA.D5  B--PINK
        .HS 6E.5D.3B.77.3B.77.6E.5D  C--LIGHT GREEN
```

```
2030        .HS 2A.55.2A.55.AA.D5.AA.D5  D--YELLOW
2040        .HS 2A.55.2A.55.D5.AA.D5.AA  E--AQUAMARINE
2050        .HS 7F.7F.7F.7F.7F.7F.7F.7F  F--WHITE
2060 *--------------------------------
2070 *   FILL CORNER WITH SAMPLES OF EACH COLOR
2080 *--------------------------------
2090 PLOT   LDY #0
2100        STY COLOR
2110 .1     LDX #3
2120 .2     LDA COLOR    00, 44, 88, CC
2130        STA $400,Y   GR ROWS 0-3
2140        STA $480,Y
2150        CLC
2160        ADC #$11     11, 55, 99, DD
2170        STA $500,Y   GR ROWS 4-7
2180        STA $580,Y
2190        ADC #$11     22, 66, AA, EE
2200        STA $600,Y   GR ROWS 8-11
2210        STA $680,Y
2220        ADC #$11     33, 77, BB, FF
2230        STA $700,Y   GR ROWS 12-15
2240        STA $780,Y
2250        INY
2260        DEX
2270        BPL .2
2280        ADC #$11     .., 44, 88, CC, END
2290        STA COLOR
2300        BCC .1       ...MORE
2310        RTS
2320 *--------------------------------
```

```
==================================================================
DOCUMENT :AAL-8601:ProDOS:S.MUL16X1665802.txt
==================================================================

 1000  *SAVE S.MUL16X1665802
 1010  *--------------------------------
 1020         .OP 65802
 1030  *--------------------------------
 1040  A       .EQ 0,1
 1050  B       .EQ 2,3
 1060  P       .EQ 4,5,6,7
 1070  *--------------------------------
 1080  MUL.EVEN.FASTER
 1090         CLC
 1100         XCE           ENTER NATIVE MODE
 1110         REP #$20      16-BIT A-REGISTER
 1120         STZ P+2       MAKE SURE NO ADDEND IN HI-16
 1130         STZ P   (DELETE IF WANT AN ADDEND IN LO-16)
 1140         LDX #8
 1150         BRA .2        ...HOP OVER SHIFTS
 1160  *--------------------------------
 1170  .1     ASL P         DOUBLE THE PRODUCT
 1180         ROL P+2
 1190  .2     LDA A
 1200         AND ##$0080   LOOK AT SIGN OF LO-BYTE
 1210         BEQ .3        ...DON'T ADD MULTIPLICAND
 1220         CLC
 1230         LDA P
 1240         ADC B
 1250         STA P
 1260         BCC .3
 1270         INC P+2       ADD CARRY TO HI-16
 1280  *--------------------------------
 1290  .3     ASL A         SHIFT MULTIPLIER
 1300         BCC .4
 1310         CLC
 1320         LDA P+1       ADD TO MIDDLE OF PRODUCT
 1330         ADC B
 1340         STA P+1
 1350         BCC .4
 1360         INC P+3       (NEVER BOTHERS P+4)
 1370  *--------------------------------
 1380  .4     DEX
 1390         BNE .1
 1400         SEC
 1410         XCE
 1420         RTS
 1430  *--------------------------------
 1440  T
 1450         JSR MUL.EVEN.FASTER
 1460         LDA P+3
 1470         JSR PRB
 1480         LDA P+2
```

```
1490        JSR PRB
1500        LDA P+1
1510        JSR PRB
1520        LDA P+0
1530 PRB    JMP $FDDA
1540 *--------------------------------
1550        .LIF
```

==================================================================
DOCUMENT :AAL-8601:ProDOS:S.MULTIPLY16X16.txt
==================================================================

```
1000  *SAVE S.MULTIPLY 16X16
1010  *--------------------------------
1020  PLICAND    .EQ $00,01    MULTIPLICAND
1030  PLIER      .EQ $02,03    MULTIPLIER, LO-16 OF PRODUCT
1040  PRODUCT    .EQ $04,05    HI-16 OF PRODUCT
1050  *--------------------------------
1060         .OP 6502
1070  *--------------------------------
1080  MULTIPLY.16X16.6502
1090         LDX #16
1100  .1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
1110         LSR
1120         BCC .2       ...DON'T ADD MULTIPLICAND
1130         CLC
1140         LDA PRODUCT
1150         ADC PLICAND
1160         STA PRODUCT
1170         LDA PRODUCT+1
1180         ADC PLICAND+1
1190         STA PRODUCT+1
1200  .2     ROR PRODUCT+1
1210         ROR PRODUCT
1220         ROR PLIER+1
1230         ROR PLIER
1240         DEX
1250         BNE .1
1260         RTS
1270  *--------------------------------
1280         .OP 65802
1290  *--------------------------------
1300  MULTIPLY.16X16.65802.SMALLER
1310         CLC
1320         XCE          NATIVE MODE
1330         REP #$20     A-REG 16-BITS
1340         LDX #16      LOOP 16 TIMES
1350  .1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
1360         LSR
1370         LDA PRODUCT  GET HI-16 OF PRODUCT
1380         BCC .2       ...DO NOT NEED TO ADD
1390         CLC
1400         ADC PLICAND
1410  .2     ROR
1420         STA PRODUCT
1430         ROR PLIER    USE FOR LO-16 OF PRODUCT
1440         DEX
1450         BNE .1
1460         SEC
1470         XCE          BACK TO EMULATION MODE
1480         RTS
```

```
*--------------------------------
MULTIPLY.16X16.65802.FASTER
       CLC
       XCE          NATIVE MODE
       REP #$20     A-REG 16-BITS
       LDA PLICAND
       BEQ .3       0*ANYTHING=0
       DEC
       STA PLICAND
       LDX #16      LOOP 16 TIMES
.1     LDA PLIER    CHECK NEXT BIT OF MULTIPLIER
       LSR
       LDA PRODUCT  GET HI-16 OF PRODUCT
       BCC .2       ...DO NOT NEED TO ADD
       ADC PLICAND
.2     ROR
       STA PRODUCT
       ROR PLIER    USE FOR LO-16 OF PRODUCT
       DEX
       BNE .1
       SEC
       XCE          BACK TO EMULATION MODE
       RTS
.3     LDA PRODUCT  INITIAL ADDEND
       STA PLIER    LOW 16 OF PRODUCT
       STZ PRODUCT  HIGH 16 OF PRODUCT
       SEC
       XCE          BACK TO EMULATION MODE
       RTS
*--------------------------------
       .LIF
```

```
==============================================================================
DOCUMENT :AAL-8601:ProDOS:S.MULTIPLY8X8.txt
==============================================================================

 1000  *SAVE S.MULTIPLY 8X8
 1010  *--------------------------------
 1020  CAND   .EQ 2
 1030  PLIER  .EQ 3
 1040  PROD   .EQ 4,5
 1050  *--------------------------------
 1060  *   FAST 6502 MULTIPLICATION, BY DAMON SLYE
 1070  *   CALL APPLE, JUNE 1983, P45-48.
 1080  *      (A-REG) = MULTIPLIER
 1090  *      (X-REG) = MULTIPLICAND
 1100  *      RETURNS PRODUCT IN A,X (X=LO-BYTE)
 1110  *--------------------------------
 1120  FAST.8X8.SLYE
 1130         CPX #0
 1140         BEQ .3       A*0=0
 1150         DEX          DECR. CAND TO AVOID
 1160         STX CAND        THE CLC BEFORE ADC CAND
 1170         LSR          PREPARE FIRST BIT
 1180         STA PLIER
 1190         LDA #0
 1200         LDX #8
 1210  .1     BCC .2       NO ADD
 1220         ADC CAND
 1230  .2     ROR
 1240         ROR PLIER
 1250         DEX
 1260         BNE .1
 1270         LDX PLIER
 1280         RTS
 1290  .3     TXA
 1300         RTS
 1310  *--------------------------------
 1320  FAST.8X8.RBSC
 1330         CPX #0
 1340         BEQ .3       A*0=0
 1350         DEX          DECR. CAND TO AVOID
 1360         STX CAND        THE CLC BEFORE ADC CAND
 1370         LSR          PREPARE FIRST BIT
 1380         STA PLIER
 1390         LDA #0
 1400         LDX #4
 1410  .1     BCC .2       NO ADD
 1420         ADC CAND
 1430  .2     ROR
 1440         ROR PLIER
 1450         BCC .25      NO ADD
 1460         ADC CAND
 1470  .25    ROR
 1480         ROR PLIER
```

```
1490        DEX
1500        BNE .1
1510        LDX PLIER
1520        RTS
1530 .3     TXA
1540        RTS
1550 *--------------------------------
1560        .OP 65816
1570 *--------------------------------
1580 *      MULTIPLIER IN A(15-8), MULTIPLICAND IN A(7-0)
1590 *      RETURN PRODUCT IN A(15-0)
1600 *--------------------------------
1610 MULTIPLY.8X8.65802
1620        PHX
1630        STA .2+1     SAVE MULTIPLICAND
1640        LDA #0
1650        CLC
1660        XCE
1670        LDX #8
1680        REP #$20     A-REG 16 BITS
1690        EOR ##$FF00  COMPLEMENT MULTIPLIER
1700 .1     ASL
1710        BCS .3       ...IF ORIGINAL BIT=0
1720 .2     ADC ##0      ADD MULTIPLICAND
1730 .3     DEX
1740        BNE .1
1750        SEC
1760        XCE
1770        PLX
1780        RTS
1790 *--------------------------------
1800        .LIF
```

==============================================================================
DOCUMENT :AAL-8602:Articles:ErvEdge.Wildcat.txt
==============================================================================

WildCAT for DOS 3.3....................................Erv Edge

WildCAT is a series of patches to DOS 3.3 which modify the CATALOG command. The new features include:

* A catalog by "wildcard" FILENAME facility.
* A catalog by FILETYPE facility.
* An alternate, short-form: either DIR or CAT.
* Catalog free space patch.
* Ctrl-Q catalog abort.
* TYPE a random or sequential text file.

Lee Reynold's FILEDUMP command has been re-packaged and re-presented as TYPE (see Call-A.P.P.L.E. 6/82 p47). More on this later. WildCAT, along with TYPE, is an attempt to teach new tricks to an old dog, as it were.

The normal DOS CATALOG command allows slot, drive, and volume parameters. I have added a filename parameter, but process it a little differently than the way file names are usually processed. To display the catalog entries for all files whose names contain a particular string, type any of the following:

```
   CATALOG ^string [,Dn] [,Sn] [,Vn]
   DIR ^string [,Dn] [,Sn] [,Vn]
   CAT ^string [,Dn] [,Sn] [,Vn]
```

where "^string" begins with the "^" or caret symbol (shifted N on the ][+ or shifted 6 on the //e or //c); the string should contain no blanks, although it may "end" with them; the string would normally end with a carriage return or with a comma if a drive or slot number is specified. Only those files that contain the "string" somewhere in the filename will be listed. (Of course you already know that the D, S, and V parameters are shown in brackets above because they are optional; you do not type the brackets.)

For example, "CATALOG ^TEST" would list each file with 'TEST' as part of the filename; while "DIR ^PAY." would list those with 'PAY.' somewhere in the filename; and "CAT^.OBJ,D2" would list filenames on drive 2 that contain the partial string '.OBJ'. "CAT" and "DIR" are simply synonyms for "CATALOG".

I have also arranged things so you can list the catalog entries of a specified file-type. You simply type the file type code in the CATALOG command:

```
   CATALOG t [,Dn] [,Sn] [,Vn]
   DIR t [,Dn] [,Sn] [,Vn]
   CAT t [,Dn] [,Sn] [,Vn]
```

where "t" is any of the unadorned, single-letter filetypes: A B I R S T. Only that type of file (if present) will be listed.

For example, "CATALOG T" would list all the text files; "DIR A,D2" would list all of the Applesoft files on drive 2; "CAT B,S5,D1" would list all the binary files on slot 5, drive 1. Yes, "DIRT" works just fine.

I added the TYPE command, which allows you to display the contents of text files. Both CATALOG and TYPE will optionally:

1. Print "hidden" control characters as inverse:
   POKE 234,0 to print as inverse (default)
   POKE 234,255 to function as-is

2. Lower case letters may be shifted to upper case:
   POKE -18700,255 no shift (default)
   POKE -18700,223 to shift lower to upper case.

You can slow down TYPE's output via SPEED=xx or POKE 241,xx; or pause by pressing any key; then Ctrl-Q to abort. Also, TYPE pauses and waits for a keypress when it encounters a hex 00 imbedded in the file or at end of file; press Ctrl-Q to quit. You may TYPE random text files by holding down REPT-SPACE to get past the hex 00's at the end of each logical record.

The listing that follows is intended for information only: it is not BRUNable. My intention is that you prepare the EXEC shown below to actually install the patches. Any word processor that produces a straight, sequential text file may be used to prepare the EXEC. Of course you can also use the S-C Macro Assembler for this purpose. Then, type EXEC WILDCAT to apply the patches to DOS 3.3 in memory. After checking it out and running any other tests you like, put in a new diskette, enter a HELLO program, and type INIT HELLO to "permanently" install WildCAT in the DOS on tracks 0, 1, and 2.

When I wrote WildCAT, I had two main goals in mind: it should be a (mostly in-place) code replacement, and it should be compatible with the known means of using (abusing?) the existing CATALOG code at $AD98-AE69.

One major design consideration was a mechanism for entering the ^string/type parameter. This required merely changing the "keyword parameter table" so CATALOG could have a "filename".

Next, a distinction had to be made between a "wildcard" and a "filetype" parameter. It made sense to 'delimit' the wildcard string; then the single-character filetype would be just that: a single character, entered without a delimiter. But this "phony" name mechanism has it's own problems:

First, "What's in a Name?" (DOS Manual p. 16): a filename has to start with a letter...which automatically eliminates most special characters (eg, equal, pound, slash, colon, etc) as the delimiter. The command parsing routine doesn't really know what it's working on at the time. All it knows is: if a name may be present, it must be valid. The validity test is only that the first character be equal to or greater that $C0 or an @-sign. The @-sign could have been used, but it's a problem on some 80-column boards; the ^ or caret works nicely (and besides, it looks good).

Second, now that we have a name (however, phony) and since the CATALOG command lives in the File Manager (FM) portion of DOS, there will be a buffer allocated for it. Unfortunately, the Command Interpreter (CI) DOCAT routine, which calls the FM, already "knows" that there will not really be a name, so it does not include housekeeping code to deallocate a buffer. So merrily allocating files without closing them...after the third time: NO BUFFERS AVAILABLE. And naively adding CLOSE (even if there were room for it), would have one very undesirable side effect if a "regular" catalog were requested: CATALOG-CLOSE without FNAME will close all open files. WildCAT instead plays a little shell game with DOS: The new DOCAT routine saves the first character of FNAME and substitutes a zero. Thereafter, neither the File Manager nor the rest of DOS ever knows that a name has been entered, so FM never actually allocates a buffer.

Third, what really should happen if a phony name is not entered? A regular catalog, of course, but how would this be indicated to WildCAT? Well, the shell game has a sting. Early on when the CI PARSE routine discovers that a filename is a valid parameter, it first clears FNAME to all blanks, expecting to fill it in with whatever comes in next. If a comma or carriage return comes in next, then FNAME still contains the blank; and that's what WildCAT saves off (under the shell) before it substitutes the zero.

Thus, the "sting" is that the CI "tricks" itself into telling WildCAT what to do in the absence of a ^string/type specifier: WildCAT takes a blank to indicate "do a regular" catalog; just as positively as a "^" indicates "do a wildcard" catalog, and a single character indicates "do a filetype" catalog.

The blank indicator also helps satisfy the second goal above and solves the problem of compatibilty with the "known means" of using/abusing the existing CATALOG code. WildCAT simply has to put a blank under the shell at each of the points where the code could most reasonably be entered without going thru the Command Interpreter's new DOCAT routine. That's exactly what all the JSR's to the routine AllowENTry are doing.

Satisfying that second goal takes up a lot of space, however; and has somewhat undermined the first constraint: WildCAT certainly isn't "in-place" in one place! And I apologize for this rather bizarre, serpentine code; I do hope that now you understand why some things were done the way they were.

Although considerable effort was spent to maintain compatibilty with the existing DOS commands, there were some compromises:

1. While the DOS manual (page 22) states: "To specify drive 1, you use the notation D1 separated from the file name by a comma", you can in fact leave out the comma between CATALOG and D1. With WildCAT that comma is now required; otherwise, it would take the "D" as a filetype and try to find it ... which of course it wouldn't and there would be no files reported. This would also be a problem for Applesoft programs that do something like: PRINT D$"CATALOG D1" without the comma. Therefore, WildCAT issues a (late) "SYNTAX ERROR" message if it encounters an undelimited string of length 2 or more.

2. CATALOG is a favorite routine to execute directly, bypassing the DOS Command Interpreter. FID, for example, provides its CATALOG via the "external" entry to the File Manager, which means that the main entry at CATHNDLR must provide for a "regular" catalog. It is also possible from machine language, however, to bypass both the CI and the FM. This usually involves changing the exit JMP address at DONEXT2 (to return to the user's code) and then jumping directly into almost anywhere in the CATALOG code (see the Listing 1 labels that begin "at"). I believe most of these cases are covered, but you may find some programs, which provide an "internal" CATALOG, that just won't work with WildCAT.

3. In order to both gain some patch space and provide the DIR/CAT short-form command name, the DOS command POSITION was eliminated. You may have to look it up just to find out that it is, much less what it is. Its relative rarity may be due to its implementation: it, like APPEND, finds its way through the file one byte at a time...all day long. Any program that uses it will now get a syntax error. If POSITION is really needed, it can be readily simulated by programming a read-loop to discard N-1 fields before processing the desired Nth field.

The following is a brief commentary on the assembly listing. The paragraph numbers correspond to numbers in comment lines.

The page zero locations I used ($EB thru $EF) are free, i.e. not used by DOS, the Monitor, or the Basics.

(1) In CMDTBL, replace Integer CHAIN address with TYPE and DOCAT address with NewDOCAT.

(2) Rearrange some code (and change both references to it) to add a "print blank" capability. The Command Interpreter uses its own vector to a "COUT" routine via CSW at $36; however, the File Manager (previously) used the Monitor COUT and CROUT routines for printing the catalog. With WildCAT all of DOS now consistently uses the vector at $9FCA for output; plus it has a new BlankOUT routine, all within the original code space.

(3) Recode a very cumbersome form of the "indexed indirect jump" to use register Y and leave X (which is zero by a previous operation) so it can be used in NewDOCAT.

(4) Replace old DOCAT's 12 bytes of code with a JMP to NewDOCAT and use the remainder to space over to column 7 after the file length has been displayed.

(5) NewDOCAT saves the first character of FNAME and substitutes a zero to prevent buffer allocation. It then loads 13, the new Catalog Function Code, and proceeds to CMDHNDLR2. Function 13 enters the catalog code past the "allow for irregular, direct entry".

(6) In the keyword parameter table, change parms to allow a filename with CATALOG and a filename, drive, and slot with DIR. Set new Function 13 address (previously a "no-op" to NOERROR) to WildCAT and change the range check to 14 to allow for it.

(7) Replace the Integer CHAIN code; PrtLOCK displays an asterisk or blank if the file is locked or not.

(8) Shorten the "NO BUFFERS AVAILABLE" message to "NO BUFFER" and re-use the space to decide which Basic is active, then JMP to the appropriate decimal print routine; used to print the free sector value and catalog filesizes. The value to be printed has been previously loaded into A and X.

(9) First, eliminate the need for "NOT DIRECT COMMAND" error message and then re-use the space to check for a "regular" catalog (no filename) or for a catalog by filetype (undelimited, single character). If more than a single, non-blank character is detected (ie, 2nd byte of FNAME is not blank), then "SYNTAX ERROR" message is issued.

(10) At beginning of catalog code allow for most normal points where the code could be directly entered. The new "official" Function 13, WildCAT initializes the FM workarea (per normal) and branches to Read VTOC to "find" the first catalog sector.

(11) Freespace "prolog"; clear carry and branch around another likely "irregular" entry point. Read first/next catalog sector, then lookup and save the filetype. Setup Y with 30 for name length and branch to CkFNAME.

(12) AllowVTOC fakes a "regular" catalog and falls into a JSR to read the VTOC. The BCC to initialize linecount is always taken; only if there had been an I/O error would the carry be set, in which case, control would have passed to the error-message-print exit anyway.

(13) PrtCat displays a catalog line. Note that loc $24, CH, is "POKEd" with 7 for uniform spacing over to the filename. If your printer interface board or 80-column card do not support this convention, then the display will not be properly spaced. The DONEXT routine is unchanged. SKIPLN has been re-arranged to allow init linecount, put out a carriage return, and check for a keypress (Ctrl-Q to quit) after 22 lines. Note: This leaves the cursor in column 37; see below.

(14) CkFNAME "looks under the shell" to figure out what to do. A caret indicates to check for a wildcard string. After JSR to CkCAT, if the equal status is set, then branch to print the catalog line. DoWild checks for the occurence of the wildcard string within the filename. $B4C9,X indexes the name in the Catalog Sector; $AA75,Y indexes the wildcard string; CatNmLen counts from 30 to 0, to scan the whole name.

(15) FreeSpce counts the free sectors, as indicated by the VTOC, loads X and A with the count, and JMPs ToPrtDec.

(16) WaitCk79 provides the "wait" for TYPE; also checks and puts out a carriage return after 79 characters to avoid over- printing long lines on certain printers, such as the MX-80.

(17) TYPE displays the contents of a sequential or random text file. A keypress will pause the display, and Ctrl-Q quits.

(18) InvCOUT is used by both CATALOG and TYPE. It converts hi- bit off characters to proper inverse. It will optionally show control characters as inverse or allow them to "function" as- is; and it will optionally "shift" lower case letters to upper case, if you do not have a lower case adapter; see "...Options" above. Loc $EA, decimal 234, is the Applesoft Hi-Res collision counter; it should always be zero, unless you POKE it.

(19) WaitCQ waits for a keypress and sets the equal status, if Ctrl-Q was pressed.

(20) Replace the inverted phrase DISK VOLUME with FREE SPACE=.

(21) The DOSCMDS list is moved down 6 bytes. AllowENT re-uses these 6 bytes to force a blank in FName1 "under the shell" to ease "irregular" entries into the catalog code; and clears the carry in case the entry was 'atADC9' which also cleared the carry. In the command list, TYPE replaces CHAIN and DIR replaces POSITION; change $A8BF:43 41 D4 to replace with CAT.

(22) Change the two references to DOSCMDS to the new location. These two changes must be done last as the EXEC is changing the very code that is executing.

I would like to thank Lee Reynolds and Art Schumer for their helpful comments and suggestions.

=====================================================================
DOCUMENT :AAL-8602:Articles:Faster.CRCs.txt
=====================================================================

Faster Cyclic Redundancy Checking..........Bob Sander-Cederlof

In the April 1984 issue of AAL I showed how to compute a cyclic redundancy check code (CRC) for a buffer full of data. I also tried to explain a little of the theory, as much as I understood. In the June 1984 issue Bruce Love explained how to work backward from the computed CRC of a received buffer to correct a single bit error. Both of these programs were written in plain 6502 code.

In the February 1986 "Dr. Dobb's Journal" Terry Ritter writes about "The Great CRC Mystery". He also presents some Pascal programs and 8088 machine code programs for calculating the CRC in various ways. Terry describes very briefly a table driven method (the very fastest way) and a byte-oriented method (almost as fast as table-driven).

I translated Terry's machine-coded byte-oriented method from 8088 to 65802 code, but even after twiddling and tweaking for half a day I could not make it give the correct answers. I don't know if his method is correct or not, but of course it MUST be, since it is printed in Dr. Dobb's and since he claims it works and since he even tells how many milliseconds it takes.

Anyway, I decided to derive my own byte-oriented method. The CRC algorithm is basically a "long division" of the entire bit stream in the buffer as though it were one long binary word. The divisor is $11021 in the CCITT scheme. The check code we use is the remainder of the division. The normal algorithm does "long division" on a bit-by-bit basis. The byte-oriented algorithm does "long division" on a byte-by-byte basis.

I put long division in quotation marks above because it is not EXACTLY long division. The difference is that the subtraction steps are replaced with exclusive-or operations. The exclusive-or is performed whenever the leading bit of the new dividend is a 1-bit. Here is a fully worked out example, for a CRC-so-far = $E1F0, and the next byte = $CC:

```
       "divide"   $E1F0CC by $11021, "quotient bits" down
       the left edge.  Next CRC is the "remainder"

       1110 0001 1111 0000 1100 1100   (E1F0CC in binary)
1  eor 1000 1000 0001 0000 1           (11021 in binary)
       ---------------------
        110 1001 1110 0000 01
1   eor 100 0100 0000 1000 01
        ---------------------
         10 1101 1110 1000 000
1    eor 10 0010 0000 0100 001
         ---------------------
```

```
0         0 1111 1110 1100 0010
            1111 1110 1100 0010 1
1     eor   1000 1000 0001 0000 1
            ---------------------
             111 0110 1101 0010 01
1      eor   100 0100 0000 1000 01
             ---------------------
              11 0010 1101 1010 000
1       eor   10 0010 0000 0100 001
              ---------------------
               1 0000 1101 1110 0010
1        eor   1 0001 0000 0010 0001
               ---------------------
                 0001 1101 1100 0011 = $1DC3
```

Note that the "quotient" is $EF.  This "quotient" can always be exactly computed by using just the first byte of the dividend (the high byte of the old CRC code):  quotient = crchi .eor. crchi/16.  If you carefully study the worked out example above, you should be able to see why this is true.  Now, if we use the exclusive-or rather than addition to perform a multiplication of the quotient times $11021, it will look like this:

```
                      uuuu.vvvv  (symbolic quotient in binary)
                       x $11021  (multiplier in hexadecimal)
                       --------
                      uuuu.vvvv
               u.uuuu.vvv0
       uuuu.vvvv
  uuuu.vvvv
  -----------------------------
    whatever........
```

There are several significant things to notice about the multiplication above.  First, we only need to save the rightmost 16 bits of the "product".  If we exclusive-or those bits with the rightmost 16 bits of the original dividend (which means the low byte of the old CRC followed by the new byte), we will get the next CRC. (This trick relies on the fact that exclusive-or is a reversible operation, so that "adding" and "subtracting" give the same result!)

Furthermore, we can organize those "partial products" in a more efficient way for computation.  Now, let's write the original CRC symbolically as "aaaa.bbbb.cccc.dddd", and the next data byte as "eeee.ffff".  The "quotient" after "dividing" by $11021 will be "aaaa.bbbb exclusive-or 0000.aaaa"; let's write that symbolically as "aaaa.gggg".  Then we can compute the next CRC code by the following very simple steps:

```
       cccc.dddd.eeee.ffff
  eor  gggg.0000.aaaa.gggg
  eor  000a.aaag.ggg0.0000
       -------------------
       wwww.xxxx.yyyy.zzzz
```

Believe it or not!

The program that follows implements this algorithm, in lines 1550-1760. I used 65802 code, but it really could be done quite nicely in plain 6502 as well. I leave it as "an exercise for the reader" (as college textbooks are wont to say), should you wish to try the algorithm in a plain-vanilla 6502.

The SEND and RECV programs simulate sending and receiving a buffer-full of data. I chose to put my buffer at $4000, for 258 bytes. This is the same as in the April 1984 article.

The FIND.BAD.BIT program is simply a translation of Bruce Love's 1984 program into 65802 code. Thanks to 16-bit registers, it is significantly faster and shorter.

Speaking of speed, the code for computing the next CRC code for one new byte takes (if I counted correctly) 57 clock cycles. In a normal Apple that means about 56 microseconds. The time for 8088 machine code in Terry Ritter's article was 17 microseconds for the equivalent steps. He was running with a 7.16 MHz clock. If you ran the 65802 code in an Applied Engineering Transwarp card or a Titan Accelerator card with a 4-MHz 65802 (running at 3.58 MHz), the time would be only 15.9 microseconds in an Apple.

==============================================================
DOCUMENT :AAL-8602:Articles:Front.Page.txt
==============================================================

$1.80

Volume 6 -- Issue 5 February, 1986

In This Issue...



Bag of Tricks 2

You've been asking when Bag of Tricks, that very popular and useful disk utility package, will be updated for ProDOS. Well, you can relax now: it's here.

The new ZAP program in "Bag of Tricks 2" adds the ability to access ProDOS blocks, directories, and files; the 80-column display can show most of a block at one view. The new version of FIXCAT can reconstruct a blown ProDOS directory, as far as is possible. You do still need to follow up with ZAP to correct things like file size and load address, which completely disappear when a directory is damaged.

This new, non-copy-protected edition of an old friend costs $49.95, or $45 + shipping from S-C. Owners of the older Bag of Tricks can get an upgrade directly from Quality Software for only $20 by returning your original disk.

Correction to Day of Week Programs

On page 20 of the December 1985 AAL, change lines 130 and 140 to the following:

```
     130 FOR I=0 TO 11 : READ MD(I) : NEXT
     140 FOR I=0 TO 6  : READ D$(I) : NEXT
```

On page 24, same issue, change line 120 to:

```
     120 FOR I=0 TO 6 : READ D$(I) : NEXT
```

That's what we get for typing a program into the Word Processor rather than printing a LISTing!

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage

for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).

==============================================================================
DOCUMENT :AAL-8602:Articles:Garbage.Correx.txt
==============================================================================

Correction to Fast Garbage Collector...Bob Sander-Cederlof

In the March 1984 AAL, Paul Shetler gave us a very fast garbage collector for Applesoft. Last week Keith Satterley called from Australia, and mentioned he thought there was a bug in the handling of strings over 128 characters long. I looked into it, and he is right.

The bug is in the loop in lines 3240-3320, on page 9 of that issue. The loop moves a string from one place in memory to another. The way we printed the code, a string longer than 128 characters would only have one byte moved! Here is the old code and the correct code, side-by-side:

```
-----old code------------        -----correct code--------

3240    LDY STRING.LENGTH        3240    LDY STRING.LENGTH
3250    DEY                      3250 .3 DEY
3260 .3 LDA (FRESPC),Y           3260    LDA (FRESPC),Y
3270    STA (LOWTR),Y            3270    STA (LOWTR),Y
3280    DEY                      3280    TYA
3290    BPL .3                   3290    BNE .3
3300    BMI .1                   3300    BEQ .1
```

Can you see why the new code works and the old doesn't?
.
.
.
.
.
.
.
1

==============================================================================
DOCUMENT :AAL-8602:Articles:Mitsubishi.txt
==============================================================================

Mitsubishi 50740 Series

I received information from several sources this week about an interesting new branch of the 6502 tree. Mitsubishi has pub- lished specs for eight varieties, all part of the "740 series". The chip is based on the 6502, adds some new addressing modes for some of the standard 6502 opcodes, and adds 13 new opcodes. (Unfortunately, the opcode enhancements are not compatible with any of the other enhanced 6502s.)

The chips in the 740 series are intended for use as microcon- trollers. As such, most of them have on-chip RAM and ROM. They all have built-in I/O ports, timers, and other goodies. The most interesting (to Don Lancaster, Nigel Nathan, and me) is the M50734. This chip, said to cost only $12, has four A/D converters, UART, six timers, a serial I/O port, four 8-bit I/O ports, a pair of stepper- motor drivers, and more. It all lives in a 64-pin shrink-DIP package. The M50734 is the only one in the 740 series which has no internal ROM and RAM. It is CMOS. The clock runs at 8 MHz, which in effect runs the opcodes at 2 MHz (that is, two cycle instructions take one microsecond).

To control all these functions, the bytes in page zero from $DA through $FF are used as I/O, control, and status registers.

One of the trickiest enhancements allows direct access (without bank switching or bank registers) to a second 64K memory, for data only. Apparently one of the address modes changes the state of one of the output signals during data memory references; if you use that signal to enable another bank of memory. ALMOST like having direct- addressability of 128K.

The data bus is multiplexed with half of the address bus, so it's a little harder to interface. Naturally, to get all the functions I mentioned above with only 64 pins, there have to be shared pins. Depending on which functions you are using, some of the timers and some of the I/O pins have dedicated uses.

The 6502 has one unused status bit. The 740 series calls this the T- flag, and gives it a use. If T=1, a special address mode is enabled which allows memory-to-memory operations without using the A-register. As I understand it, when T=1, address modes which use X as an index register take on a new meaning: rather than moving data between the indexed address and the A-register, data is moved between the absolute address and the zero page location whose address is in the X-register. If I am correct, ADC $400,X (assume X contains $34) would add the contents of $400 to the contents of $34, and store the result in $34. If T=0, indexing works in the old-fashioned 6502 way.

Another powerful enhancement allows you call subroutines with a two-byte version of the JSR opcode. One variation uses vectors stored in page zero, and the other uses vectors stored $FF00 through $FFF3. JMP can also uses vectors stored in page zero, so you have a two-byte JMP indirect.

Four new opcodes give you the ability to set, clear, or test any bit in the A-register or in page zero. This uses up 64 opcodes, because the bit number and bit state are coded into the opcode byte. Rockwell's version of the 65C02 includes page-zero bit-addressing, but the opcodes are not the same.

There are other new instructions, including several about which I do not have accurate complete information.

```
       RRF zp     (I think it swaps nybbles in the byte)
       COM zp     (Probably forms 2's complement at zp)
       LDM zp     (Probably loads ABS(zp) into A-register)
       CLT        clear T-bit in status
       SET        set T-bit in status
       STP        stop the clock until reset or interrupt
       WIT        low power mode   "     "   "      "
       SLW        (slow?)
       FST        (fast?)
       INC        increment A-reg
       DEC        decrement A-reg
       BRA rel    branch always.
```

Of all the extensions, only ONE (BRA) is compatible with the standard 65C02 and 65816 extensions from Western Design Center (the OFFICIAL source for 6502 designs). The others, even if they do the same thing, use a different opcode value. Why?

If you have worked up an appetite for more information on the 740 series, contact Mitsubishi. I don't have all their numbers, but you can get close by calling 1-800-421-1132.

When we get all the data, we will be writing a Cross Assembler so you can use your Apple to develop software for this chip.

==============================================================================
DOCUMENT :AAL-8602:Articles:RichardDOSPatch.txt
==============================================================================

DOS Patch:  Prevent Direct Commands.....Richard Gendron

I operate a AE/CATFUR line using my Apple and a modem in Montreal, Quebec.  I have found that protecting your DOS from illegal entry can be a tough job to say the least.

In searching for ways to protect my system, I came across an interesting address in DOS:  at $A026 there is some code which is executed whenever you try to type in a DOS command.  The code checks to see if the command you typed is allowed as a direct command, and if not gives you the NOT DIRECT COMMAND message (or ERROR 15 if you are using DiversiDOS).

I have written a little patch that will catch you when you type a DOS command, and re-RUN the Applesoft program.  If a sneaky caller finds a way to get out of the executing Applesoft program, at least he/she will be prevented from doing DOS commands.

Now every lock should have a key.  You do want to be able to use your own DOS in direct mode, so I have included a way to turn off the protection.  If you type "PRINT USR (0)" the system will respond with "PW:".  Then enter a two-character  password and the protection patch will be removed.  Then you can CATALOG, DELETE, or whatever you want to do.

Since I use Diversi-DOS, and in both the 48K and 64K configurations, I set up my patching program so that it will work with both.  The code which checks which version is loaded is in lines 1220-1260 and lines 1390-1410.  If the output hook at $36,37 points up to $BDxx or higher, the 64K version must be running.  Normal 48K DOS points to $9EBD.

These patches worked on my system, but yours may be a little different depending on which version of DOS you use.  Examine carefully all the addresses I use inside DOS to see if yours is the same as mine before you try to use these patches.

```
==============================================================================
DOCUMENT :AAL-8602:DOS3.3:Gendron.DOS.Mod.txt
==============================================================================

 1000  *SAVE GENDRON DOS MODS
 1010  *-------------------------------
 1020  *  DOS PROTECTION FOR THE DIRECT COMMAND "ERROR 15"
 1030  *   WRITTEN BY RICHARD GENDRON FOR USE ON TRANSFERS ][
 1040  *  (514) 738-1247 (AE/CAT-FUR)
 1050  *-------------------------------
 1060         .OR $300
 1070  *-------------------------------
 1080  INSTALL
 1090         LDA #$4C     BUILD "USR" VECTOR
 1100         STA $0A      "JMP" OPCODE
 1110         LDA #USR
 1120         STA $0B
 1130         LDA /USR
 1140         STA $0C
 1150  *---MOVE DATA INTO DOS-----------
 1160         LDX #P1-PATCHES      POINT AT OUR PATCHES
 1170  ***    JMP PATCH.DOS
 1180  *-------------------------------
 1190  PATCH.DOS
 1200         LDA #$A026
 1210         STA $00
 1220         LDA $37      48K OR 64K DOS?
 1230         CMP #$BD     CARRY CLEAR IF 48K
 1240         LDA /$A026   ...48K
 1250         BCC .1       ...48K
 1260         LDA /$E026   ...64K
 1270  .1     STA $01
 1280         LDY #4       MOVE 5 BYTES
 1290  .2     LDA PATCHES,X
 1300         STA ($00),Y
 1310         DEX
 1320         DEY
 1330         BPL .2
 1340         RTS
 1350  *-------------------------------
 1360  REBOOT
 1370         SEI                  TURN OFF ANNOYING INTERRUPTS
 1380         JSR $03EA            RESET THE I/O HOOKS
 1390         LDA $37              LETS SEE WHICH DOS WE ARE USING
 1400         CMP #$BD             IS IT 64K DOS ?
 1410         BMI .1               SNIFF, NO IT IS NOT
 1420         BIT $C081            YES IT IS, SO TURN OFF THE LANGUAGE
CARD
 1430         BIT $C081            TWICE, EVERYONE KNOWS WHY ?.
 1440         JSR $E316            DOS "CLOSE" ALL FILES
 1450         JMP $D566            NOW LET'S JUMP TO THE APPLESOFT
"RUN"
```

```
 1460  .1     JSR $A316           DOS "CLOSE" ALL FILES (NO TROUBLE PLS)
 1470         JMP $A4FC           48K INTERNAL "DOS RUN"
 1480  *--------------------------------
 1490  USR
 1500         LDY #0              START OF LOOP COUNTER
 1510  .1     LDA PASSWORD,Y      GET PASSWORD TEXT
 1520         BEQ .2              ...END OF STRING
 1530         JSR PRINT           NO, SO PRINT IT
 1540         INY                 INCREMENT THE LOOP
 1550         BNE .1              FOREVER LOOP (NEVER CAN EXIT)
 1560  *--------------------------------
 1570  .2     JSR INPUT           ALL TEXT PRINTED SO LET'S GET A KEY
 1580         CMP #"*             WAS IT A "*" ? (OR WHATEVER YOU WANT)
 1590         BNE .4              NO IT WAS NOT , SO BYE BYE
 1600         JSR INPUT           YES IT WAS, SO GET ANOTHER KEY
 1610         CMP #".             WAS IT A "." (OR WHATEVER YOU WANT)
 1620         BNE .4              NO IT WAS NOT, SO BYE BYE
 1630  *--------------------------------
 1640         LDX #P2-PATCHES
 1650         JSR PATCH.DOS
 1660  .4     RTS                 WE HAVE FINISHED
 1670  *--------------------------------
 1680  *   TEXT TO BE PRINTED WHEN A
 1690  * "PRINT  USE(0)" COMMAND IS DONE
 1700  *        IN APPLESOFT
 1710  *--------------------------------
 1720  PASSWORD
 1730         .AS -"PW:"
 1740         .HS 00
 1750  *--------------------------------
 1760  *  INPUT AND PRINT SUBROUTINES
 1770  *--------------------------------
 1780  PRINT  JMP ($36)
 1790  INPUT  JMP ($38)
 1800  *--------------------------------
 1810  PATCHES
 1820         JMP REBOOT   CALL OUR NEW CODE
 1830         NOP          NEEDED FOR DIVERSI-DOS
 1840  P1     NOP
 1850  *--------------------------------
 1860         .HS A9023909    ORIGINAL CODE
 1870  P2     .HS A9
 1880  *--------------------------------
 9999         .LIF
```

```
==========================================================================
DOCUMENT :AAL-8602:DOS3.3:S.CRC.GENERATOR.txt
==========================================================================

 1000  *SAVE S.CRC GENERATOR
 1010  *--------------------------------
 1020  BUFFER .EQ $4000
 1030  LIMIT  .EQ $4102
 1040  *--------------------------------
 1050  CRC    .EQ $00,01
 1060  PNTR   .EQ $02,03
 1070  TEMP   .EQ $0A,0B
 1080  *--------------------------------
 1090  PRNTAX .EQ $F941
 1100  CROUT  .EQ $FD8E
 1110  *--------------------------------
 1120  *      SIMULATE SENDING A BUFFER-FULL
 1130  *--------------------------------
 1140  SEND
 1150         LDA #0       CLEAR CRC BYTES IN BUFFER
 1160         STA LIMIT-1
 1170         STA LIMIT-2
 1180         JSR NEW.CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1190         LDX CRC      STORE CRC INTO LAST 2 BYTES
 1200         LDA CRC+1
 1210         STX LIMIT-1
 1220         STA LIMIT-2
 1230         JSR PRNTAX   DISPLAY THE CRC
 1240         JMP CROUT    <RETURN> AND RETURN
 1250  *--------------------------------
 1260  *      SIMULATE RECEIVING A BUFFER-FULL
 1270  *--------------------------------
 1280  RECV
 1290         JSR NEW.CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1300         LDX CRC      DISPLAY CRC IN HEX
 1310         LDA CRC+1
 1320         JSR PRNTAX
 1330         JMP CROUT
 1340  *--------------------------------
 1350         .OP 65802
 1360  *--------------------------------
 1370  *      CRCH      CRCL      DATA
 1380  *   aaaa.bbbb.cccc.dddd.eeee.ffff
 1390  *  +0000.aaaa
 1400  *   ---------
 1410  *   aaaa.gggg
 1420  *            +gggg.0000.aaaa.gggg
 1430  *            +000a.aaag.ggg0.0000
 1440  *             ------------------
 1450  *             (crchi)   (crclo)
 1460  *--------------------------------
 1470  NEW.CRC.BUFFER
 1480         CLC
```

```
1490        XCE
1500        REP #$30     M&X BOTH 16-BITS
1510        LDA ##$FFFF
1520        STA CRC      INITIALIZE CRC FOR BUFFER
1530        LDX ##BUFFER POINT TO BUFFER
1540  *--------------------------------
1550  .1    SEP #$20     CRC=aaaabbbbccccdddd, DATA=eeeeffff
1560        LDA CRC+1    aaaabbbb .eor. 0000aaaa = aaaagggg
1570        LSR
1580        LSR
1590        LSR
1600        LSR          0000aaaa
1610        EOR CRC+1    aaaabbbb
1620        XBA          AGXX
1630        LDA #0       AG00
1640        REP #$20
1650        LSR
1660        LSR
1670        LSR          000a.aaag.ggg0.0000
1680        STA TEMP
1690        LSR          0000.aaaa.gggg.0000
1700        EOR CRC      aaaa.bbbb.cccc.dddd = aaaa.gggg.kkkk.dddd
1710        XBA          kkkk.dddd.aaaa.gggg
1720        EOR TEMP     000a.aaag.ggg0.0000
1730        SEP #$20
1740        EOR 0,X        crchi.crclo
1750        REP #$20
1760        STA CRC
1770  *--------------------------------
1780        INX
1790        CPX ##LIMIT
1800        BCC .1
1810        XCE
1820        RTS
1830  *--------------------------------
1840  *   FIND BAD BIT BY BRUCE LOVE'S METHOD
1850  *--------------------------------
1860  DUMMY.CRC  .EQ $10,11
1870  *--------------------------------
1880  FIND.BAD.BIT
1890        JSR RECV     RECEIVE, COMPUTING NEW CRC
1900  *--------------------------------
1910        CLC
1920        XCE          ENTER NATIVE MODE
1930        REP #$30     X,M 16 BITS
1940        LDX ##$80F   X=BIT NUMBER
1950        LDA ##1      START DUMMY CRC IN A-REG
1960  .1    CMP CRC
1970        BEQ .2       ...FOUND BAD BIT!
1980        DEX          DECREMENT BIT NUMBER
1990        BMI .2       ...WENT TOO FAR, COULDN'T FIND BAD BIT
2000        ASL          SHIFT DUMY CRC
2010        BCC .1
2020        EOR ##$1021
```

```
2030        BCS .1       ...ALWAYS
2040  *--------------------------------
2050  .2    TXA          BIT NUMBER
2060        SEC
2070        XCE
2080        XBA
2090        JMP PRNTAX
2100  *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8602:DOS3.3:S.WILDCAT.EXEC.txt
==========================================================================

 1000  MON I
 1010  CALL -151
 1020  9D26:B2 B6
 1030  9D3E:DC A5
 1040  9FA8:CA
 1050  9FC5:A9 A0 2C A9 8D 6C 36 00
 1060  A710:CA
 1070  A186:AC 5F AA B9 1F 9D 48 B9
 1080      :1E 9D 48 60 EA
 1090  A56E:4C DD A5 A5 EC 20 CA 9F
 1100      :4C C5 9F EA
 1110  A5DD:AD 75 AA 85 EE 8E 75 AA
 1120      :A9 0D 4C AA A2 EA
 1130  A921:60 70
 1140  A929:60
 1150  AAE3:9A AD
 1160  AB10:C9 0E
 1170  A4F0:A9 A0 BE C8 B4 10 02 A9
 1180      :AA 4C CA 9F
 1190  A9FD:D2 2C 06 E0 30 03 4C 24
 1200      :ED 4C 1B E5
 1210  A021:EA EA EA
 1220  AA2C:C9 A0 F0 0C A0 A0 CC 76
 1230      :AA F0 03 4C C4 A6 C5 EC
 1240      :60 EA EA
 1250  AD98:20 84 A8 20 DC AB D0 57
 1260      :4C F4 AD EA EA 20 84 A8
 1270      :20 84 A8 20 38 AE
 1280  ADAE:20 2F AE A2 0C BD AE B3
 1290      :20 CA 9F CA D0 F7 20 69
 1300      :BA 20 2F AE 20 2F AE
 1310  ADC5:18 90 04 EA 20 84 A8 20
 1320      :11 B0 B0 5B A2 00 8E 9C
 1330      :B3 BD C6 B4 F0 51 30 48
 1340  ADDD:BD C8 B4 0A A0 07 0A B0
 1350      :03 88 D0 FA
 1360  ADE9:B9 A7 B3 85 EC A0 1E 84
 1370      :EF D0 4B
 1380  ADF4:20 84 A8 20 F7 AF 90 AF
 1390  ADFC:EA EA AC 9C B3 20 F0 A4
 1400      :20 71 A5
 1410  AE07:BE E7 B4 B9 E8 B4 20 FE
 1420      :A9 A0 07 84 24 AE 9C B3
 1430  AE17:BD C9 B4 20 DA B6 E8 C6
 1440      :EF D0 F5 20 2F AE
 1450  AE25:20 30 B2 90 A9 B0 A0 4C
 1460      :7F B3
 1470  AE2F:C6 EB D0 09 20 8D B7 F0
 1480      :F4 A9 15 85 EB 4C C8 9F
```

```
1490  AE3F:A5 EE C9 DE F0 07 20 2C
1500       :AA F0 B4 D0 D9
1510  AE4C:84 ED A0 01 C6 ED 30 D1
1520       :CA 88 D0 FC C8 E8 B9 75
1530       :AA C9 A0 F0 9D DD C9 B4
1540       :F0 F2 E8 D0 E7 EA
1550  BA69:86 44 86 45 A0 C8 B9 F2
1560       :B3 0A 90 06 E6 44 D0 F9
1570       :E6 45 D0 F5 88 D0 EF A6
1580       :44 A5 45 4C FE A9
1590  BA87:20 A8 FC C6 55 D0 82 A9
1600       :4F 85 55 4C C8 9F EA
1610  B6B3:20 A3 A2 20 8E BA 20 8C
1620       :A6 F0 14 C9 8D F0 F4 20
1630       :DA B6 A5 F1 20 87 BA AD
1640       :00 C0 10 EA 8D 10 C0 20
1650       :8D B7 D0 E2 4C FC A2
1660  B6DA:A8 10 08 C9 A0 B0 0E 24
1670       :EA 30 0A 46 32 46 32 29
1680       :3F 69 1F 49 E0
1690  B6EF:C9 E0 90 02 29 FF 20 CA
1700       :9F A9 FF 85 32 60
1710  B78D:20 0C FD C9 91 60
1720  B3AF:BD C5 C3 C1 D0 D3 A0 C5
1730       :C5 D2 C6 A0
1740  A884:A9 A0 85 EE 18 60
1750  A88A:49 4E 49 D4 4C 4F 41 C4
1760       :53 41 56 C5 52 55 CE 54
1770       :59 50 C5
1780  A89D:44 45 4C 45 54 C5 4C 4F
1790       :43 CB 55 4E 4C 4F 43 CB
1800  A8AD:43 4C 4F 53 C5 52 45 41
1810       :C4 45 58 45 C3 57 52 49
1820       :54 C5
1830  A8BF:44 49 D2
1840  9FFB:B9 8A A8
1850  9FED:59 8A A8
1860  48:04 N 3D0G
1870  NOMON I
```

```
===========================================================================
DOCUMENT :AAL-8602:DOS3.3:S.WILDCAT.txt
===========================================================================

 1000  *SAVE S.WILDCAT
 1010  *--------------------------------
 1020  CatLnCnt   .EQ $EB Catalog Linecount
 1030  FType      .EQ $EC Hold looked-up filetype
 1040  FName1     .EQ $EE Hold FNAME shell 1st char
 1050  CatNmLen   .EQ $ED CatName check-length=30
 1060  CatPtLen   .EQ $EF CatName print-length=30
 1070  *-----(1)----
 1080         .PH $9D26 In CMDTBL, command addresses,
 1090         .DA TYPE-1 change Integer CHAIN to TYPE
 1100         .PH $9D3E In CMDTBL, change address to
 1110         .DA NewDOCAT-1 new DOCAT in POSITION code
 1120  *-----(2)----
 1130         .PH $9FA8 In ECHO, change old COUT ref
 1140         .DA #$CA was JSR $9FC5 now JSR $9FCA
 1150         .PH $9FC5 Cleanup CDI COUT and CROUT
 1160  BlankOUT LDA #" " and add BLANK out routine
 1170         .DA #$2C fake BIT-NOP on fall-thru
 1180  CROUT  LDA #$8D  DOS vectored CROUT; same loc
 1190  COUT   JMP ($36) DOS vectored COUT; new loc
 1200         .PH $A710 In PRTERROR, change old COUT
 1210         .DA #$CA was JSR $9FC5 now JSR $9FCA
 1220  *-----(3)----
 1230         .PH $A186 Cleanup DOCMD; X=0 in NewDOCAT
 1240         LDY $AA5F CMDINDX
 1250         LDA $9D1F,Y CMDTBL+1; use Y instead of X
 1260         PHA
 1270         LDA $9D1E,Y CMDTBL
 1280         PHA
 1290         RTS
 1300         NOP
 1310  *-----(4)----
 1320         .PH $A56E Replace old DOCAT code:
 1330  OldDOCAT JMP NewDOCAT To allow for direct entry
 1340  PrtTYPE LDA FType Print looked-up filetype
 1350         JSR COUT and
 1360         JMP BlankOUT a blank
 1370         NOP
 1380  *-----(5)----
 1390         .PH $A5DD Replace old POSITION code:
 1400  NewDOCAT LDA $AA75 FNAME set by CATALOG command
 1410         STA FName1 save first byte, then zero
 1420         STX $AA75 to avoid buffer allocation
 1430         LDA #13 FM WildCAT Function Code
 1440         JMP $A2AA CMDHNDL2 routine, per usual
 1450         NOP
 1460  *-----(6)----
 1470         .PH $A921 DIR [string] [,Dn] [,Sn]
 1480         .DA #$60,$70 ->First comma: is NOT optional
```

```
        .PH $A929 CATALOG [string] <[,Dn] [,Sn]>
        .DA #$60 ->Must be CATALOG,D1 or DIR,D2
        .PH $AAE3 In FM function table, "borrow"
        .DA WildCAT-1 otherwise useless address
        .PH $AB10 Change range check from 13 for
        CMP #14 above now USEFULL address
*-----(7)----
        .PH $A4F0 Replace Integer CHAIN code
PrtLOCK LDA #" " blank=unlocked
        LDX $B4C8,Y Catalog Filetype entry
        BPL ToPrint
        LDA #"*" *=locked
ToPrint JMP COUT Print " " or "*" indicator
*-----(8)----
        .PH $A9FD Shorten NO BUFFER[S AVAILABLE]
        .AS -"R" to free 11 bytes for ToPrtDec:
ToPrtDec BIT $E006 Check which Basic...
        BMI ToInt Integer or
        JMP $ED24 Applesoft; use appropriate
ToInt JMP $E51B print decimal routine
*-----(9)----
        .PH $A021 Replace JSR ISBASRUN to allow
        NOP  ALL commands entered direct
        NOP  then error msg is redundant so
        NOP  ok to re-use msg space below
        .PH $AA2C Replace NOT DIRECT COMMAND msg
CkCAT CMP #" " If blank, do regular catalog
        BEQ ToRTS
        LDY #" " Must be single-char filetype
        CPY $AA76 FNAME+1, ie blank afterwards
        BEQ CkType if catalog by filetype; else
        JMP $A6C4 CSYNTAX error
CkType CMP FType Does filetype match?
ToRTS RTS
        NOP
        NOP
*----(10)----
        .PH $AD98
CATHNDLR JSR AllowENT Allow for non-CDI entry
WildCAT JSR $ABDC Init File Manager Workarea
        BNE ToRWVTOC
atADA0 JMP AlowVTOC Allow for non-CDI entry
atADA3 NOP  Allow for non-CDI entry and
        NOP  alignment
atADA5 JSR AllowENT Allow for non-CDI entry
atADA8 JSR AllowENT Allow for non-CDI entry
atADAB JSR InitCR Init Linecount; output C/R
*----(11)----
        JSR SKIPLN
        LDX #12
PrtFreSP LDA FreeMsg-1,X
        JSR COUT Print " FREE SPACE="
        DEX
        BNE PrtFreSP X=0 for PrtFreSP
```

```
       JSR FreeSpce Count & print free sectors
       JSR SKIPLN
       JSR SKIPLN
       CLC  Setup for RDNXTDIR to read
       BCC RDNXTDIR first sector; always branch
       NOP  alignment
atADC9 JSR AllowENT Allow non-CDI, non-FM entry
RDNXTDIR JSR $B011 RDDIRSEC
       BCS DONEXT2
       LDX #0
GTRKNUM STX $B39C DIRINDX
       LDA $B4C6,X Track part of T/S list
       BEQ DONEXT2 If End of Catalog, then exit
       BMI DONEXT If Deleted File, then skip it
       LDA $B4C8,X Catalog Filetype
       ASL  ;skip hi-bit LOCK/UNLOCK flag
       LDY #7
FindTYPE ASL
       BCS GotTYPE
       DEY
       BNE FindTYPE
GotTYPE LDA $B3A7,Y From filetype table,
       STA FType save looked-up filetype
       LDY #30 Check CatName length and
       STY CatPtLen Print CatName length
       BNE CkFNAME always BNE
*----(12)----
AlowVTOC JSR AllowENT Allow for non-CDI entry
ToRWVTOC JSR $AFF7 RWVTOC read VTOC
       BCC atADAB always; carry set=I/O ERROR
*----(13)----
       NOP  ;alignment
       NOP
PrtCAT LDY $B39C Restore Y from DIRINDX
       JSR PrtLOCK Print Lock indicator
       JSR PrtTYPE Print filetype and BlankOUT
       LDX $B4E7,Y Filesize
       LDA $B4E8,Y Filesize+1
       JSR ToPrtDec Print "true" filesize
       LDY #7 "Poke" CH with 7 to "tab"
       STY $24 over for filename spacing
       LDX $B39C Restore X from DIRINDX
PrtFN LDA $B4C9,X Print Catalog Filename
       JSR InvCOUT with optional conversions
       INX
       DEC CatPtLen CatName print length
       BNE PrtFN
       JSR SKIPLN
DONEXT JSR $B230 NXTDIREN...atAE25
       BCC GTRKNUM
       BCS RDNXTDIR
DONEXT2 JMP $B37F NOERROR....atAE2C
SKIPLN DEC CatLnCnt Linecount..atAE2F
       BNE ToCR If not zero, C/R & return
```

```
2570        JSR WaitCQ else wait for keypress
2580        BEQ DONEXT2 If Ctrl-Q, exit to NOERROR
2590  InitCR LDA #22-1 else setup for next 22 lines
2600        STA CatLnCnt in line count
2610  ToCR JMP CROUT DOS vectored C/R out
2620  *----(14)----
2630  CkFNAME LDA FName1 Holds FNAME first character
2640        CMP #"^" Wildcard string?
2650        BEQ DoWild yes...maybe
2660        JSR CkCAT Regular or by filetype?
2670        BEQ PrtCAT yes...else
2680        BNE DONEXT none of the above
2690  DoWild STY CatNmLen CatName length=30, for NotEQ
2700        LDY #1 Decr'd to 0; indexes FNAME
2710  NotEQ DEC CatNmLen Checked all 30 chars?
2720        BMI DONEXT Yes; no match, do next CatName
2730  BackDown DEX  Backdown to string match start
2740        DEY  Backdown to 0, ie. FNAME start
2750        BNE BackDown
2760  YesEQ INY  First Y=1, then on past "^"
2770        INX
2780        LDA $AA75,Y FNAME
2790        CMP #" " If blank then wildcard EOS and
2800        BEQ PrtCAT still =, so we have a match!
2810        CMP $B4C9,X FNAME = CatName?
2820        BEQ YesEQ
2830        INX  No, setup X to backdown 1 past
2840        BNE NotEQ string match start; always BNE
2850        NOP
2860  *----(15)----
2870        .PH $BA69 Catalog Free Space Patch
2880  FreeSpce STX $44 X=0
2890        STX $45 Init Free Sec Count var
2900        LDY #50*4 VTOC entries * entry length
2910  NxBitMap LDA $B3F2,Y BITMAP-1 in VTOC buffer
2920  CkFree ASL  ;shift hi-order bit into CARRY
2930        BCC CkMore In use, so check if any more
2940        INC $44 Incr free sector count
2950        BNE CkFree Zero means > 255, so
2960        INC $45 incr "page" part of word
2970  CkMore BNE CkFree More bits in same byte?
2980        DEY  decr index to next VTOC byte
2990        BNE NxBitMap All done?
3000        LDX $44 Yes, so setup count in X & A
3010        LDA $45 for decimal print via
3020        JMP ToPrtDec one of the BASICs
3030  *----(16)----
3040  WaitCk79 JSR $FCA8 Monitor WAIT routine
3050        DEC $55 Decr char cnt
3060        BNE $BA10 Fortuitous RTS; else fall thru
3070  InitLine LDA #79 TYPE prolog/setup
3080        STA $55 Init printer 80-col char cnt
3090        JMP CROUT
3100        NOP
```

```
*----(17)----
       .PH $B6B3
TYPE JSR $A2A3 DOS Open file
DoInitLn JSR InitLine Init char cnt & CROUT
ToRead JSR $A68C DOS Read char
       BEQ ToWaitCQ EOF maybe...Ctrl-Q quit?
       CMP #$8D Carriage return?
       BEQ DoInitLn Yes, handle immediately
       JSR InvCOUT Optional Ctrls & Hibit=0 INV
       LDA $F1 Applesoft SPEED=nn byte
       JSR WaitCk79 Wait SPEED; 79 chars yet?
       LDA $C000 Has a key been pressed?
       BPL ToRead No, read on
       STA $C010 Reset keyboard strobe
ToWaitCQ JSR WaitCQ Wait keypress, check Ctrl-Q?
       BNE ToRead If not Ctrl-Q, read on
       JMP $A2FC DOS Close, Deallocate, Exit
*----(18)----
InvCOUT TAY  If < $80, then hibit off
       BPL SetINV so set inverse flag & convert
       CMP #$A0 Ctrl-char?
       BCS CkLoCase No
       BIT $EA Usually, loc 234 contains 0:
       BMI CkLoCase POKE 234,255 skips conversion
SetINV LSR $32 Set Inverse by shifting 0 into
       LSR $32 INVFLG first 2 bits; set carry
       AND #$3F Turn off 1st 2 bits maps down
       ADC #$1F maps up into hibit-on part of
       EOR #$E0 upper-case screen-char range
CkLoCase CMP #$E0 Lower-case?
       BCC ToCOUT No; but POKE -18700,223 or
       AND #$FF B6F4:DF shifts l.c. to U.C.
ToCOUT JSR COUT DOS vectored COUT
       LDA #$FF
       STA $32 Set normal video; always
       RTS
*----(19)----
       .PH $B78D Wait keypress; check Ctrl-Q
WaitCQ JSR $FD0C Monitor RDKEY
       CMP #$91 Was it Ctrl-Q?
       RTS
*----(20)----
       .PH $B3AF Replace: DISK VOLUME inverted
FreeMsg .AS -"=ECAPS EERF " with FREE SPACE=
*----(21)----
       .PH $A884 Setup FName1 for "irregular"
AllowENT LDA #" " entry into CATALOG code
       STA FName1 Force blank at CkFNAME above
       CLC  For possible RDNXTDIR entry
       RTS
DOSCMDS .AT 'INIT' Move down DOSCMDS table and
       .AT 'LOAD' re-use the freed space above
       .AT 'SAVE'
       .AT 'RUN'
```

```
        .AT 'TYPE' was CHAIN
        .AT 'DELETE'
        .AT 'LOCK'
        .AT 'UNLOCK'
        .AT 'CLOSE'
        .AT 'READ'
        .AT 'EXEC'
        .AT 'WRITE'
 atA8BF .AT 'DIR' was POSITION; for CAT:43 41 D4
 *----(22)----
        .PH $9FFB In Command Interpreter PARSE
        LDA DOSCMDS,Y DOSCMDS table ref. was $A884
        .PH $9FED Only 2 references to DOSCMDS
        EOR DOSCMDS,Y DO THIS AFTER ABOVE CHANGES!
 *--------------------------------
```

================================================================
DOCUMENT :AAL-8602:DOS3.3:WILDCAT.EXEC.txt
================================================================

```
MON I
CALL -151
9D26:B2 B6
9D3E:DC A5
9FA8:CA
9FC5:A9 A0 2C A9 8D 6C 36 00
A710:CA
A186:AC 5F AA B9 1F 9D 48 B9
    :1E 9D 48 60 EA
A56E:4C DD A5 A5 EC 20 CA 9F
    :4C C5 9F EA
A5DD:AD 75 AA 85 EE 8E 75 AA
    :A9 0D 4C AA A2 EA
A921:60 70
A929:60
AAE3:9A AD
AB10:C9 0E
A4F0:A9 A0 BE C8 B4 10 02 A9
    :AA 4C CA 9F
A9FD:D2 2C 06 E0 30 03 4C 24
    :ED 4C 1B E5
A021:EA EA EA
AA2C:C9 A0 F0 0C A0 A0 CC 76
    :AA F0 03 4C C4 A6 C5 EC
    :60 EA EA
AD98:20 84 A8 20 DC AB D0 57
    :4C F4 AD EA EA 20 84 A8
    :20 84 A8 20 38 AE
ADAE:20 2F AE A2 0C BD AE B3
    :20 CA 9F CA D0 F7 20 69
    :BA 20 2F AE 20 2F AE
ADC5:18 90 04 EA 20 84 A8 20
    :11 B0 B0 5B A2 00 8E 9C
    :B3 BD C6 B4 F0 51 30 48
ADDD:BD C8 B4 0A A0 07 0A B0
    :03 88 D0 FA
ADE9:B9 A7 B3 85 EC A0 1E 84
    :EF D0 4B
ADF4:20 84 A8 20 F7 AF 90 AF
ADFC:EA EA AC 9C B3 20 F0 A4
    :20 71 A5
AE07:BE E7 B4 B9 E8 B4 20 FE
    :A9 A0 07 84 24 AE 9C B3
AE17:BD C9 B4 20 DA B6 E8 C6
    :EF D0 F5 20 2F AE
AE25:20 30 B2 90 A9 B0 A0 4C
    :7F B3
AE2F:C6 EB D0 09 20 8D B7 F0
    :F4 A9 15 85 EB 4C C8 9F
```

```
AE3F:A5 EE C9 DE F0 07 20 2C
    :AA F0 B4 D0 D9
AE4C:84 ED A0 01 C6 ED 30 D1
    :CA 88 D0 FC C8 E8 B9 75
    :AA C9 A0 F0 9D DD C9 B4
    :F0 F2 E8 D0 E7 EA
BA69:86 44 86 45 A0 C8 B9 F2
    :B3 0A 90 06 E6 44 D0 F9
    :E6 45 D0 F5 88 D0 EF A6
    :44 A5 45 4C FE A9
BA87:20 A8 FC C6 55 D0 82 A9
    :4F 85 55 4C C8 9F EA
B6B3:20 A3 A2 20 8E BA 20 8C
    :A6 F0 14 C9 8D F0 F4 20
    :DA B6 A5 F1 20 87 BA AD
    :00 C0 10 EA 8D 10 C0 20
    :8D B7 D0 E2 4C FC A2
B6DA:A8 10 08 C9 A0 B0 0E 24
    :EA 30 0A 46 32 46 32 29
    :3F 69 1F 49 E0
B6EF:C9 E0 90 02 29 FF 20 CA
    :9F A9 FF 85 32 60
B78D:20 0C FD C9 91 60
B3AF:BD C5 C3 C1 D0 D3 A0 C5
    :C5 D2 C6 A0
A884:A9 A0 85 EE 18 60
A88A:49 4E 49 D4 4C 4F 41 C4
    :53 41 56 C5 52 55 CE 54
    :59 50 C5
A89D:44 45 4C 45 54 C5 4C 4F
    :43 CB 55 4E 4C 4F 43 CB
A8AD:43 4C 4F 53 C5 52 45 41
    :C4 45 58 45 C3 57 52 49
    :54 C5
A8BF:44 49 D2
9FFB:B9 8A A8
9FED:59 8A A8
48:04 N 3D0G
NOMON I
```

```
==================================================================
DOCUMENT :AAL-8602:ProDOS:S.CRC.GENERATOR.txt
==================================================================

 1000  *SAVE S.CRC.GENERATOR
 1010  *--------------------------------
 1020  BUFFER .EQ $4000
 1030  LIMIT  .EQ $4102
 1040  *--------------------------------
 1050  CRC    .EQ $00,01
 1060  PNTR   .EQ $02,03
 1070  TEMP   .EQ $0A,0B
 1080  *--------------------------------
 1090  PRNTAX .EQ $F941
 1100  CROUT  .EQ $FD8E
 1110  *--------------------------------
 1120  *      SIMULATE SENDING A BUFFER-FULL
 1130  *--------------------------------
 1140  SEND
 1150         LDA #0       CLEAR CRC BYTES IN BUFFER
 1160         STA LIMIT-1
 1170         STA LIMIT-2
 1180         JSR NEW.CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1190         LDX CRC      STORE CRC INTO LAST 2 BYTES
 1200         LDA CRC+1
 1210         STX LIMIT-1
 1220         STA LIMIT-2
 1230         JSR PRNTAX   DISPLAY THE CRC
 1240         JMP CROUT    <RETURN> AND RETURN
 1250  *--------------------------------
 1260  *      SIMULATE RECEIVING A BUFFER-FULL
 1270  *--------------------------------
 1280  RECV
 1290         JSR NEW.CRC.BUFFER    COMPUTE CRC OF 258 BYTES
 1300         LDX CRC      DISPLAY CRC IN HEX
 1310         LDA CRC+1
 1320         JSR PRNTAX
 1330         JMP CROUT
 1340  *--------------------------------
 1350         .OP 65802
 1360  *--------------------------------
 1370  *      CRCH      CRCL      DATA
 1380  *   aaaa.bbbb.cccc.dddd.eeee.ffff
 1390  *  +0000.aaaa
 1400  *   ---------
 1410  *   aaaa.gggg
 1420  *            +gggg.0000.aaaa.gggg
 1430  *            +000a.aaag.ggg0.0000
 1440  *             -------------------
 1450  *             (crchi)   (crclo)
 1460  *--------------------------------
 1470  NEW.CRC.BUFFER
 1480         CLC
```

```
1490        XCE
1500        REP #$30     M&X BOTH 16-BITS
1510        LDA ##$FFFF
1520        STA CRC      INITIALIZE CRC FOR BUFFER
1530        LDX ##BUFFER POINT TO BUFFER
1540 *---------------------------------
1550 .1     SEP #$20     CRC=aaaabbbbccccdddd, DATA=eeeeffff
1560        LDA CRC+1    aaaabbbb .eor. 0000aaaa = aaaagggg
1570        LSR
1580        LSR
1590        LSR
1600        LSR          0000aaaa
1610        EOR CRC+1    aaaabbbb
1620        XBA          AGXX
1630        LDA #0       AG00
1640        REP #$20
1650        LSR
1660        LSR
1670        LSR          000a.aaag.ggg0.0000
1680        STA TEMP
1690        LSR          0000.aaaa.gggg.0000
1700        EOR CRC      aaaa.bbbb.cccc.dddd = aaaa.gggg.kkkk.dddd
1710        XBA          kkkk.dddd.aaaa.gggg
1720        EOR TEMP     000a.aaag.ggg0.0000
1730        SEP #$20
1740        EOR 0,X        crchi.crclo
1750        REP #$20
1760        STA CRC
1770 *---------------------------------
1780        INX
1790        CPX ##LIMIT
1800        BCC .1
1810        XCE
1820        RTS
1830 *---------------------------------
1840 *   FIND BAD BIT BY BRUCE LOVE'S METHOD
1850 *---------------------------------
1860 DUMMY.CRC  .EQ $10,11
1870 *---------------------------------
1880 FIND.BAD.BIT
1890        JSR RECV     RECEIVE, COMPUTING NEW CRC
1900 *---------------------------------
1910        CLC
1920        XCE          ENTER NATIVE MODE
1930        REP #$30     X,M 16 BITS
1940        LDX ##$80F   X=BIT NUMBER
1950        LDA ##1      START DUMMY CRC IN A-REG
1960 .1     CMP CRC
1970        BEQ .2       ...FOUND BAD BIT!
1980        DEX          DECREMENT BIT NUMBER
1990        BMI .2       ...WENT TOO FAR, COULDN'T FIND BAD BIT
2000        ASL          SHIFT DUMY CRC
2010        BCC .1
2020        EOR ##$1021
```

```
2030        BCS .1       ...ALWAYS
2040  *--------------------------------
2050  .2     TXA          BIT NUMBER
2060        SEC
2070        XCE
2080        XBA
2090        JMP PRNTAX
2100  *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8603:Articles:Boughner.Mult.txt
==============================================================================

Even Faster 65802 16x16 Multiply......Bob Sander-Cederlof

Bob Boughner, faithful reader from Yorktown, Virginia, decided that the challenge at the end of my article in the January 1986 AAL could not be ignored. He was able to slightly increase the speed of my 16x16 multiply subroutine for the 65802. After studying his code, I made a few more little changes and squeezed out even more cycles.

To see just how much faster the new subroutine is, I carefully counted the cycles, and then went back and did the same to January's subroutine. For some reason I got a new answer for January's program, slightly slower than published. Here are the results:

| | Minimum | Maximum | Average |
|---|---|---|---|
| January | 333 | 693 | 513 |
| New One | 321 | 633 | 477 |

The times include 6 cycles for a JSR to call the subroutine, and 6 cycles for the RTS to return. By putting the code in-line, even these 12 cycles could be eliminated. The so-called average time is merely the arithmetic average of the minimum and maximum times. The "real" average for random factors will be faster, because one or both of the INC instructions at lines 1350 and 1430 would be skipped. In fact, almost always at least one would be skipped, saving 48 cycles. Note also that if the factor in CAND is zero, the total time is only 45 cycles.

In counting cycles I assumed that the D-register, which tells the 65802 where the direct page is, has a low byte = 0. If it is non-zero, all of the references to CAND, PLIER, and PROD would require one more cycle.

The new subroutine is only 4 bytes longer than the January one. The new one uses the Y-register, while the old one did not. There are three tricks in the new code which save time. The first one is holding the multiplicand in the Y-register, so that TYA instructions can be used at lines 1310 and 1390. This saves 2 cycles each time, or a total of 32 cycles in the maximum case. The cost is the LDY CAND in line 1200, 4 cycles.

The second trick eliminates the CLC instruction before the multiplier is added in lines 1370-1430. The savings is 16 cycles maximum, and the cost is 8 cycles to set it up in lines 1120-1140 by inverting the high byte of the multiplier. This doesn't affect the average time any, but it does lower the maximum time.

The third trick is at lines 1280 and 1290. I saved 24 cycles by eliminating January's AND ##$0080 instruction here. The LDA PLIER-1 instruction picks up the low byte of the multiplier in the high byte of the A-register, allowing me to see what bit 7 of the multiplier is without any masking or shifting.

==============================================================================
DOCUMENT :AAL-8603:Articles:Disasm65816Plus.txt
==============================================================================

Add Smarts to 65816 Dis-Assembler...............Jim Poponoe

I found fascinating the article by Bob Sander-Cederlof in the March 1985 AAL, entitled "A Disassembler for the 65816". I purchased AAL Quarterly Disk 18 and tried it out for myself, watching 65802 instructions zip before my eyes.

But, whoa! Bob was correct in warning that his disassembler would not know whether immediate-mode instructions are two or three bytes long. Bob explained "only by executing the programming, and tracing it line-by-line, can we tell." A fully accurate disassembler for the 65816 would have to execute the equivalent of STEP and TRACE, following the logic flow of the program.

I wanted an easier, quick-and-dirty way to spiff up the output, one that would at least recognize simple, straightforward changes in the processor status. I reasoned that:

1) Interpretation of immediate-mode instructions depends on the state of E, M, and X bits in the status register.

2) E and C bits are exchangeable.

3) The disassembler must keep track of all four bits (C, E, X, and M) in order to disassemble immediate mode opcodes correctly.

4) The disassembler should also keep track of when the processor status is pushed onto or pulled off the stack.

My implementation assigns a memory location for the E-bit, and a small "stack" of 8 memory locations for the status register. One more memory location serves as the stack pointer. Here is the initialization code for these memory locations, replacing lines 1450-1480 in Bob's March 1985 listing:

    <<<<lines 1450-1486>>>>

I added a JSR TEST.OP.CODES line at 5865, to call some new code which looks for CLC, SEC, REP, SEP, PHP, PLP, and XCE instruc- tions. It adjusts the flags appropriately in response to these instructions. If the current opcode is none of the above, TEST.OP.CODES checks the status bits and the opcode to set up the correct immediate-mode length. If the opcode is an immediate mode operation on the A-register, and if E=0 and M=0, then 16-bit immediate will be disassembled. If the opcode is an immediate mode operation on the X- or Y-register, and if E=0 and X=0, then 16-bit immediate will be disassembled. Otherwise, any kind of immediate mode instruction will be disassembled with an 8-bit operand.

I tried the program on all the sample 65802 code I could find, and it was all disassembled correctly. Of course it is certainly possible to fool my program. The C-bit, and hence possibly the E-bit, can be changed in many other ways than by using the CLC and SEC instructions. The program flow is not followed, so it is possible than my emulation of the carry status and the XCE will not agree with what happens in some code. If you adhere to the "nice" standard of always using explicit SEC or CLC opcodes before an XCE opcode, the disassembler should stay in step perfectly.

When you type 800G to link in the disassembler (refer to Bob's article to know what I mean here) the status is initialized to E=C=M=X=1. This means normal 6502 mode. If you disassemble some code with XCE's in it, the status I keep will probably be left in some other mode. If you then try to disassemble some plain vanilla 6502 code, the immediate instructions may be disassembled with 16-bit operands. Just type 800G again to get back to normal.

By the way, in working with Bob's disassembler I discovered a typing error in his code. Line 3980 was originally >OXA TAY, and it should have been >OXA DEY. The hex listing in Bob's article showed $AF stored in $963; it really should be $89. Without this change, the DEY opcode disassembles as TAY!

The listing that follows has been extensively modified by Bob, based on my code I sent him last September. The lines are numbered to follow after the last line of the program on the quarterly disk.

<<<<<listing of lines 7060-7880>>>>

Further notes by Bob Sander-Cederlof:

Thanks, Jim! Your ideas were a big help! In looking back over my work, I noticed some more improvements.

R. F. O'Brien wrote us just this week with the news that he had found two bugs in the disassembler. One was the typing error at line 3980 which Jim noted above. But Robert found a second typo, at line 4960. ">OXB LDX" should be changed to ">OXB CPX". This changes the byte shown in the original article at $9BF from $19 to $0F.

I found a way to simplify the >ON macro, which speeds up assembly and shortens the listing. Replace lines 1220-1290 with the following:

<<<<<listing of lines 1220-1290>>>>

I also discovered that one kind of Apple monitor ROM did not have the RELADR subroutine, so I re-coded lines 6760-6950. Replace those lines with the following:

<<<<listing of lines 6760-6900>>>>>

One last item. I wrote a test routine to call the disassembler for every possible opcode from 00 to FF. Here it is:

<<<<listing of lines 7890 to end>>>>

==========================================================================
DOCUMENT :AAL-8603:Articles:Front.Page.txt
==========================================================================

$1.80
Volume 6 -- Issue 6                          March, 1986

In This Issue...



ES-CAPE

Whatever happened to the Extended S-C Applesoft Program Editor? That's a question we've heard more than a few times in the last year or two, and we finally have some kind of answer.

We got bogged down in producing Version 2.0 of the program.  The new printer control, Park and Join, and Applesoft and DOS command features are great.  The 40-column, STB-80, and //e versions came out just fine, but the Videx and Viewmaster versions stumped us.  The planned Renumber and Merge features never made it, and we couldn't settle on a mechanism for adding other utility programs.

Anyway, we've got a deal for you!  How's this for a package?:  ES-CAPE 1.0 Source and Object Code and manual, along with ES-CAPE 2.0 Source and Object Code and a manual supplement on disk.  That's all the source and object code for both versions of the program, for a total of only $50.00.  Registered owners of ES-CAPE 1.0 can purchase this new package for only $30.00.

New 65816 Book

There's another book coming along on programming the 658xx processors. This one is called "65816/65802 Assembly Language Programming", by Michael Fischer, published by Osborne/ McGraw-Hill as an addition to their Assembly Language Programming series, mostly by Lance Leventhal. Fischer's book is scheduled for May delivery, so we have ordered some copies and are beginning to accept orders.  Our price will be $18.00 + shipping.

==============================================================================
DOCUMENT :AAL-8603:Articles:PAL.Programmer.txt
==============================================================================

New Hardware for Programming PALs......Bob Sander-Cederlof

PALs (programmable array logic chips) are to logic circuitry as ROMs are to memory. Most of the new cards coming out these days contain one or more PALs. Engineers write logic equations, feed them into a PAL Assembler, and run the output to a PAL burner. The programmed PAL is then ready to use in a circuit. Until now, you had to buy a PAL development system, either stand-alone or perhaps interfaced to an IBM-alike.

But now, Dynatek Electronics has introduced a new board than slips nicely into an Apple slot for programming 20- and 24-pin PALs. The PALP-701A, for $245, programs 20-pin PALs. The PALP-702A handles both 20- and 24-pin chips, and can also blow the security fuse when you are ready for it. Both of them come with the PAL Assembler software.

Dynatek's PAL Assembler is compatible with Monolithic Memories PALASM. It creates a fuse plot from a PAL source file of Boolean equations. The fuse plot is then used by the PAL Programmer card via on-board firmware to program the PAL. The firmware on the Programmer card can also read un-protected PALs, and verify them. There is also a screen editor for creating, examining, and modifying a fuse plot.

Almost any Apple II system will do. You need at least 16K RAM to use the card, at least 48K and a disk drive to use the PAL Assembler. And who, these days, does not have at LEAST 48K?

If you design and build circuits, you ought to investigate this card. Call Jerry Wang at (312) 255-3469, or write to Dynatek Electronics, Inc., P. O. Box 1567, Arlington Heights, IL 60006. Tell him we sent you!

=====================================================================
DOCUMENT :AAL-8603:Articles:PDos.Franklines.txt
=====================================================================

Modifying ProDOS for Non-Standard ROMs...Bob Sander-Cederlof

We have published several times ways to defeat the ROM Checksummer that is executed during a ProDOS boot, so that owners of Franklin clones (or even real Apples with modified monitor ROMs) could use ProDOS-based software. See AALs of March and June, 1984.

Both of these previous articles are out of date now, because they apply to older versions of ProDOS than are current. What follows applies to Version 1.1.1 of ProDOS.

There are two problems with getting ProDOS to boot on a non-standard machine. The first is the ROM Checksummer. This subroutine starts at $267C in Version 1.1.1, and is only called from $25EE. The code is purposely weird, designed to look like it is NOT checking the ROMs. It also has apparently purposeful side effects. Here is a listing of the subroutine:

<<<<listing of ROM.CHECKSUMMER>>>>

The pointer at $0A,0B was set up to point to $FB09 using very sneaky code at $248A. Location $2674 initially contains a 0, and $2677 contains an 8. Only the bytes from $FB09 through $FB10 are checksummed. Truthfully, "checksummed" is not the correct word.

The wizards who put ProDOS together figured out a fancy function which changes the 64 bits from $FB09 through $FB10 into the value $75. Their function does this whether your ROMs are the original monitor ROM from 1977-78, the Autostart ROM, the original //e ROM, or any other standard Apple ROM. The values in $FB09-FB10 are not the same in all cases, but the function result is always $75. However, a Franklin ROM does not produce $75. Probably a BASIS also gives a different result, and other clones. Once $75 is obtained, further slippery code changes the value to $00.

The original Apple II ROM has executable code at $FB09, and in hex it is this: B0 A2 20 4A FF 38 B0 9E. All other Apple monitor ROMs have an ASCII string at $FB09. The string is either "APPLE ][" or "Apple ][". Notice that the "AND #$DF" in the checksummer strips out the upper/lower case bit, making both ASCII strings the same.

I wrote a test program to print out all the intermediate values during the "Checksummer's" operation. Here are the results, for both kinds of ROMs.

```
Original ROM           Later ROMs
LDA AND ADC STA ROL    LDA   AND ADC STA ROL
B0  90  00  90  20     C1    C1  00  C1  82
A2  82  20  A2  44     D0/F0 D0  82  52  A5
```

```
20  00  44  44  88     D0/F0 D0  A5  75  EB
4A  4A  88  D2  A4     CC/EC CC  EB  B7  6F
FF  DF  A4  83  07     C5/E5 C5  6F  34  69
38  18  07  1F  3E     A0    80  69  E9  D2
B0  90  3E  C3  9C     DD    DD  D2  AF  5F
9E  9E  9C  3A  75     DB    DB  5F  3A  75
```

I don't understand why this code gives the same result, but I see it does. Now, dear readers, tell me how anyone ever figured out what sequence of operations would produce the same result using these two different sets of eight bytes, and yet produce a different result for clones! If you understand it, please explain it to me!

By the way, here is a listing of my test program:

<<<<listing of test program>>>>

The checksummer can be defeated. The best way, preserving the various side effects, is to change the byte at $269F from $03 to $00. This changes the BNE to an effective no-operation, because it will branch to the next instruction regardless of the status. Another way to get the same result is to store $EA at both $269E and $269F. Still another way is to change the "LDA #0" at $26A3,4 to "LDA $0C" (A5 0C), so that either case gives the same result.

If it thinks it is in a valid Apple computer, the checksummer returns a value in the A-register which is non-zero, obtained from location $0C. The value at $0C has been previously set by looking at other locations in the ROM, trying to tell which version is there. Part of this code is at $2402 and following, and part is at $2047 and following. The byte at $0C will eventually become the Machine ID byte at $BF98 in the System Global Page, so it also gets some bits telling how much RAM is available, and whether an 80-column card and a clock card are found.

If you have a non-standard Apple or a clone the bytes which are checked to determine which kind of ROM you have may give an illegal result. The following table shows the bytes checked, and the resulting values for $0C. The values in parentheses are not ever checked, but I included them for completeness. The value in $0C will be further modified to indicate the amount of RAM found and the presence of a clock card.

| Version | FBB3 | FB1E | FBC0 | FBBF | $0C |
|---|---|---|---|---|---|
| Original Apple II | 38 | (AD) | (60) | (2F) | 00 |
| Autostart, II Plus | EA | AD | (EA) | (EA) | 40 |
| Original //e | 06 | (AD) | EA | (C1) | 80 |
| Enhanced //e | 06 | (AD) | E0 | (00) | 80 |
| DEBUG //e | 06 | (AD) | E1 | (00) | 80 |
| Original //c | 06 | (4C) | 00 | FF | 88 |
| //c Unidisk 3.5 | 06 | (4C) | 00 | 00 | 88 |
| /// Emulating II | EA | 8A | (??) | (??) | C0 |

By the way, ProDOS 1.1.1 will not allow booting by an Apple /// emulating a II Plus, possibly because the standard emulator only emulates a 48K machine.

I have no idea what a clone would have in those four locations, but chances are it would be different. You should probably try to fool ProDOS into thinking you are in a II Plus, because most clones are II Plus clones. This means you should somehow change the ID procedures so that the result in $0C is a value of $40. One way to do this is change the code at $2402 and following like this:

```
      Standard                      Change to

2402- A9 00    LDA #0         2402- A9 40    LDA #$40
2404- 85 0C    STA $0C        2404- 4C 2E 24 JMP $242E
2406- A3 B3 FB LDX $FBB3
```

If your clone or modified ROM is a //e, change $2402 to LDA #$80 instead.

You may also need to modify the code at $2047 and following. If you are trying to fool ProDOS into thinking you are an Apple II Plus or //e, and have already made the change described above, change $2047-9 like this:

```
      Standard                     Change to

2047- AE B3 FB LDX $FBB3    2047- 4C 6D 20 JMP $206D
```

No doubt future versions of ProDOS will make provision for clones and modified ROMs even more difficult. And there are always the further problems encountered by usage of the ROMs from BASIC.SYSTEM and the ProDOS Kernel and whatever application program is running.

I am intrigued about seeing what the minimum amount of code is that can distinguish between the four legal varieties of ROM for ProDOS. I notice from the table above that I can identify the four types and weed out the ///emulator by the following simple code at $2402:

```
       LDA $FBB3
       ORA $FB1E
       LDX #3
  .1   CMP TABLE.1,X
       BEQ .2
       DEX
       BPL .1
       SEC
       RTS
*
TABLE.1  .HS BD.EF.AF.4E
TABLE.2  .HS 00.40.80.88
*
  .2   LDA TABLE.2,X
       JMP $242E
```

With this code installed, all the code from $2047-$206C is not needed, and the JMP $206E should be installed at $2047. The new code at $2402 fits in the existing space with room to spare. Can you do it with even shorter code?

```
================================================================================
DOCUMENT :AAL-8603:Articles:Putney.Mul8x8.txt
================================================================================
```

Fastest 6502 Multiplication Yet................Charles Putney
Shankill, Dublin, Ireland

Here is an 8x8 multiply routine that will blow your socks off! The maximum time, including both a calling JSR and a returning RTS, is only 66 cycles! The minimum is 60 cycles, and most factors will multiply in 63 cycles. Recall that the fastest time in Bob S-C's January 1986 AAL article for a 6502 was 132 cycles. My new one is twice as fast!

As with most fast routines, there is a trade off in memory space. My program uses 1024 bytes of lookup tables. This isn't so bad if you really need or want a 2:1 speed advantage.

My routine is based on the fact that:

```
       4 * X * Y = (X+Y)^2 - (X-Y)^2
```

I got this idea from an article in EDN Magazine by Arch D. Robison (October 13, 1983, pages 263-4). His routine used the fact that:

```
       2 * X * Y = X^2 + Y^2 - (X-Y)^2
```

Robison's method requires three dips into the lookup tables. Formulated to the same method for passing parameters, his method takes either 74 or 77 cycles. Here is my rendition of his method:

<<<<listing of Robison's program>>>>

The entries in the two tables (SQL and SQH) are the squares of the numbers from 0 to 255, divided by two. The low bytes are in the SQL table, and the high bytes are in SQH. Dividing by two throws away an important bit for odd factors, but lines 1160-1170 compensate for the loss.

I looked for a way to add fewer table entries together and came upon the sum^2 - diff^2. Since the sum can be as large as 255+255=510, I need twice as much table space. Lest you despair of typing in such a large table, let me offer an Applesoft program which will write a text file of the source code for the table:

<<<listing of Applesoft source creator>>>>

My tables contain the squares divided by four. I can hear you saying, "Wait a minute! You can't just divide by four and truncate!" Well, even squares are all multiples of four; odd squares are all multiples of four with a remainder = 1. The sum of two numbers and the difference of the same numbers are either both even or both odd. Therefore, we never lose anything by throwing away our truncated 1.

The number of cycles my MULT8 takes depends on the values of the two factors. You call MULT8 with one factor in the A-register and the other in the X-register. If (A) is less than (X), it takes an extra 3 cycles to perform a complement operation. If the sum of the factors is greater than 255, add another three cycles. To summarize,

| | A>=X | A<X |
|---|---|---|
| sum<256 | 60 | 63 |
| sum>255 | 63 | 66 |

Just for fun, I also wrote a program to generate the square/4 tables. This takes less time than loading the tables from disk, so it could mean faster booting for some hi-resolution game program that needs super-fast multiplications. It is in lines 1560-2100 below.

The origin I used in my program is meant just to allow me to test it. I wrote an Applesoft program to call TEST at $6000 (CALL 24576). The program POKEd two factors at $FA and $FB, called TEST, and then checked the result at the same two locations. If you want to use MULT8, you should just assemble it along with the rest of your program, without any special origin. You should make sure that the tables start on an even page boundary, or it will cost you up to 8 cycles extra for indexing across a page boundary.

================================================================
DOCUMENT :AAL-8603:Articles:Transwarp.Rvw.txt
================================================================

Review of Applied Engineering Transwarp.....Bob Sander-Cederlof

We reviewed the M-c-T SpeedDemon accelerator card in AAL of July 1985. At the time the price was $295 from the manufacturer or $199 through Call APPLE. We recently received a promotion sent to software publishers offering wholesale prices if we would advertise the SpeedDemon in conjunction with our software. The suggested price is now $249. (We notice that at least one game publisher took them up on the offer.)

Now Applied Engineering has released their new accelerator card, the Transwarp. Their price is $279 with a 65C02 installed, and an optional upgrade to a fast 65802 for an additional $89. The higher price is probably well justified by the features.

Transwarp includes 256K of high-speed RAM on the card. This compares to 64K on the Titan Accelerator, and a 4K cache on the SpeedDemon. Transwarp will run with the SWYFT card installed, while the others apparently will not.

Transwarp's 256K RAM is effectively divided into four 64K banks. When you power-up your Apple with Transwarp installed, all of the ROM from $D000 through $FFFF is copied into one of the high-speed RAM banks. The rest of this bank is not used. A second bank is used in place of the motherboard RAM. The third and fourth banks are used in place of the first and second banks of AUXMEM, if you have a RAM card such as RAMWORKS installed in the AUX slot. If you have a large RAMWORKS in the auxiliary slot of a //e, any additional banks beyond two will still be usable but at "only" 1 MHz.

When you write data to one of the screen areas (any address $400-$BFF or $2000-$5FFF), the data is "written through" to the motherboard RAM. (The video hardware in the Apple requires that the screen data be in motherboard RAM.) When you read from any of these addresses, the data will be read from the fast Transwarp RAM.

Transwarp keeps track of the state of all the AUXMEM soft switches, as well as the RAMWORKS bank register. All reads from any memory that is supported in the Transwarp RAM will be done at full speed. Reads from and writes to any address in the range $C000-$CFFF will slow down to 1 MHz for one cycle.

There are 16 dip switches on the card, allowing you to configure for most environments. Seven switches indicate which slots must execute code at 1 MHz. Slots designated by switches will slow down the processor for about 1/2 second after any access to either the slot ROM or the slot registers. An Apple disk Controller must run at the slow speed, while most other slots can run faster. Some I/O cards, especially serial cards, must run at slow speed due to internal software-controlled timing. The Transwarp's switches are much more flexible than the SpeedDemon's system of always slowing down for slot 6 and using jumpers to allow a slowdown for slots 4 and 5.

Another seven switches let you indicate which slots (1-7) have RAM cards installed. The two remaining switches let you select the initial speed of the Transwarp card. You can select a default speed of 3.58 MHz, 1.7 MHz, or 1 MHz. This is the speed the card runs at when you power up. You might like the 1.7 MHz speed for making your game software just a LITTLE faster.

Once the Transwarp has taken over, you can switch back and forth between the default speed and 1 MHz by storing either 0 (default speed) or 1 (1 MHz) into $C074. In BASIC this would be POKE to -16268 or 49268 of either 0 or 1.

If you POKE a value of 3 to $C074, Transwarp will be shut down completely; the motherboard processor will take over when you hit CTRL-RESET. In order to turn Transwarp back on, you have to turn the computer off and back on again with the power switch. You also have the option of disabling Transwarp during the power-on cycle, by typing the ESCAPE key within a couple of seconds after turning on the computer.

Transwarp has a 4K EPROM on-board with startup and self-test firmware. Naturally, I disassembled the code to see how it all works. The self-test is initiated by typing a "0" or "9" during the first two seconds. The test checks for the type of processor installed (65C02 or 65802), measures the speed, tests bank switching, and tests RAM. If you are in a //e, you can hold down the Open-Apple key to keep it looping through the speed test.

Transwarp measures its own speed by counting how many cycles it takes for the Vertical Blanking Signal to pass by. This signal is not available on the II or II Plus, so no speed information is tested on the older machines.

We tested Transwarp doing various jobs such as assembling, word processing, and spreadsheet-ing. Everything worked, no glitches, and a lot faster. The speedup factor depends on the amount of disk I/O, screen I/O, and so on. Nothing runs with a full 3.5 or 3.6 speed increase, not even a short timing loop. The very highest factor I could coax out of my board was about 3.3, on a timing loop running at $C00. This loop included a large number of STA instructions, on purpose. When I moved the program to $800, so that the STA instructions were storing into the range slowed down to 1MHz (between $400 and $BFF), the loop only ran 2.0 times faster under Transwarp than under a normal 1 MHz processor.

Why do the advertisements for accelerators claim a 3.6 or larger speedup factor? I think they are rounding up the clock speed of 3.579... to 3.6, and likewise rounding down the Apple's clock speed from 1.023 to 1. That is not the way the IRS likes you to do math....

The actual ratio of the two clock speeds is exactly 3.5, but the mist does not entirely clear yet.

Remember that the Apple stretches one cycle out of every 65 by an amount equal to one cycle of the 7MHz signal. See chapter 3 of Jim Sather's "Understanding the Apple //e" for details. This means the normal Apple runs a hair slower than the clock rate. But also remember that dynamic RAM needs refreshing from time to time. The refresh of the 256K RAM on the Transwarp card occurs once out of every 16 Apple phase 0 (1MHz) clock cycles. During each 16th 1MHz cycle, the Transwarp slows down to 1MHz. This means that in the time a normal Apple would execute 16 clock cycles, the full-speed Transwarp will execute 53 clock cycles. If not for the long refresh cycle, Transwarp would execute 56 cycles during 16 phase 0 cycles. Now 53 divided by 16 is 3.3125, showing that the maximum speedup factor for Transwarp is 3.3125. I don't know for certain, but the Titan Accelerator II probably has the same characteristic. If so, they both run at a full 3.5 times faster for 15 micro- seconds, slow down for one microsecond, and then take off again.

The SpeedDemon, on the other hand, can run at a full 3.5 times faster for somewhat longer bursts. If every byte needed is in the SpeedDemon cache memory (static RAM, needing no refresh), execution should proceed at 3.5 times normal Apple speed. Normal programs, however, which are long enough to make us worry about speed, will never be entirely inside the cache. In all comparison tests of real software, Transwarp is faster than either SpeedDemon or Titan. SpeedDemon loses due to its cache, and Titan loses because it does not speed up any accesses to AUXMEM.

The S-C Word Processor increased its speed by about 3.2 for compute-bound operations like searching. Interestingly, an operation that is limited by screen output, like inserting characters from the yank buffer, showed almost no increase in speed. In THE Spreadsheet (MagiCalc) the acceleration factor was about 3.1-3.3, running in a II+ with a Viewmaster 80-column card. Our mailing label system, written mostly in Applesoft, showed a pretty consistent 3.3 speedup. Programs which involve disk I/O will not speed up as much, because the disk still spins at the same 300 rpm.

All in all, we think the Transwarp is a good investment: you get a quality product at a reasonable price which significantly enhances the performance of your computer.

==============================================================================
DOCUMENT :AAL-8603:Articles:V6N6.IIX.Rumors.txt
==============================================================================

Some More Rumors

Electronics magazine printed a brief news item about a second source for 65816 chips. Western Design has signed up a lot of licensees to make these chips, but none of them are in production as of this month. Electronics says VLSI Technology Inc., of San Jose, California, is projecting prices in the $10 range for volume purchases. When? Target is to start selling sample quantities next summer. Meanwhile, volume prices are in the $35 range from Western Design Center. The single-unit price is still about $100.

The parts we are selling are the 65C802 from Western Design Center. Our price to you is $50 each. These are normally spec'd at 2MHz, but sometimes we get 4MHz parts at the same price, when they are out of the slower ones. Either speed works equally well in an Apple motherboard, but you need the 4MHz chip to use in a Transwarp accelerator card.

Rumors continue to ricochet around the club newsletter circuit about the possible configuration of the new Apple II (usually called the //x). Most rumor sources agree now that the //x will use a 65C816. We sure HOPE so! One source said he looks for an 8MHz clock. We doubt that, because current projections are for 8MHz chips becoming available about 1st quarter 1987. And the RAM for 8MHz operation would be far too expensive. My guess we will see either 2MHz or 3.58MHz.

Most are now including a SCSI port in their list of features, since the Macintosh Plus has one. Some are talking about a smaller set of normal slots, supplemented by some new super-slots having more signals available. There are reportedly a number of different versions of the //x already in existence, seeded around. If that is true, it could be than no one (even inside Apple) yet knows what the REAL //x will be.

================================================================================
DOCUMENT :AAL-8603:Articles:Weishaars.Book.txt
================================================================================

New Book by Tom Weishaar, reviewed by Bob Sander-Cederlof

A little over a year ago, just before he started the "Open-Apple" newsletter, Tom wrote a book.  Info Books has just released it, called "Your Best Interest: A Money Book for the Computer Age."  It's not about Apple assembly language, but I cannot resist telling you about it anyway!

The book is about interest rates -- how to understand them, how to calculate them, how they affect you.  It was written for people who know how to use a spreadsheet program.  All the hard math and books of tables are replaced your favorite calc-alike.

If you remember Tom's DOSTalk column from the much-missed pages of Softalk Magazine, or are familiar with his current Open-Apple newsletter, you know that what he writes is easy to read, fun to read, and WORTH READING.

Seven fascinating chapters lead you to an understanding of how financial transactions really work.  He starts with simple percentage calculations, at a level your Junior High children can follow.  If you think that is starting too simply, try explaining percentages to YOUR children!  But he keeps going....

Have you thought about buying a house recently?  Tom shows you how to figure the true cost of an adjustable-rate mortgage, how to compare different financing schemes, and how to protect your money.  You'll learn about the tricks money lenders sometimes use to take advantage of unwary investors and borrowers.  And all is tied to spreadsheet models you can put into your Apple.  I wish I had only known how to do these things when I bought a pickup truck last summer.  Or leased a copying machine three years ago.  And when we bought some land in the country....

The book is 160 pages slim (172 counting everything), only $9.95 at your favorite book store.  And worth a trip!  Or call Gerald Rafferty at Info Books, (213) 470-6786.  Or write to them at P. O. Box 1018, Santa Monica, CA 90406.

==============================================================================
DOCUMENT :AAL-8603:Articles:Which.Processor.txt
==============================================================================

Which Processor Am I In?.......................Jim Poponoe

One of the first programs I wrote after receiving my 65802 chip was one which tells me which microprocessor is in my Apple. Since the 65C02 has instructions not in the 6502, and since the 65802 has all of those and still more, it is possible to tell which is which.

The instructions in the 65802 (or 65816) which are not in the 65C02 are all "no-operation" opcodes in the 65C02. The same is not true for the un-implemented codes in the 6502! Bob S-C detailed what all the un-implemented 6502 opcodes do in the March 1981 issue of AAL. Some of them do really exotic things, but some are in fact NOPs. $80 is a two-byte NOP in the 6502, but a Branch Always (BRA) in the 65C02 and 658xx. Therefore, the BRA opcode can be used to distinguish between the 6502 and higher versions.

The XBA instruction ($EB) is a one-byte no-operation in the 65C02. In the 658xx it exchanges the low and high bytes of the 16-bit A-register. Therefore it can be used to distinguish between the 65C02 and the 658xx processors.

The following program will print out either "6502", "65C02", or "65802" depending on which it finds. A few more tests could distinguish the Rockwell 65C02, which has four opcodes beyond those in 65C02s made by other manufacturers. And a few more might distinguish between a 65802 in my motherboard and a 65816 running in a co-processor card. I'll leave those for interested readers to try.

==============================================================================
DOCUMENT :AAL-8603:DOS3.3:Boughner.Mult.txt
==============================================================================

```
1000        .OP 65802
1010 *SAVE BOUGHNER'S MULTIPLY
1020 *--------------------------------
1030 *   CONTRIBUTED BY BOB BOUGHNER
1040 *   MODIFIED A LITTLE MORE BY BOB S-C
1050 *--------------------------------
1060 CAND   .EQ 0,1
1070 PLIER  .EQ 2,3
1080 PROD   .EQ 4,5,6,7
1090 *--------------------------------
1100 MUL.FASTER.YET.16X16.65802
1110        LDX #8       WILL LOOP 8 TIMES
1120        LDA PLIER+1  INVERT HIGH BYTE
1130        EOR #$FF     TO SAVE "CLC" IN LOOP
1140        STA PLIER+1
1150        CLC
1160        XCE          ENTER "NATIVE" MODE
1170        REP #$30     16-BITS BOTH X & M
1180        STZ PROD     CLEAR PRODUCT
1190        STZ PROD+2
1200        LDY CAND     MULTIPLICAND IN Y-REG
1210        BNE .2       ...NON-ZERO, START LOOP
1220        XCE          ...ZERO, EXIT NOW
1230        RTS
1240 *--------------------------------
1250 .1     ASL PROD     DOUBLE THE PRODUCT
1260        ROL PROD+2
1270 *--------------------------------
1280 .2     LDA PLIER-1  GET LOW BYTE IN A(15-8)
1290        BPL .3       ...ORIG. BIT=0, DON'T ADD
1300        CLC
1310        TYA          ...ORIG. BIT=1, ADD 'CAND
1320        ADC PROD
1330        STA PROD
1340        BCC .3
1350        INC PROD+2   ADD CARRY TO HI-16
1360 *--------------------------------
1370 .3     ASL PLIER    SHIFT MULTIPIER, GET HI-BIT
1380        BCS .4       ...ORIG. BIT=0, DON'T ADD
1390        TYA          ...ORIG. BIT=1, ADD 'CAND
1400        ADC PROD+1   ADD TO MIDDLE OF PRODUCT
1410        STA PROD+1
1420        BCC .4
1430        INC PROD+3   (NEVER BOTHERS PROD+4)
1440 *--------------------------------
1450 .4     DEX
1460        BNE .1
1470        SEC
1480        XCE
```

```
1490        RTS
1500 *--------------------------------
1510   .LIF
```

==============================================================================
DOCUMENT :AAL-8603:DOS3.3:Creat.SqTbl.Src.txt
==============================================================================

( DTC removed -- lots of garbage characters )

```
==============================================================================
DOCUMENT :AAL-8603:DOS3.3:Putney.Fst.8x8.txt
==============================================================================

 1000  *SAVE ROBISON'S FAST 8X8
 1010  *--------------------------------
 1020  *   MODIFIED FROM ORIGINAL PROGRAM
 1030  *   BY ARCH D. ROBISON, BURROUGHS CORP.
 1040  *      EDN, OCTOBER 13, 1983.
 1050  *--------------------------------
 1060  *   ENTER WITH (A)=MULTIPLIER # 1
 1070  *              (X)=MULTIPLIER #2
 1080  *   EXIT WITH (A)=PRODUCT HI BYTE
 1090  *             (X)=PRODUCT LO BYTE
 1100  *--------------------------------
 1110  PROD   .EQ $06      PRODUCT TEMP OF M1*M2 (LOW BYTE)
 1120  M2     .EQ $07      TEMP FOR M2 SAVE
 1130  *--------------------------------
 1140  MULT8  TAY          SAVE M1 IN Y
 1150         STX M2       SAVE M2
 1160         AND M2       CHECK IF BOTH FACTORS ARE ODD
 1170         LSR          SET CARRY <--> BOTH ODD
 1180         LDA SQL,X    ADD (X*X)/2 AND (Y*Y)/2
 1190         ADC SQL,Y
 1200         STA PROD     SAVE LO BYTE OF PRODUCT
 1210         LDA SQH,X
 1220         ADC SQH,Y
 1230         TAX          SAVE HI BYTE OF PRODUCT
 1240         TYA          GET M1 BACK
 1250         SEC
 1260         SBC M2       FIND M1 - M2
 1270         BCS .1       M1 >= M2, CONTINUE
 1280         SBC #0       M1 < M2, FORM 2'S COMPLEMENT
 1290         EOR #$FF
 1300  .1     TAY          USE ABS(M1-M2) AS INDEX
 1310         LDA PROD        TO FIND SQUARE/2 IN TABLE
 1320         SBC SQL,Y    NOW SUBTRACT (X-Y)*(X-Y)
 1330         STA PROD     SAVE LO BYTE OF RESULT
 1340         TXA          HI BYTE FROM PREVIOUS SUM
 1350         SBC SQH,Y
 1360         LDX PROD     LO BYTE OF FINAL PRODUCT
 1370         RTS
 1380  *--------------------------------
 1390         .OR $900     PAGE BOUNDARY TO SAVE MAX 6 CYCLES
 1400  *--------------------------------
 1410  SQL
 1420   .DA #0,#0,#2,#4,#8,#12,#18,#24
 1430   .DA #32,#40,#50,#60,#72,#84,#98,#112
 1440   .DA #128,#144,#162,#180,#200,#220,#242,#264
 1450   .DA #288,#312,#338,#364,#392,#420,#450,#480
 1460   .DA #512,#544,#578,#612,#648,#684,#722,#760
 1470   .DA #800,#840,#882,#924,#968,#1012,#1058,#1104
 1480   .DA #1152,#1200,#1250,#1300,#1352,#1404,#1458,#1512
```

```
1490   .DA #1568,#1624,#1682,#1740,#1800,#1860,#1922,#1984
1500   .DA #2048,#2112,#2178,#2244,#2312,#2380,#2450,#2520
1510   .DA #2592,#2664,#2738,#2812,#2888,#2964,#3042,#3120
1520   .DA #3200,#3280,#3362,#3444,#3528,#3612,#3698,#3784
1530   .DA #3872,#3960,#4050,#4140,#4232,#4324,#4418,#4512
1540   .DA #4608,#4704,#4802,#4900,#5000,#5100,#5202,#5304
1550   .DA #5408,#5512,#5618,#5724,#5832,#5940,#6050,#6160
1560   .DA #6272,#6384,#6498,#6612,#6728,#6844,#6962,#7080
1570   .DA #7200,#7320,#7442,#7564,#7688,#7812,#7938,#8064
1580   .DA #8192,#8320,#8450,#8580,#8712,#8844,#8978,#9112
1590   .DA #9248,#9384,#9522,#9660,#9800,#9940,#10082,#10224
1600   .DA #10368,#10512,#10658,#10804,#10952,#11100,#11250,#11400
1610   .DA #11552,#11704,#11858,#12012,#12168,#12324,#12482,#12640
1620   .DA #12800,#12960,#13122,#13284,#13448,#13612,#13778,#13944
1630   .DA #14112,#14280,#14450,#14620,#14792,#14964,#15138,#15312
1640   .DA #15488,#15664,#15842,#16020,#16200,#16380,#16562,#16744
1650   .DA #16928,#17112,#17298,#17484,#17672,#17860,#18050,#18240
1660   .DA #18432,#18624,#18818,#19012,#19208,#19404,#19602,#19800
1670   .DA #20000,#20200,#20402,#20604,#20808,#21012,#21218,#21424
1680   .DA #21632,#21840,#22050,#22260,#22472,#22684,#22898,#23112
1690   .DA #23328,#23544,#23762,#23980,#24200,#24420,#24642,#24864
1700   .DA #25088,#25312,#25538,#25764,#25992,#26220,#26450,#26680
1710   .DA #26912,#27144,#27378,#27612,#27848,#28084,#28322,#28560
1720   .DA #28800,#29040,#29282,#29524,#29768,#30012,#30258,#30504
1730   .DA #30752,#31000,#31250,#31500,#31752,#32004,#32258,#32512
1740  SQH
1750   .DA /0,/0,/2,/4,/8,/12,/18,/24
1760   .DA /32,/40,/50,/60,/72,/84,/98,/112
1770   .DA /128,/144,/162,/180,/200,/220,/242,/264
1780   .DA /288,/312,/338,/364,/392,/420,/450,/480
1790   .DA /512,/544,/578,/612,/648,/684,/722,/760
1800   .DA /800,/840,/882,/924,/968,/1012,/1058,/1104
1810   .DA /1152,/1200,/1250,/1300,/1352,/1404,/1458,/1512
1820   .DA /1568,/1624,/1682,/1740,/1800,/1860,/1922,/1984
1830   .DA /2048,/2112,/2178,/2244,/2312,/2380,/2450,/2520
1840   .DA /2592,/2664,/2738,/2812,/2888,/2964,/3042,/3120
1850   .DA /3200,/3280,/3362,/3444,/3528,/3612,/3698,/3784
1860   .DA /3872,/3960,/4050,/4140,/4232,/4324,/4418,/4512
1870   .DA /4608,/4704,/4802,/4900,/5000,/5100,/5202,/5304
1880   .DA /5408,/5512,/5618,/5724,/5832,/5940,/6050,/6160
1890   .DA /6272,/6384,/6498,/6612,/6728,/6844,/6962,/7080
1900   .DA /7200,/7320,/7442,/7564,/7688,/7812,/7938,/8064
1910   .DA /8192,/8320,/8450,/8580,/8712,/8844,/8978,/9112
1920   .DA /9248,/9384,/9522,/9660,/9800,/9940,/10082,/10224
1930   .DA /10368,/10512,/10658,/10804,/10952,/11100,/11250,/11400
1940   .DA /11552,/11704,/11858,/12012,/12168,/12324,/12482,/12640
1950   .DA /12800,/12960,/13122,/13284,/13448,/13612,/13778,/13944
1960   .DA /14112,/14280,/14450,/14620,/14792,/14964,/15138,/15312
1970   .DA /15488,/15664,/15842,/16020,/16200,/16380,/16562,/16744
1980   .DA /16928,/17112,/17298,/17484,/17672,/17860,/18050,/18240
1990   .DA /18432,/18624,/18818,/19012,/19208,/19404,/19602,/19800
2000   .DA /20000,/20200,/20402,/20604,/20808,/21012,/21218,/21424
2010   .DA /21632,/21840,/22050,/22260,/22472,/22684,/22898,/23112
2020   .DA /23328,/23544,/23762,/23980,/24200,/24420,/24642,/24864
```

```
2030   .DA /25088,/25312,/25538,/25764,/25992,/26220,/26450,/26680
2040   .DA /26912,/27144,/27378,/27612,/27848,/28084,/28322,/28560
2050   .DA /28800,/29040,/29282,/29524,/29768,/30012,/30258,/30504
2060   .DA /30752,/31000,/31250,/31500,/31752,/32004,/32258,/32512
```

```
==============================================================================
DOCUMENT :AAL-8603:DOS3.3:Putney.Fstr.8x8.txt
==============================================================================

 1000  *SAVE PUTNEY'S FASTER 8X8
 1010  *--------------------------------
 1020  *      ULTRA-FAST 8 X 8 MULTIPLY
 1030  *--------------------------------
 1040  *   ENTER WITH (A)=MULTIPLIER # 1
 1050  *              (X)=MULTIPLIER #2
 1060  *   EXIT WITH (A)=PRODUCT HI BYTE
 1070  *             (X)=PRODUCT LO BYTE
 1080  *--------------------------------
 1090  *   TIMING DATA
 1100  *      MINIMUM TIME = 54 CYCLES
 1110  *      MAXIMUM TIME = 60 CYCLES
 1120  *      AVERAGE TIME = 57 CYCLES
 1130  *--------------------------------
 1140  PROD   .EQ $06      PRODUCT TEMP OF M1*M2 (LOW BYTE)
 1150  M2     .EQ $07      TEMP FOR M2 SAVE
 1160  *--------------------------------
 1170         .OR $6000    SAFE PLACE
 1180  *--------------------------------
 1190  *   TEST FOR APPLESOFT DRIVER
 1200  *--------------------------------
 1210  TEST   LDA $FA      LOAD ACC AND X SO BASIC CAN TEST
 1220         LDX $FB
 1230         JSR MULT8
 1240         STX $FA      NOW BASIC CAN CHECK ACC AND X
 1250         STA $FB
 1260         RTS
 1270  *--------------------------------
 1280  MULT8  TAY          SAVE M1 IN Y
 1290         STX M2       SAVE M2
 1300         SEC          SET CARRY FOR SUBTRACT
 1310         SBC M2       FIND DIFFERENCE
 1320         BCS .1       WAS M1 > M2 ?
 1330         EOR #$FF     INVERT IT
 1340         ADC #$01     AND ADD 1
 1350  .1     TAX          USE ABS(M1-M2) AS INDEX
 1360         CLC
 1370         TYA          GET M1 BACK
 1380         ADC M2       FIND M1 + M2
 1390         TAY          USE M1+M2 AS INDEX
 1400         BCC .2       M1+M2 < 255 ?
 1410         LDA SQL+256,Y     FIND SUM SQUARED LOW IF > 255
 1420         SBC SQL,X         SUBTRACT DIFF SQUARED
 1430         STA PROD          SAVE IN PRODUCT
 1440         LDA SQH+256,Y     HI BYTE
 1450         SBC SQH,X
 1460         LDX PROD     GET PROD LOW IN X
 1470         RTS          DONE
 1480  .2     SEC          SET CARRY FOR SUBTRACT
```

```
1490        LDA SQL,Y    FIND SUM OF SQUARES LOW IF < 255
1500        SBC SQL,X    SUBTRACT DIFF SQUARED
1510        STA PROD     SAVE IN PRODUCT
1520        LDA SQH,Y    HI BYTE
1530        SBC SQH,X
1540        LDX PROD     GET PROD LOW IN X
1550        RTS
1560 *--------------------------------
1570 *   PROGRAM TO CREATE A TABLE OF SQUARES/4
1580 *--------------------------------
1590 LOTP   .EQ 0,1
1600 HITP   .EQ 2,3
1610 *--------------------------------
1620 SQUARE LDY #0
1630        STY LOTP
1640        STY HITP
1650        STY SQ
1660        STY SQ+1
1670        STY SQ+2
1680        STY DELTA+1
1690        STY DELTA+2
1700        STY $6800
1710        STY $6A00
1720        INY
1730        LDA #$40
1740        STA DELTA
1750        LDA /$6800
1760        STA LOTP+1
1770        LDA /$6A00
1780        STA HITP+1
1790        LDX #1
1800 *--------------------------------
1810 .1     CLC
1820        LDA DELTA
1830        ADC SQ
1840        STA SQ
1850        LDA DELTA+1
1860        ADC SQ+1
1870        STA SQ+1
1880        STA (LOTP),Y
1890        LDA DELTA+2
1900        ADC SQ+2
1910        STA SQ+2
1920        STA (HITP),Y
1930 *--------------------------------
1940        LDA DELTA
1950        ADC #$80
1960        STA DELTA
1970        BCC .2
1980        INC DELTA+1
1990        BNE .2
2000        INC DELTA+2
2010 .2     INY
2020        BNE .1
```

```
 2030          INC LOTP+1
 2040          INC HITP+1
 2050          DEX
 2060          BPL .1
 2070          RTS
 2080  *--------------------------------
 2090  DELTA   .BS 3
 2100  SQ      .BS 3
 2110  *--------------------------------
 2120  *   TABLE OF SQUARES/4 FROM 0 TO 511
 2130  *--------------------------------
 2140          .BS *+$FF/$100*$100-*  KEEP TABLES ALIGNED ON PAGE
BOUNDARY
 2150  *--------------------------------
 2160  SQL     .DA #0,#0,#1,#2,#4,#6,#9,#12
 2170          .DA #16,#20,#25,#30,#36,#42,#49,#56
 2180          .DA #64,#72,#81,#90,#100,#110,#121,#132
 2190          .DA #144,#156,#169,#182,#196,#210,#225,#240
 2200    .LIF
 2210          .DA #256,#272,#289,#306,#324,#342,#361,#380
 2220          .DA #400,#420,#441,#462,#484,#506,#529,#552
 2230          .DA #576,#600,#625,#650,#676,#702,#729,#756
 2240          .DA #784,#812,#841,#870,#900,#930,#961,#992
 2250          .DA #1024,#1056,#1089,#1122,#1156,#1190,#1225,#1260
 2260          .DA #1296,#1332,#1369,#1406,#1444,#1482,#1521,#1560
 2270          .DA #1600,#1640,#1681,#1722,#1764,#1806,#1849,#1892
 2280          .DA #1936,#1980,#2025,#2070,#2116,#2162,#2209,#2256
 2290          .DA #2304,#2352,#2401,#2450,#2500,#2550,#2601,#2652
 2300          .DA #2704,#2756,#2809,#2862,#2916,#2970,#3025,#3080
 2310          .DA #3136,#3192,#3249,#3306,#3364,#3422,#3481,#3540
 2320          .DA #3600,#3660,#3721,#3782,#3844,#3906,#3969,#4032
 2330          .DA #4096,#4160,#4225,#4290,#4356,#4422,#4489,#4556
 2340          .DA #4624,#4692,#4761,#4830,#4900,#4970,#5041,#5112
 2350          .DA #5184,#5256,#5329,#5402,#5476,#5550,#5625,#5700
 2360          .DA #5776,#5852,#5929,#6006,#6084,#6162,#6241,#6320
 2370          .DA #6400,#6480,#6561,#6642,#6724,#6806,#6889,#6972
 2380          .DA #7056,#7140,#7225,#7310,#7396,#7482,#7569,#7656
 2390          .DA #7744,#7832,#7921,#8010,#8100,#8190,#8281,#8372
 2400          .DA #8464,#8556,#8649,#8742,#8836,#8930,#9025,#9120
 2410          .DA #9216,#9312,#9409,#9506,#9604,#9702,#9801,#9900
 2420          .DA
#10000,#10100,#10201,#10302,#10404,#10506,#10609,#10712
 2430          .DA
#10816,#10920,#11025,#11130,#11236,#11342,#11449,#11556
 2440          .DA
#11664,#11772,#11881,#11990,#12100,#12210,#12321,#12432
 2450          .DA
#12544,#12656,#12769,#12882,#12996,#13110,#13225,#13340
 2460          .DA
#13456,#13572,#13689,#13806,#13924,#14042,#14161,#14280
 2470          .DA
#14400,#14520,#14641,#14762,#14884,#15006,#15129,#15252
 2480          .DA
#15376,#15500,#15625,#15750,#15876,#16002,#16129,#16256
```

```
 2490         .DA
#16384,#16512,#16641,#16770,#16900,#17030,#17161,#17292
 2500         .DA
#17424,#17556,#17689,#17822,#17956,#18090,#18225,#18360
 2510         .DA
#18496,#18632,#18769,#18906,#19044,#19182,#19321,#19460
 2520         .DA
#19600,#19740,#19881,#20022,#20164,#20306,#20449,#20592
 2530         .DA
#20736,#20880,#21025,#21170,#21316,#21462,#21609,#21756
 2540         .DA
#21904,#22052,#22201,#22350,#22500,#22650,#22801,#22952
 2550         .DA
#23104,#23256,#23409,#23562,#23716,#23870,#24025,#24180
 2560         .DA
#24336,#24492,#24649,#24806,#24964,#25122,#25281,#25440
 2570        .DA
#25600,#25760,#25921,#26082,#26244,#26406,#26569,#26732
 2580        .DA
#26896,#27060,#27225,#27390,#27556,#27722,#27889,#28056
 2590        .DA
#28224,#28392,#28561,#28730,#28900,#29070,#29241,#29412
 2600        .DA
#29584,#29756,#29929,#30102,#30276,#30450,#30625,#30800
 2610        .DA
#30976,#31152,#31329,#31506,#31684,#31862,#32041,#32220
 2620        .DA
#32400,#32580,#32761,#32942,#33124,#33306,#33489,#33672
 2630        .DA
#33856,#34040,#34225,#34410,#34596,#34782,#34969,#35156
 2640        .DA
#35344,#35532,#35721,#35910,#36100,#36290,#36481,#36672
 2650        .DA
#36864,#37056,#37249,#37442,#37636,#37830,#38025,#38220
 2660        .DA
#38416,#38612,#38809,#39006,#39204,#39402,#39601,#39800
 2670        .DA
#40000,#40200,#40401,#40602,#40804,#41006,#41209,#41412
 2680        .DA
#41616,#41820,#42025,#42230,#42436,#42642,#42849,#43056
 2690        .DA
#43264,#43472,#43681,#43890,#44100,#44310,#44521,#44732
 2700        .DA
#44944,#45156,#45369,#45582,#45796,#46010,#46225,#46440
 2710        .DA
#46656,#46872,#47089,#47306,#47524,#47742,#47961,#48180
 2720        .DA
#48400,#48620,#48841,#49062,#49284,#49506,#49729,#49952
 2730        .DA
#50176,#50400,#50625,#50850,#51076,#51302,#51529,#51756
 2740        .DA
#51984,#52212,#52441,#52670,#52900,#53130,#53361,#53592
 2750        .DA
#53824,#54056,#54289,#54522,#54756,#54990,#55225,#55460
```

```
 2760        .DA
#55696,#55932,#56169,#56406,#56644,#56882,#57121,#57360
 2770        .DA
#57600,#57840,#58081,#58322,#58564,#58806,#59049,#59292
 2780        .DA
#59536,#59780,#60025,#60270,#60516,#60762,#61009,#61256
 2790        .DA
#61504,#61752,#62001,#62250,#62500,#62750,#63001,#63252
 2800        .DA
#63504,#63756,#64009,#64262,#64516,#64770,#65025,#65280
 2810  *--------------------------------
 2820     .LIST ON
 2830  SQH    .DA /0,/0,/1,/2,/4,/6,/9,/12
 2840         .DA /16,/20,/25,/30,/36,/42,/49,/56
 2850         .DA /64,/72,/81,/90,/100,/110,/121,/132
 2860    .LIST OFF
 2870         .DA /144,/156,/169,/182,/196,/210,/225,/240
 2880         .DA /256,/272,/289,/306,/324,/342,/361,/380
 2890         .DA /400,/420,/441,/462,/484,/506,/529,/552
 2900         .DA /576,/600,/625,/650,/676,/702,/729,/756
 2910         .DA /784,/812,/841,/870,/900,/930,/961,/992
 2920         .DA /1024,/1056,/1089,/1122,/1156,/1190,/1225,/1260
 2930         .DA /1296,/1332,/1369,/1406,/1444,/1482,/1521,/1560
 2940         .DA /1600,/1640,/1681,/1722,/1764,/1806,/1849,/1892
 2950         .DA /1936,/1980,/2025,/2070,/2116,/2162,/2209,/2256
 2960         .DA /2304,/2352,/2401,/2450,/2500,/2550,/2601,/2652
 2970         .DA /2704,/2756,/2809,/2862,/2916,/2970,/3025,/3080
 2980         .DA /3136,/3192,/3249,/3306,/3364,/3422,/3481,/3540
 2990         .DA /3600,/3660,/3721,/3782,/3844,/3906,/3969,/4032
 3000         .DA /4096,/4160,/4225,/4290,/4356,/4422,/4489,/4556
 3010         .DA /4624,/4692,/4761,/4830,/4900,/4970,/5041,/5112
 3020         .DA /5184,/5256,/5329,/5402,/5476,/5550,/5625,/5700
 3030         .DA /5776,/5852,/5929,/6006,/6084,/6162,/6241,/6320
 3040         .DA /6400,/6480,/6561,/6642,/6724,/6806,/6889,/6972
 3050         .DA /7056,/7140,/7225,/7310,/7396,/7482,/7569,/7656
 3060         .DA /7744,/7832,/7921,/8010,/8100,/8190,/8281,/8372
 3070         .DA /8464,/8556,/8649,/8742,/8836,/8930,/9025,/9120
 3080         .DA /9216,/9312,/9409,/9506,/9604,/9702,/9801,/9900
 3090         .DA
/10000,/10100,/10201,/10302,/10404,/10506,/10609,/10712
 3100         .DA
/10816,/10920,/11025,/11130,/11236,/11342,/11449,/11556
 3110         .DA
/11664,/11772,/11881,/11990,/12100,/12210,/12321,/12432
 3120         .DA
/12544,/12656,/12769,/12882,/12996,/13110,/13225,/13340
 3130         .DA
/13456,/13572,/13689,/13806,/13924,/14042,/14161,/14280
 3140         .DA
/14400,/14520,/14641,/14762,/14884,/15006,/15129,/15252
 3150         .DA
/15376,/15500,/15625,/15750,/15876,/16002,/16129,/16256
 3160         .DA
/16384,/16512,/16641,/16770,/16900,/17030,/17161,/17292
```

```
 3170         .DA
/17424,/17556,/17689,/17822,/17956,/18090,/18225,/18360
 3180         .DA
/18496,/18632,/18769,/18906,/19044,/19182,/19321,/19460
 3190         .DA
/19600,/19740,/19881,/20022,/20164,/20306,/20449,/20592
 3200         .DA
/20736,/20880,/21025,/21170,/21316,/21462,/21609,/21756
 3210         .DA
/21904,/22052,/22201,/22350,/22500,/22650,/22801,/22952
 3220         .DA
/23104,/23256,/23409,/23562,/23716,/23870,/24025,/24180
 3230         .DA
/24336,/24492,/24649,/24806,/24964,/25122,/25281,/25440
 3240        .DA
/25600,/25760,/25921,/26082,/26244,/26406,/26569,/26732
 3250        .DA
/26896,/27060,/27225,/27390,/27556,/27722,/27889,/28056
 3260        .DA
/28224,/28392,/28561,/28730,/28900,/29070,/29241,/29412
 3270        .DA
/29584,/29756,/29929,/30102,/30276,/30450,/30625,/30800
 3280        .DA
/30976,/31152,/31329,/31506,/31684,/31862,/32041,/32220
 3290        .DA
/32400,/32580,/32761,/32942,/33124,/33306,/33489,/33672
 3300        .DA
/33856,/34040,/34225,/34410,/34596,/34782,/34969,/35156
 3310        .DA
/35344,/35532,/35721,/35910,/36100,/36290,/36481,/36672
 3320        .DA
/36864,/37056,/37249,/37442,/37636,/37830,/38025,/38220
 3330        .DA
/38416,/38612,/38809,/39006,/39204,/39402,/39601,/39800
 3340        .DA
/40000,/40200,/40401,/40602,/40804,/41006,/41209,/41412
 3350        .DA
/41616,/41820,/42025,/42230,/42436,/42642,/42849,/43056
 3360        .DA
/43264,/43472,/43681,/43890,/44100,/44310,/44521,/44732
 3370        .DA
/44944,/45156,/45369,/45582,/45796,/46010,/46225,/46440
 3380        .DA
/46656,/46872,/47089,/47306,/47524,/47742,/47961,/48180
 3390        .DA
/48400,/48620,/48841,/49062,/49284,/49506,/49729,/49952
 3400        .DA
/50176,/50400,/50625,/50850,/51076,/51302,/51529,/51756
 3410        .DA
/51984,/52212,/52441,/52670,/52900,/53130,/53361,/53592
 3420        .DA
/53824,/54056,/54289,/54522,/54756,/54990,/55225,/55460
 3430        .DA
/55696,/55932,/56169,/56406,/56644,/56882,/57121,/57360
```

```
 3440        .DA
/57600,/57840,/58081,/58322,/58564,/58806,/59049,/59292
 3450        .DA
/59536,/59780,/60025,/60270,/60516,/60762,/61009,/61256
 3460    .LIST ON
 3470        .DA
/61504,/61752,/62001,/62250,/62500,/62750,/63001,/63252
 3480        .DA
/63504,/63756,/64009,/64262,/64516,/64770,/65025,/65280
 3490  *--------------------------------
 3500    .LIF
```

```
=============================================================================
DOCUMENT :AAL-8603:DOS3.3:S.Which.CPU.txt
=============================================================================

 1000  *SAVE S.WHICH PROCESSOR
 1010         .OP 65802
 1020  *--------------------------------
 1030  PRBYTE .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  *--------------------------------
 1060  WHICH.PROCESSOR
 1070         LDA #$65
 1080         JSR PRBYTE
 1090         BRA .1
 1100         JMP .2
 1110  .1     LDA #"8"
 1120         XBA
 1130         LDA #"C"
 1140         XBA
 1150         JSR COUT
 1160  .2     LDA #$02
 1170         JMP PRBYTE
 1180  *--------------------------------
```

```
===========================================================
DOCUMENT :AAL-8603:ProDOS:BOUGHNERS.MULT.txt
===========================================================
```

```
 1000         .OP 65802
 1010  *SAVE BOUGHNERS.MULT
 1020  *--------------------------------
 1030  *   CONTRIBUTED BY BOB BOUGHNER
 1040  *   MODIFIED A LITTLE MORE BY BOB S-C
 1050  *--------------------------------
 1060  CAND   .EQ 0,1
 1070  PLIER  .EQ 2,3
 1080  PROD   .EQ 4,5,6,7
 1090  *--------------------------------
 1100  MUL.FASTER.YET.16X16.65802
 1110         LDX #8       WILL LOOP 8 TIMES
 1120         LDA PLIER+1  INVERT HIGH BYTE
 1130         EOR #$FF     TO SAVE "CLC" IN LOOP
 1140         STA PLIER+1
 1150         CLC
 1160         XCE          ENTER "NATIVE" MODE
 1170         REP #$30     16-BITS BOTH X & M
 1180         STZ PROD     CLEAR PRODUCT
 1190         STZ PROD+2
 1200         LDY CAND     MULTIPLICAND IN Y-REG
 1210         BNE .2       ...NON-ZERO, START LOOP
 1220         XCE          ...ZERO, EXIT NOW
 1230         RTS
 1240  *--------------------------------
 1250  .1     ASL PROD     DOUBLE THE PRODUCT
 1260         ROL PROD+2
 1270  *--------------------------------
 1280  .2     LDA PLIER-1  GET LOW BYTE IN A(15-8)
 1290         BPL .3       ...ORIG. BIT=0, DON'T ADD
 1300         CLC
 1310         TYA          ...ORIG. BIT=1, ADD 'CAND
 1320         ADC PROD
 1330         STA PROD
 1340         BCC .3
 1350         INC PROD+2   ADD CARRY TO HI-16
 1360  *--------------------------------
 1370  .3     ASL PLIER    SHIFT MULTIPIER, GET HI-BIT
 1380         BCS .4       ...ORIG. BIT=0, DON'T ADD
 1390         TYA          ...ORIG. BIT=1, ADD 'CAND
 1400         ADC PROD+1   ADD TO MIDDLE OF PRODUCT
 1410         STA PROD+1
 1420         BCC .4
 1430         INC PROD+3   (NEVER BOTHERS PROD+4)
 1440  *--------------------------------
 1450  .4     DEX
 1460         BNE .1
 1470         SEC
 1480         XCE
```

```
1490        RTS
1500  *---------------------------------
1510    .LIF
```

==============================================================
DOCUMENT :AAL-8603:ProDOS:CHECKSUMMER.txt
==============================================================

```
 1000  *SAVE CHECKSUMMER
 1010  *--------------------------------
 1020         .OR $267C    POSITION IN PRODOS SYSTEM FILE
 1030  *--------------------------------
 1040  CHECKSUMMER
 1050         CLC
 1060         LDY $2674    (GETS A VALUE 0)
 1070  .1     LDA ($0A),Y  GETS (FB09...FB10)
 1080         AND #$DF     STRIP OFF LOWER CASE BIT
 1090         ADC $2674    ACCUMULATE SHIFTED SUM
 1100         STA $2674
 1110         ROL $2674    SHIFT RESULT, CARRY INTO BIT 0
 1120         INY
 1130         CPY $2677    DO IT 8 TIMES
 1140         BNE .1
 1150         TYA          A = Y = 8
 1160         ASL          FORM $80 BY SHIFTING
 1170         ASL
 1180         ASL
 1190         ASL
 1200         TAY          $80 TO Y FOR LATER TRICK
 1210         EOR $2674    MERGE WITH PREVIOUS "SUM"
 1220         ADC #11      FORM $00 FOR VALID ROMS
 1230         BNE .2       ...NOT A VALID ROM
 1240         LDA $0C      GET MACHINE ID BYTE
 1250         RTS
 1260  .2     LDA #0       SIGNAL INVALIDITY
 1270         RTS
```

===============================================================================
DOCUMENT :AAL-8603:ProDOS:CREATE.SQUARE.T.txt
===============================================================================

( DTC removed -- lots of garbage characters )

```
==================================================================
DOCUMENT :AAL-8603:ProDOS:PUTNEYS.8X8.txt
==================================================================

 1000  *SAVE PUTNEYS.8X8
 1010  *--------------------------------
 1020  *      ULTRA-FAST 8 X 8 MULTIPLY
 1030  *--------------------------------
 1040  *   ENTER WITH (A)=MULTIPLIER # 1
 1050  *              (X)=MULTIPLIER #2
 1060  *   EXIT WITH (A)=PRODUCT HI BYTE
 1070  *             (X)=PRODUCT LO BYTE
 1080  *--------------------------------
 1090  *   TIMING DATA
 1100  *      MINIMUM TIME = 54 CYCLES
 1110  *      MAXIMUM TIME = 60 CYCLES
 1120  *      AVERAGE TIME = 57 CYCLES
 1130  *--------------------------------
 1140  PROD   .EQ $06      PRODUCT TEMP OF M1*M2 (LOW BYTE)
 1150  M2     .EQ $07      TEMP FOR M2 SAVE
 1160  *--------------------------------
 1170         .OR $6000    SAFE PLACE
 1180  *--------------------------------
 1190  *   TEST FOR APPLESOFT DRIVER
 1200  *--------------------------------
 1210  TEST   LDA $FA      LOAD ACC AND X SO BASIC CAN TEST
 1220         LDX $FB
 1230         JSR MULT8
 1240         STX $FA      NOW BASIC CAN CHECK ACC AND X
 1250         STA $FB
 1260         RTS
 1270  *--------------------------------
 1280  MULT8  TAY          SAVE M1 IN Y
 1290         STX M2       SAVE M2
 1300         SEC          SET CARRY FOR SUBTRACT
 1310         SBC M2       FIND DIFFERENCE
 1320         BCS .1       WAS M1 > M2 ?
 1330         EOR #$FF     INVERT IT
 1340         ADC #$01     AND ADD 1
 1350  .1     TAX          USE ABS(M1-M2) AS INDEX
 1360         CLC
 1370         TYA          GET M1 BACK
 1380         ADC M2       FIND M1 + M2
 1390         TAY          USE M1+M2 AS INDEX
 1400         BCC .2       M1+M2 < 255 ?
 1410         LDA SQL+256,Y     FIND SUM SQUARED LOW IF > 255
 1420         SBC SQL,X         SUBTRACT DIFF SQUARED
 1430         STA PROD          SAVE IN PRODUCT
 1440         LDA SQH+256,Y     HI BYTE
 1450         SBC SQH,X
 1460         LDX PROD     GET PROD LOW IN X
 1470         RTS          DONE
 1480  .2     SEC          SET CARRY FOR SUBTRACT
 1490         LDA SQL,Y    FIND SUM OF SQUARES LOW IF < 255
 1500         SBC SQL,X    SUBTRACT DIFF SQUARED
 1510         STA PROD     SAVE IN PRODUCT
 1520         LDA SQH,Y    HI BYTE
 1530         SBC SQH,X
```

```
1540        LDX PROD     GET PROD LOW IN X
1550        RTS
1560 *--------------------------------
1570 *   PROGRAM TO CREATE A TABLE OF SQUARES/4
1580 *--------------------------------
1590 LOTP   .EQ 0,1
1600 HITP   .EQ 2,3
1610 *--------------------------------
1620 SQUARE LDY #0
1630        STY LOTP
1640        STY HITP
1650        STY SQ
1660        STY SQ+1
1670        STY SQ+2
1680        STY DELTA+1
1690        STY DELTA+2
1700        STY $6800
1710        STY $6A00
1720        INY
1730        LDA #$40
1740        STA DELTA
1750        LDA /$6800
1760        STA LOTP+1
1770        LDA /$6A00
1780        STA HITP+1
1790        LDX #1
1800 *--------------------------------
1810 .1     CLC
1820        LDA DELTA
1830        ADC SQ
1840        STA SQ
1850        LDA DELTA+1
1860        ADC SQ+1
1870        STA SQ+1
1880        STA (LOTP),Y
1890        LDA DELTA+2
1900        ADC SQ+2
1910        STA SQ+2
1920        STA (HITP),Y
1930 *--------------------------------
1940        LDA DELTA
1950        ADC #$80
1960        STA DELTA
1970        BCC .2
1980        INC DELTA+1
1990        BNE .2
2000        INC DELTA+2
2010 .2     INY
2020        BNE .1
2030        INC LOTP+1
2040        INC HITP+1
2050        DEX
2060        BPL .1
2070        RTS
2080 *--------------------------------
2090 DELTA  .BS 3
2100 SQ     .BS 3
2110 *--------------------------------
2120 *   TABLE OF SQUARES/4 FROM 0 TO 511
```

```
*-------------------------------
       .BS *+$FF/$100*$100-*  KEEP TABLES ALIGNED ON PAGE BOUNDARY
*-------------------------------
SQL    .DA #0,#0,#1,#2,#4,#6,#9,#12
       .DA #16,#20,#25,#30,#36,#42,#49,#56
       .DA #64,#72,#81,#90,#100,#110,#121,#132
       .DA #144,#156,#169,#182,#196,#210,#225,#240
  .LIF
       .DA #256,#272,#289,#306,#324,#342,#361,#380
       .DA #400,#420,#441,#462,#484,#506,#529,#552
       .DA #576,#600,#625,#650,#676,#702,#729,#756
       .DA #784,#812,#841,#870,#900,#930,#961,#992
       .DA #1024,#1056,#1089,#1122,#1156,#1190,#1225,#1260
       .DA #1296,#1332,#1369,#1406,#1444,#1482,#1521,#1560
       .DA #1600,#1640,#1681,#1722,#1764,#1806,#1849,#1892
       .DA #1936,#1980,#2025,#2070,#2116,#2162,#2209,#2256
       .DA #2304,#2352,#2401,#2450,#2500,#2550,#2601,#2652
       .DA #2704,#2756,#2809,#2862,#2916,#2970,#3025,#3080
       .DA #3136,#3192,#3249,#3306,#3364,#3422,#3481,#3540
       .DA #3600,#3660,#3721,#3782,#3844,#3906,#3969,#4032
       .DA #4096,#4160,#4225,#4290,#4356,#4422,#4489,#4556
       .DA #4624,#4692,#4761,#4830,#4900,#4970,#5041,#5112
       .DA #5184,#5256,#5329,#5402,#5476,#5550,#5625,#5700
       .DA #5776,#5852,#5929,#6006,#6084,#6162,#6241,#6320
       .DA #6400,#6480,#6561,#6642,#6724,#6806,#6889,#6972
       .DA #7056,#7140,#7225,#7310,#7396,#7482,#7569,#7656
       .DA #7744,#7832,#7921,#8010,#8100,#8190,#8281,#8372
       .DA #8464,#8556,#8649,#8742,#8836,#8930,#9025,#9120
       .DA #9216,#9312,#9409,#9506,#9604,#9702,#9801,#9900
       .DA #10000,#10100,#10201,#10302,#10404,#10506,#10609,#10712
       .DA #10816,#10920,#11025,#11130,#11236,#11342,#11449,#11556
       .DA #11664,#11772,#11881,#11990,#12100,#12210,#12321,#12432
       .DA #12544,#12656,#12769,#12882,#12996,#13110,#13225,#13340
       .DA #13456,#13572,#13689,#13806,#13924,#14042,#14161,#14280
       .DA #14400,#14520,#14641,#14762,#14884,#15006,#15129,#15252
       .DA #15376,#15500,#15625,#15750,#15876,#16002,#16129,#16256
       .DA #16384,#16512,#16641,#16770,#16900,#17030,#17161,#17292
       .DA #17424,#17556,#17689,#17822,#17956,#18090,#18225,#18360
       .DA #18496,#18632,#18769,#18906,#19044,#19182,#19321,#19460
       .DA #19600,#19740,#19881,#20022,#20164,#20306,#20449,#20592
       .DA #20736,#20880,#21025,#21170,#21316,#21462,#21609,#21756
       .DA #21904,#22052,#22201,#22350,#22500,#22650,#22801,#22952
       .DA #23104,#23256,#23409,#23562,#23716,#23870,#24025,#24180
       .DA #24336,#24492,#24649,#24806,#24964,#25122,#25281,#25440
      .DA #25600,#25760,#25921,#26082,#26244,#26406,#26569,#26732
      .DA #26896,#27060,#27225,#27390,#27556,#27722,#27889,#28056
      .DA #28224,#28392,#28561,#28730,#28900,#29070,#29241,#29412
      .DA #29584,#29756,#29929,#30102,#30276,#30450,#30625,#30800
      .DA #30976,#31152,#31329,#31506,#31684,#31862,#32041,#32220
      .DA #32400,#32580,#32761,#32942,#33124,#33306,#33489,#33672
      .DA #33856,#34040,#34225,#34410,#34596,#34782,#34969,#35156
      .DA #35344,#35532,#35721,#35910,#36100,#36290,#36481,#36672
      .DA #36864,#37056,#37249,#37442,#37636,#37830,#38025,#38220
      .DA #38416,#38612,#38809,#39006,#39204,#39402,#39601,#39800
      .DA #40000,#40200,#40401,#40602,#40804,#41006,#41209,#41412
      .DA #41616,#41820,#42025,#42230,#42436,#42642,#42849,#43056
      .DA #43264,#43472,#43681,#43890,#44100,#44310,#44521,#44732
      .DA #44944,#45156,#45369,#45582,#45796,#46010,#46225,#46440
      .DA #46656,#46872,#47089,#47306,#47524,#47742,#47961,#48180
```

```
2720        .DA #48400,#48620,#48841,#49062,#49284,#49506,#49729,#49952
2730        .DA #50176,#50400,#50625,#50850,#51076,#51302,#51529,#51756
2740        .DA #51984,#52212,#52441,#52670,#52900,#53130,#53361,#53592
2750        .DA #53824,#54056,#54289,#54522,#54756,#54990,#55225,#55460
2760        .DA #55696,#55932,#56169,#56406,#56644,#56882,#57121,#57360
2770        .DA #57600,#57840,#58081,#58322,#58564,#58806,#59049,#59292
2780        .DA #59536,#59780,#60025,#60270,#60516,#60762,#61009,#61256
2790        .DA #61504,#61752,#62001,#62250,#62500,#62750,#63001,#63252
2800        .DA #63504,#63756,#64009,#64262,#64516,#64770,#65025,#65280
2810  *--------------------------------
2820     .LIST ON
2830  SQH    .DA /0,/0,/1,/2,/4,/6,/9,/12
2840         .DA /16,/20,/25,/30,/36,/42,/49,/56
2850         .DA /64,/72,/81,/90,/100,/110,/121,/132
2860    .LIST OFF
2870         .DA /144,/156,/169,/182,/196,/210,/225,/240
2880         .DA /256,/272,/289,/306,/324,/342,/361,/380
2890         .DA /400,/420,/441,/462,/484,/506,/529,/552
2900         .DA /576,/600,/625,/650,/676,/702,/729,/756
2910         .DA /784,/812,/841,/870,/900,/930,/961,/992
2920         .DA /1024,/1056,/1089,/1122,/1156,/1190,/1225,/1260
2930         .DA /1296,/1332,/1369,/1406,/1444,/1482,/1521,/1560
2940         .DA /1600,/1640,/1681,/1722,/1764,/1806,/1849,/1892
2950         .DA /1936,/1980,/2025,/2070,/2116,/2162,/2209,/2256
2960         .DA /2304,/2352,/2401,/2450,/2500,/2550,/2601,/2652
2970         .DA /2704,/2756,/2809,/2862,/2916,/2970,/3025,/3080
2980         .DA /3136,/3192,/3249,/3306,/3364,/3422,/3481,/3540
2990         .DA /3600,/3660,/3721,/3782,/3844,/3906,/3969,/4032
3000         .DA /4096,/4160,/4225,/4290,/4356,/4422,/4489,/4556
3010         .DA /4624,/4692,/4761,/4830,/4900,/4970,/5041,/5112
3020         .DA /5184,/5256,/5329,/5402,/5476,/5550,/5625,/5700
3030         .DA /5776,/5852,/5929,/6006,/6084,/6162,/6241,/6320
3040         .DA /6400,/6480,/6561,/6642,/6724,/6806,/6889,/6972
3050         .DA /7056,/7140,/7225,/7310,/7396,/7482,/7569,/7656
3060         .DA /7744,/7832,/7921,/8010,/8100,/8190,/8281,/8372
3070         .DA /8464,/8556,/8649,/8742,/8836,/8930,/9025,/9120
3080         .DA /9216,/9312,/9409,/9506,/9604,/9702,/9801,/9900
3090         .DA /10000,/10100,/10201,/10302,/10404,/10506,/10609,/10712
3100         .DA /10816,/10920,/11025,/11130,/11236,/11342,/11449,/11556
3110         .DA /11664,/11772,/11881,/11990,/12100,/12210,/12321,/12432
3120         .DA /12544,/12656,/12769,/12882,/12996,/13110,/13225,/13340
3130         .DA /13456,/13572,/13689,/13806,/13924,/14042,/14161,/14280
3140         .DA /14400,/14520,/14641,/14762,/14884,/15006,/15129,/15252
3150         .DA /15376,/15500,/15625,/15750,/15876,/16002,/16129,/16256
3160         .DA /16384,/16512,/16641,/16770,/16900,/17030,/17161,/17292
3170         .DA /17424,/17556,/17689,/17822,/17956,/18090,/18225,/18360
3180         .DA /18496,/18632,/18769,/18906,/19044,/19182,/19321,/19460
3190         .DA /19600,/19740,/19881,/20022,/20164,/20306,/20449,/20592
3200         .DA /20736,/20880,/21025,/21170,/21316,/21462,/21609,/21756
3210         .DA /21904,/22052,/22201,/22350,/22500,/22650,/22801,/22952
3220         .DA /23104,/23256,/23409,/23562,/23716,/23870,/24025,/24180
3230         .DA /24336,/24492,/24649,/24806,/24964,/25122,/25281,/25440
3240        .DA /25600,/25760,/25921,/26082,/26244,/26406,/26569,/26732
3250        .DA /26896,/27060,/27225,/27390,/27556,/27722,/27889,/28056
3260        .DA /28224,/28392,/28561,/28730,/28900,/29070,/29241,/29412
3270        .DA /29584,/29756,/29929,/30102,/30276,/30450,/30625,/30800
3280        .DA /30976,/31152,/31329,/31506,/31684,/31862,/32041,/32220
3290        .DA /32400,/32580,/32761,/32942,/33124,/33306,/33489,/33672
3300        .DA /33856,/34040,/34225,/34410,/34596,/34782,/34969,/35156
```

```
3310        .DA /35344,/35532,/35721,/35910,/36100,/36290,/36481,/36672
3320        .DA /36864,/37056,/37249,/37442,/37636,/37830,/38025,/38220
3330        .DA /38416,/38612,/38809,/39006,/39204,/39402,/39601,/39800
3340        .DA /40000,/40200,/40401,/40602,/40804,/41006,/41209,/41412
3350        .DA /41616,/41820,/42025,/42230,/42436,/42642,/42849,/43056
3360        .DA /43264,/43472,/43681,/43890,/44100,/44310,/44521,/44732
3370        .DA /44944,/45156,/45369,/45582,/45796,/46010,/46225,/46440
3380        .DA /46656,/46872,/47089,/47306,/47524,/47742,/47961,/48180
3390        .DA /48400,/48620,/48841,/49062,/49284,/49506,/49729,/49952
3400        .DA /50176,/50400,/50625,/50850,/51076,/51302,/51529,/51756
3410        .DA /51984,/52212,/52441,/52670,/52900,/53130,/53361,/53592
3420        .DA /53824,/54056,/54289,/54522,/54756,/54990,/55225,/55460
3430        .DA /55696,/55932,/56169,/56406,/56644,/56882,/57121,/57360
3440        .DA /57600,/57840,/58081,/58322,/58564,/58806,/59049,/59292
3450        .DA /59536,/59780,/60025,/60270,/60516,/60762,/61009,/61256
3460    .LIST ON
3470        .DA /61504,/61752,/62001,/62250,/62500,/62750,/63001,/63252
3480        .DA /63504,/63756,/64009,/64262,/64516,/64770,/65025,/65280
3490  *--------------------------------
3500    .LIF
```

```
==============================================================================
DOCUMENT :AAL-8603:ProDOS:ROBISONS.8X8.txt
==============================================================================

 1000  *SAVE ROBISONS.8X8
 1010  *--------------------------------
 1020  *   MODIFIED FROM ORIGINAL PROGRAM
 1030  *   BY ARCH D. ROBISON, BURROUGHS CORP.
 1040  *      EDN, OCTOBER 13, 1983.
 1050  *--------------------------------
 1060  *   ENTER WITH (A)=MULTIPLIER # 1
 1070  *              (X)=MULTIPLIER #2
 1080  *   EXIT WITH (A)=PRODUCT HI BYTE
 1090  *             (X)=PRODUCT LO BYTE
 1100  *--------------------------------
 1110  PROD   .EQ $06      PRODUCT TEMP OF M1*M2 (LOW BYTE)
 1120  M2     .EQ $07      TEMP FOR M2 SAVE
 1130  *--------------------------------
 1140  MULT8  TAY          SAVE M1 IN Y
 1150         STX M2       SAVE M2
 1160         AND M2       CHECK IF BOTH FACTORS ARE ODD
 1170         LSR          SET CARRY <--> BOTH ODD
 1180         LDA SQL,X    ADD (X*X)/2 AND (Y*Y)/2
 1190         ADC SQL,Y
 1200         STA PROD     SAVE LO BYTE OF PRODUCT
 1210         LDA SQH,X
 1220         ADC SQH,Y
 1230         TAX          SAVE HI BYTE OF PRODUCT
 1240         TYA          GET M1 BACK
 1250         SEC
 1260         SBC M2       FIND M1 - M2
 1270         BCS .1       M1 >= M2, CONTINUE
 1280         SBC #0       M1 < M2, FORM 2'S COMPLEMENT
 1290         EOR #$FF
 1300  .1     TAY          USE ABS(M1-M2) AS INDEX
 1310         LDA PROD        TO FIND SQUARE/2 IN TABLE
 1320         SBC SQL,Y    NOW SUBTRACT (X-Y)*(X-Y)
 1330         STA PROD     SAVE LO BYTE OF RESULT
 1340         TXA          HI BYTE FROM PREVIOUS SUM
 1350         SBC SQH,Y
 1360         LDX PROD     LO BYTE OF FINAL PRODUCT
 1370         RTS
 1380  *--------------------------------
 1390         .OR $900     PAGE BOUNDARY TO SAVE MAX 6 CYCLES
 1400  *--------------------------------
 1410  SQL
 1420   .DA #0,#0,#2,#4,#8,#12,#18,#24
 1430   .DA #32,#40,#50,#60,#72,#84,#98,#112
 1440   .DA #128,#144,#162,#180,#200,#220,#242,#264
 1450   .DA #288,#312,#338,#364,#392,#420,#450,#480
 1460   .DA #512,#544,#578,#612,#648,#684,#722,#760
 1470   .DA #800,#840,#882,#924,#968,#1012,#1058,#1104
 1480   .DA #1152,#1200,#1250,#1300,#1352,#1404,#1458,#1512
```

```
1490   .DA #1568,#1624,#1682,#1740,#1800,#1860,#1922,#1984
1500   .DA #2048,#2112,#2178,#2244,#2312,#2380,#2450,#2520
1510   .DA #2592,#2664,#2738,#2812,#2888,#2964,#3042,#3120
1520   .DA #3200,#3280,#3362,#3444,#3528,#3612,#3698,#3784
1530   .DA #3872,#3960,#4050,#4140,#4232,#4324,#4418,#4512
1540   .DA #4608,#4704,#4802,#4900,#5000,#5100,#5202,#5304
1550   .DA #5408,#5512,#5618,#5724,#5832,#5940,#6050,#6160
1560   .DA #6272,#6384,#6498,#6612,#6728,#6844,#6962,#7080
1570   .DA #7200,#7320,#7442,#7564,#7688,#7812,#7938,#8064
1580   .DA #8192,#8320,#8450,#8580,#8712,#8844,#8978,#9112
1590   .DA #9248,#9384,#9522,#9660,#9800,#9940,#10082,#10224
1600   .DA #10368,#10512,#10658,#10804,#10952,#11100,#11250,#11400
1610   .DA #11552,#11704,#11858,#12012,#12168,#12324,#12482,#12640
1620   .DA #12800,#12960,#13122,#13284,#13448,#13612,#13778,#13944
1630   .DA #14112,#14280,#14450,#14620,#14792,#14964,#15138,#15312
1640   .DA #15488,#15664,#15842,#16020,#16200,#16380,#16562,#16744
1650   .DA #16928,#17112,#17298,#17484,#17672,#17860,#18050,#18240
1660   .DA #18432,#18624,#18818,#19012,#19208,#19404,#19602,#19800
1670   .DA #20000,#20200,#20402,#20604,#20808,#21012,#21218,#21424
1680   .DA #21632,#21840,#22050,#22260,#22472,#22684,#22898,#23112
1690   .DA #23328,#23544,#23762,#23980,#24200,#24420,#24642,#24864
1700   .DA #25088,#25312,#25538,#25764,#25992,#26220,#26450,#26680
1710   .DA #26912,#27144,#27378,#27612,#27848,#28084,#28322,#28560
1720   .DA #28800,#29040,#29282,#29524,#29768,#30012,#30258,#30504
1730   .DA #30752,#31000,#31250,#31500,#31752,#32004,#32258,#32512
1740  SQH
1750   .DA /0,/0,/2,/4,/8,/12,/18,/24
1760   .DA /32,/40,/50,/60,/72,/84,/98,/112
1770   .DA /128,/144,/162,/180,/200,/220,/242,/264
1780   .DA /288,/312,/338,/364,/392,/420,/450,/480
1790   .DA /512,/544,/578,/612,/648,/684,/722,/760
1800   .DA /800,/840,/882,/924,/968,/1012,/1058,/1104
1810   .DA /1152,/1200,/1250,/1300,/1352,/1404,/1458,/1512
1820   .DA /1568,/1624,/1682,/1740,/1800,/1860,/1922,/1984
1830   .DA /2048,/2112,/2178,/2244,/2312,/2380,/2450,/2520
1840   .DA /2592,/2664,/2738,/2812,/2888,/2964,/3042,/3120
1850   .DA /3200,/3280,/3362,/3444,/3528,/3612,/3698,/3784
1860   .DA /3872,/3960,/4050,/4140,/4232,/4324,/4418,/4512
1870   .DA /4608,/4704,/4802,/4900,/5000,/5100,/5202,/5304
1880   .DA /5408,/5512,/5618,/5724,/5832,/5940,/6050,/6160
1890   .DA /6272,/6384,/6498,/6612,/6728,/6844,/6962,/7080
1900   .DA /7200,/7320,/7442,/7564,/7688,/7812,/7938,/8064
1910   .DA /8192,/8320,/8450,/8580,/8712,/8844,/8978,/9112
1920   .DA /9248,/9384,/9522,/9660,/9800,/9940,/10082,/10224
1930   .DA /10368,/10512,/10658,/10804,/10952,/11100,/11250,/11400
1940   .DA /11552,/11704,/11858,/12012,/12168,/12324,/12482,/12640
1950   .DA /12800,/12960,/13122,/13284,/13448,/13612,/13778,/13944
1960   .DA /14112,/14280,/14450,/14620,/14792,/14964,/15138,/15312
1970   .DA /15488,/15664,/15842,/16020,/16200,/16380,/16562,/16744
1980   .DA /16928,/17112,/17298,/17484,/17672,/17860,/18050,/18240
1990   .DA /18432,/18624,/18818,/19012,/19208,/19404,/19602,/19800
2000   .DA /20000,/20200,/20402,/20604,/20808,/21012,/21218,/21424
2010   .DA /21632,/21840,/22050,/22260,/22472,/22684,/22898,/23112
2020   .DA /23328,/23544,/23762,/23980,/24200,/24420,/24642,/24864
```

```
2030   .DA /25088,/25312,/25538,/25764,/25992,/26220,/26450,/26680
2040   .DA /26912,/27144,/27378,/27612,/27848,/28084,/28322,/28560
2050   .DA /28800,/29040,/29282,/29524,/29768,/30012,/30258,/30504
2060   .DA /30752,/31000,/31250,/31500,/31752,/32004,/32258,/32512
```

```
================================================================================
DOCUMENT :AAL-8603:ProDOS:S.816.DSM.NEW.txt
================================================================================

    1        .LIST OFF
  800          .TI 76,65816 DISASSEMBLER.................FEBRUARY 14,
1985...........
  810  *SAVE S.816.DSM.NEW
  820  *--------------------------------
  830          .OR $300
  840          .OP 65816
  850          LDA #3
  860          SEC
  870          LDA #3
  880          CLC
  890          LDA #3
  900          REP #$30
  910          LDA #3
  920          CLC
  930          XCE
  940          LDA ##$EA34
  950          REP #$20
  960          LDA ##$EA34
  970          LDX ##$EA34
  980          REP #$10
  990          LDA ##$EA34
 1000          LDX ##$EA34
 1010          SEP #$30
 1020          .OR $800
 1030  *--------------------------------
 1040  IMM.SIZE .EQ $00
 1050  LMNEM    .EQ $2C
 1060  RMNEM    .EQ $2D
 1070  FORMATL  .EQ $2E
 1080  LENGTH   .EQ $2F
 1090  FORMATH  .EQ $30
 1100  PCL      .EQ $3A
 1110  PCH      .EQ $3B
 1120  *--------------------------------
 1130  SCRN2  .EQ $F879
 1140  PRNTAX .EQ $F941
 1150  PRBLNK .EQ $F948
 1160  PRBL2  .EQ $F94A
 1170  PCADJ  .EQ $F953
 1180  CROUT  .EQ $FD8E
 1190  PRBYTE .EQ $FDDA
 1200  COUT   .EQ $FDED
 1210  *--------------------------------
 1211    .LIST ON
 1220          .MA ON
 1280  ]1]2]3]4 .DA ']1-64*32+']2-64*32+']3-64*2
 1290          .EM
 1291    .LIF
```

```
1300  *-------------------------------
1310         .MA OXA
1320         .DA #]1-OPNAMES.A/2+128
1330         .EM
1340  *-------------------------------
1350         .MA OXB
1360         .DA #]1-OPNAMES.B/2
1370         .EM
1380  *-------------------------------
1390  T      LDA $C083    WRITE-ENABLE RAM COPY OF MONITOR
1400         LDA $C083
1410         LDA #INSTDSP      PATCH L-COMMAND TO USE MY
1420         STA $FE65         DIS-ASSEMBLER
1430         LDA /INSTDSP
1440         STA $FE66
1444    .LIST ON
1450         LDX #$FF          START WITH E=1
1454         STX E.BIT
1458         STX STATUS.STACK  EMPTY THE STATUS STACK
1462         INX               X=0
1466         STX STATUS.PNTR
1470         RTS
1474  *-------------------------------
1478  E.BIT         .BS 1
1482  STATUS.PNTR   .BS 1
1486  STATUS.STACK  .BS 8
1488    .LIST OFF
1490  *-------------------------------
1500  OPNAMES.A
1510         >ON A,S,L,A
1520         >ON B,R,K
1530         >ON C,L,C
1540         >ON C,L,D
1550         >ON C,L,I
1560         >ON C,L,V
1570         >ON C,O,P
1580         >ON D,E,C,A
1590         >ON D,E,X
1600         >ON D,E,Y
1610         >ON I,N,C,A
1620         >ON I,N,X
1630         >ON I,N,Y
1640         >ON L,S,R,A
1650         >ON N,O,P
1660         >ON P,H,A
1670         >ON P,H,B
1680         >ON P,H,D
1690         >ON P,H,K
1700         >ON P,H,P
1710         >ON P,H,X
1720         >ON P,H,Y
1730         >ON P,L,A
1740         >ON P,L,B
1750         >ON P,L,D
```

```
1760          >ON P,L,P
1770          >ON P,L,X
1780          >ON P,L,Y
1790          >ON R,O,L,A
1800          >ON R,O,R,A
1810          >ON R,T,I
1820          >ON R,T,L
1830          >ON R,T,S
1840          >ON S,E,C
1850          >ON S,E,D
1860          >ON S,E,I
1870          >ON S,T,P
1880          >ON T,A,X
1890          >ON T,A,Y
1900          >ON T,C,D
1910          >ON T,C,S
1920          >ON T,D,C
1930          >ON T,S,C
1940          >ON T,S,X
1950          >ON T,X,A
1960          >ON T,X,S
1970          >ON T,X,Y
1980          >ON T,Y,A
1990          >ON T,Y,X
2000          >ON W,A,I
2010          >ON W,D,M
2020          >ON X,B,A
2030          >ON X,C,E
2040   *---------------------------------
2050   OPNAMES.B
2060          >ON A,D,C
2070          >ON A,N,D
2080          >ON A,S,L
2090          >ON B,C,C
2100          >ON B,C,S
2110          >ON B,E,Q
2120          >ON B,I,T
2130          >ON B,M,I
2140          >ON B,N,E
2150          >ON B,P,L
2160          >ON B,R,A
2170          >ON B,R,L
2180          >ON B,V,C
2190          >ON B,V,S
2200          >ON C,M,P
2210          >ON C,P,X
2220          >ON C,P,Y
2230          >ON D,E,C
2240          >ON E,O,R
2250          >ON I,N,C
2260          >ON J,M,L
2270          >ON J,M,P
2280          >ON J,S,L
2290          >ON J,S,R
```

```
       >ON L,D,A
       >ON L,D,X
       >ON L,D,Y
       >ON L,S,R
       >ON M,V,N
       >ON M,V,P
       >ON O,R,A
       >ON P,E,A
       >ON P,E,I
       >ON P,E,R
       >ON R,E,P
       >ON R,O,L
       >ON R,O,R
       >ON S,B,C
       >ON S,E,P
       >ON S,T,A
       >ON S,T,X
       >ON S,T,Y
       >ON S,T,Z
       >ON T,R,B
       >ON T,S,B
*--------------------------------
OPINDEX
*---0X---------------------------
       >OXA BRK
       >OXB ORA
       >OXA COP
       >OXB ORA
       >OXB TSB
       >OXB ORA
       >OXB ASL
       >OXB ORA
       >OXA PHP
       >OXB ORA
       >OXA ASLA
       >OXA PHD
       >OXB TSB
       >OXB ORA
       >OXB ASL
       >OXB ORA
*---1X---------------------------
       >OXB BPL
       >OXB ORA
       >OXB ORA
       >OXB ORA
       >OXB TRB
       >OXB ORA
       >OXB ASL
       >OXB ORA
       >OXA CLC
       >OXB ORA
       >OXA INCA
       >OXA TCS
       >OXB TRB
```

```
2840        >OXB ORA
2850        >OXB ASL
2860        >OXB ORA
2870 *---2X---------------------------
2880        >OXB JSR
2890        >OXB AND
2900        >OXB JSL
2910        >OXB AND
2920        >OXB BIT
2930        >OXB AND
2940        >OXB ROL
2950        >OXB AND
2960        >OXA PLP
2970        >OXB AND
2980        >OXA ROLA
2990        >OXA PLD
3000        >OXB BIT
3010        >OXB AND
3020        >OXB ROL
3030        >OXB AND
3040 *---3X---------------------------
3050        >OXB BMI
3060        >OXB AND
3070        >OXB AND
3080        >OXB AND
3090        >OXB BIT
3100        >OXB AND
3110        >OXB ROL
3120        >OXB AND
3130        >OXA SEC
3140        >OXB AND
3150        >OXA DECA
3160        >OXA TSC
3170        >OXB BIT
3180        >OXB AND
3190        >OXB ROL
3200        >OXB AND
3210 *---4X---------------------------
3220        >OXA RTI
3230        >OXB EOR
3240        >OXA WDM
3250        >OXB EOR
3260        >OXB MVP
3270        >OXB EOR
3280        >OXB LSR
3290        >OXB EOR
3300        >OXA PHA
3310        >OXB EOR
3320        >OXA LSRA
3330        >OXA PHK
3340        >OXB JMP
3350        >OXB EOR
3360        >OXB LSR
3370        >OXB EOR
```

```
3380  *---5X--------------------------
3390         >OXB BVC
3400         >OXB EOR
3410         >OXB EOR
3420         >OXB EOR
3430         >OXB MVN
3440         >OXB EOR
3450         >OXB LSR
3460         >OXB EOR
3470         >OXA CLI
3480         >OXB EOR
3490         >OXA PHY
3500         >OXA TCD
3510         >OXB JMP
3520         >OXB EOR
3530         >OXB LSR
3540         >OXB EOR
3550  *---6X--------------------------
3560         >OXA RTS
3570         >OXB ADC
3580         >OXB PER
3590         >OXB ADC
3600         >OXB STZ
3610         >OXB ADC
3620         >OXB ROR
3630         >OXB ADC
3640         >OXA PLA
3650         >OXB ADC
3660         >OXA RORA
3670         >OXA RTL
3680         >OXB JMP
3690         >OXB ADC
3700         >OXB ROR
3710         >OXB ADC
3720  *---7X--------------------------
3730         >OXB BVS
3740         >OXB ADC
3750         >OXB ADC
3760         >OXB ADC
3770         >OXB STZ
3780         >OXB ADC
3790         >OXB ROR
3800         >OXB ADC
3810         >OXA SEI
3820         >OXB ADC
3830         >OXA PLY
3840         >OXA TDC
3850         >OXB JMP
3860         >OXB ADC
3870         >OXB ROR
3880         >OXB ADC
3890  *---8X--------------------------
3900         >OXB BRA
3910         >OXB STA
```

```
3920        >OXB BRL
3930        >OXB STA
3940        >OXB STY
3950        >OXB STA
3960        >OXB STX
3970        >OXB STA
3976  *>>>CHANGED FROM "TAY"
3977   .LIST ON
3980        >OXA DEY
3987   .LIST OFF
3990        >OXB BIT
4000        >OXA TXA
4010        >OXA PHB
4020        >OXB STY
4030        >OXB STA
4040        >OXB STX
4050        >OXB STA
4060  *---9X---------------------------
4070        >OXB BCC
4080        >OXB STA
4090        >OXB STA
4100        >OXB STA
4110        >OXB STY
4120        >OXB STA
4130        >OXB STX
4140        >OXB STA
4150        >OXA TYA
4160        >OXB STA
4170        >OXA TXS
4180        >OXA TXY
4190        >OXB STZ
4200        >OXB STA
4210        >OXB STZ
4220        >OXB STA
4230  *---AX---------------------------
4240        >OXB LDY
4250        >OXB LDA
4260        >OXB LDX
4270        >OXB LDA
4280        >OXB LDY
4290        >OXB LDA
4300        >OXB LDX
4310        >OXB LDA
4320        >OXA TAY
4330        >OXB LDA
4340        >OXA TAX
4350        >OXA PLB
4360        >OXB LDY
4370        >OXB LDA
4380        >OXB LDX
4390        >OXB LDA
4400  *---BX---------------------------
4410        >OXB BCS
4420        >OXB LDA
```

```
4430         >OXB LDA
4440         >OXB LDA
4450         >OXB LDY
4460         >OXB LDA
4470         >OXB LDX
4480         >OXB LDA
4490         >OXA CLV
4500         >OXB LDA
4510         >OXA TSX
4520         >OXA TYX
4530         >OXB LDY
4540         >OXB LDA
4550         >OXB LDX
4560         >OXB LDA
4570  *---CX---------------------------
4580         >OXB CPY
4590         >OXB CMP
4600         >OXB REP
4610         >OXB CMP
4620         >OXB CPY
4630         >OXB CMP
4640         >OXB DEC
4650         >OXB CMP
4660         >OXA INY
4670         >OXB CMP
4680         >OXA DEX
4690         >OXA WAI
4700         >OXB CPY
4710         >OXB CMP
4720         >OXB DEC
4730         >OXB CMP
4740  *---DX---------------------------
4750         >OXB BNE
4760         >OXB CMP
4770         >OXB CMP
4780         >OXB CMP
4790         >OXB PEI
4800         >OXB CMP
4810         >OXB DEC
4820         >OXB CMP
4830         >OXA CLD
4840         >OXB CMP
4850         >OXA PHX
4860         >OXA STP
4870         >OXB JML
4880         >OXB CMP
4890         >OXB DEC
4900         >OXB CMP
4910  *---EX---------------------------
4920         >OXB CPX
4930         >OXB SBC
4940         >OXB SEP
4950         >OXB SBC
4960         >OXB CPX
```

```
4970        >OXB SBC
4980        >OXB INC
4990        >OXB SBC
5000        >OXA INX
5010        >OXB SBC
5020        >OXA NOP
5030        >OXA XBA
5040        >OXB CPX
5050        >OXB SBC
5060        >OXB INC
5070        >OXB SBC
5080  *---FX---------------------------
5090        >OXB BEQ
5100        >OXB SBC
5110        >OXB SBC
5120        >OXB SBC
5130        >OXB PEA
5140        >OXB SBC
5150        >OXB INC
5160        >OXB SBC
5170        >OXA SED
5180        >OXB SBC
5190        >OXA PLX
5200        >OXA XCE
5210        >OXB JSR
5220        >OXB SBC
5230        >OXB INC
5240        >OXB SBC
5250  *--------------------------------
5260  OPFORMAT
5270  F.0    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5280  F.1    .HS 26.16.12.1E.02.08.08.22.00.10.00.00.04.0A.0A.0C
5290  F.2    .HS 04.14.06.1C.02.02.02.20.00.00.00.00.04.04.04.06
5300  F.3    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.0A.0A.0A.0C
5310  F.4    .HS 00.14.00.1C.24.02.02.20.00.00.00.00.04.04.04.06
5320  F.5    .HS 26.16.12.1E.24.08.08.22.00.10.00.00.06.0A.0A.0C
5330  F.6    .HS 00.14.28.1C.02.02.02.20.00.00.00.00.18.04.04.06
5340  F.7    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
5350  F.8    .HS 26.14.28.1C.02.02.02.20.00.00.00.00.04.04.04.06
5360  F.9    .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.04.0A.0A.0C
5370  F.A    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5380  F.B    .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.0A.0A.10.0C
5390  F.C    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5400  F.D    .HS 26.16.12.1E.02.08.08.22.00.10.00.00.18.0A.0A.0C
5410  F.E    .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5420  F.F    .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
5430  *--------------------------------
5440  FMTBL
5450  *-----# > ( $ , X S ) , Y $ - - - LL
5460   .DA %1.0.0.1.0.0.0.0.0.0.0.0.0.0.01 -- IMMEDIATE    00
5470   .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.01 -- DIRECT       02
5480   .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.10 -- ABS          04
5490   .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.0.11 -- LONG         06
5500  *-----# > ( $ , X S ) , Y $ - - - LL
```

```
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.0.01 -- DIRECT,X     08
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.0.10 -- ABS,X        0A
 .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.0.0.11 -- LONG,X       0C
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.0.0.01 -- DIRECT,Y     0E
 .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.0.0.10 -- ABS,Y        10
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.0.0.01 -- IND          12
 .DA %0.0.1.1.1.1.0.1.0.0.0.0.0.0.0.01 -- INDX         14
 .DA %0.0.1.1.0.0.0.1.1.1.0.0.0.0.0.01 -- INDY         16
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.0.0.10 -- INDABS       18
 .DA %0.0.1.1.1.1.0.1.0.0.0.0.0.0.0.10 -- INDABSX      1A
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.0.0.1.1.0.1.0.0.0.0.0.0.0.0.01 -- STK          1C
 .DA %0.0.1.1.1.0.1.1.1.1.0.0.0.0.0.01 -- STKY         1E
*-----# > ( $ , X S ) , Y $ - - - LL
 .DA %0.1.1.1.0.0.0.1.0.0.0.0.0.0.0.01 -- INDLONG      20
 .DA %0.1.1.1.0.0.0.1.1.1.0.0.0.0.0.01 -- INDLONGY     22
 .DA %0.0.0.1.0.0.0.0.1.0.1.0.0.0.0.10 -- MVN & MVP    24
 .DA %0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.01 -- RELATIVE     26
 .DA %0.0.0.0.0.0.0.0.0.0.1.0.0.0.0.10 -- LONG RELA.   28
*--------------------------------
FMTSTR .AS -/$Y,)SX,$(>#/
*--------------------------------
INSDS1 JSR CROUT
       LDA PCH
       JSR PRBYTE
       LDA PCL
       JSR PRBYTE
       LDA #"-"
       JSR COUT
       LDA #" "
       JSR COUT
       LDY #0
       LDA (PCL),Y  GET OPCODE
 .LIST ON
*>>>INSERT LINE HERE
       JSR TEST.OP.CODES         <<<>>>
  .LIF
INSDS2 TAY          SAVE IN Y-REG
       LDA OPINDEX,Y
       ASL
       TAX
       BCC .1       ...NOT SINGLE BYTE OPCODE
       LDA OPNAMES.A,X
       STA RMNEM
       LDA OPNAMES.A+1,X
       STA LMNEM
       LDA #0
       STA LENGTH
       RTS
*--------------------------------
.1     LDA OPNAMES.B,X
```

```
6010        STA RMNEM
6020        LDA OPNAMES.B+1,X
6030        STA LMNEM
6040        LDX OPFORMAT,Y
6050        LDA FMTBL+1,X
6060        STA FORMATH
6070        LDA FMTBL,X
6080        STA FORMATL
6090        AND #3
6100        STA LENGTH
6110        TXA          CHECK IF IMMEDIATE
6120        BNE .2       ...NO
6130        BIT IMM.SIZE CHECK IF 16-BIT MODE
6140        BPL .2       ...NO
6150        INC LENGTH   ...YES
6160 .2     RTS
6170 *--------------------------------
6180 INSTDSP
6190        JSR INSDS1
6200        LDY #0       PRINT BYTES OF OPCODE & OPERAND
6210 .1     LDA (PCL),Y
6220        JSR PRBYTE
6230        LDX #1       PRINT 1 BLANK
6240 .2     JSR PRBL2
6250        CPY LENGTH
6260        INY
6270        BCC .1
6280        LDX #3
6290        CPY #4
6300        BCC .2
6310 *---PRINT MNEMONIC---------------
6320        LDY #3       THREE LETTERS
6330 .3     LDA #6       SHIFT OUT ONE LETTER, TOP BITS 11
6340 .4     ASL RMNEM
6350        ROL LMNEM
6360        ROL
6370        BPL .4       ...NOT ENUF BITS YET
6380        JSR COUT     PRINT THE LETTER
6390        DEY
6400        BNE .3       ...MORE LETTERS
6410        LDY LENGTH
6420        BEQ .8       ...SINGLE BYTE OPCODE
6430        LDA FORMATL
6440        AND #$20     SEE IF SPECIAL
6450        BNE .9       ...YES, MOVES OR RELATIVES
6460 *---PRINT NORMAL OPERANDS--------
6470        LDA #" "
6480        JSR COUT
6490        LDX #10      11 FORMAT BITS
6500 .5     ASL FORMATL
6510        ROL FORMATH
6520        BCC .7
6530        LDA FMTSTR,X
6540        JSR COUT
```

```
6550         CMP #"#"
6560         BNE .55
6570         BIT IMM.SIZE
6580         BPL .7
6590         JSR COUT
6600  .55    CMP #"$"
6610         BNE .7
6620  .6     LDA (PCL),Y
6630         JSR PRBYTE
6640         DEY
6650         BNE .6
6660  .7     DEX
6670         BPL .5
6680  .8     RTS
6690  *---SPECIAL CASES----------------
6700  .9     LDA #" "
6710         JSR COUT
6720         LDA #"$"
6730         JSR COUT
6740         LDA FORMATL
6750         BMI .11      MVN & MVP
6755   .LIST ON
6760  *---8- OR 16-BIT RELATIVE--------
6770         LDA (PCL),Y  8=OFFSET, 16=OFFSETHI
6780         DEY          TEST LENGTH
6790         STY FORMATH  =0 IF 8-BIT
6800         BEQ .10      ...8-BIT
6810         STA FORMATH  ...16-BIT
6820         LDA (PCL),Y  LOW BYTE OF 16-BIT OFFSET
6830  .10    STA FORMATL
6840         JSR PCADJ
6850         CLC
6860         ADC FORMATL
6870         TAX
6880         TYA
6890         ADC FORMATH
6900         JMP PRNTAX
6905    .LIST OFF
6960  *---MVN & MVP--------------------
6970  .11    LDA (PCL),Y
6980         JSR PRBYTE
6990         LDA #","
7000         JSR COUT
7010         LDA #"$"
7020         JSR COUT
7030         DEY
7040         LDA (PCL),Y
7050         JMP PRBYTE
7055    .LIST ON
7060  *--------------------------------
7070  TEST.OP.CODES
7080         PHA          SAVE OPCODE
7090         LSR IMM.SIZE      ASSUME 8-BIT IMMEDIATE
7100         LDX STATUS.PNTR
```

```
7110        CMP #$18     CLC?
7120        BEQ CLC.OP
7130        CMP #$38     SEC?
7140        BEQ SEC.OP
7150        INY
7160        CMP #$C2     REP?
7170        BEQ REP.OP
7180        CMP #$E2     SEP?
7190        BEQ SEP.OP
7200        DEY
7210        CMP #$08     PHP?
7220        BEQ PHP.OP
7230        CMP #$28     PLP?
7240        BEQ PLP.OP
7250        CMP #$FB     XCE?
7260        BEQ XCE.OP
7270 *---------------------------------
7280        AND #$1F     ORA, AND, EOR, ADC, BIT, LDA, CMP, SBC?
7290        CMP #$09
7300        PHP          SAVE ANSWER
7310        LDA #$20     ASSUME M-BIT
7320        PLP          GET PREVIOUS ANSWER
7330        BEQ .1       IT IS M-BIT
7340        LSR    (LDA #$10)    USE X-BIT INSTEAD
7350 .1     AND STATUS.STACK,X
7360        BNE .2       ...USE 8-BIT IMMEDIATE
7370        LDA E.BIT
7380        LSR
7390        BCS .2       E=1, USE 8-BIT IMMEDIATE
7400        LDA #$FF     ...USE 16-BIT IMMEDIATE
7410        STA IMM.SIZE
7420 .2     PLA          GET OPCODE AGAIN
7430        RTS
7440 *---------------------------------
7450 CLC.OP LDA STATUS.STACK,X
7460        AND #$FE
7470 UPDATE.STATUS
7480        STA STATUS.STACK,X
7490        PLA
7500        RTS
7510 *---------------------------------
7520 SEC.OP LDA STATUS.STACK,X
7530        ORA #$01
7540        BNE UPDATE.STATUS   ...ALWAYS
7550 *---------------------------------
7560 REP.OP LDA (PCL),Y     LOOK AT OPERAND
7570        EOR #$FF
7580        AND STATUS.STACK,X
7590        JMP UPDATE.STATUS
7600 *---------------------------------
7610 SEP.OP LDA (PCL),Y
7620        ORA STATUS.STACK,X
7630        JMP UPDATE.STATUS
7640 *---------------------------------
```

```
PHP.OP LDA STATUS.STACK,X
       INX
       CPX #8
       BCC PHP.PLP
       LDX #0
PHP.PLP
       STX STATUS.PNTR
       JMP UPDATE.STATUS
*--------------------------------
PLP.OP DEX
       BPL PHP.PLP
       LDX #7
       BEQ PHP.PLP
*--------------------------------
XCE.OP LSR E.BIT    GET E-BIT INTO CARRY
       PHP          SAVE IT
       LDA STATUS.STACK,X
       STA E.BIT    NEW E-BIT
       LSR          C-BIT INTO CARRY
       BCC .1       ...NEW E-BIT = 0
       ORA #$18     ...NEW E-BIT=1, SO SET M=X=1
.1     PLP          GET NEW C-BIT (OLD E-BIT)
       ROL          PUT IT INTO STATUS BYTE
       JMP UPDATE.STATUS
*--------------------------------
TT     LDY #0
       LDA #$C0
       STA PCL
       LDA #2       $2C0...$3C3
       STA PCH
.1     TYA
       STA $2C0,Y
       INY
       BNE .1
       STY $3C0
       INY
       STY $3C1
       INY
       STY $3C2
.2     JSR INSTDSP
       LDY #0
       LDA (PCL),Y
       CMP #$FF
       BEQ .3
.4     LDA $C000
       BPL .4
       STA $C010
       INC PCL
       BNE .2
       INC PCH
       BNE .2       ...ALWAYS
.3     RTS
*--------------------------------
   .LIF
```

================================================================
DOCUMENT :AAL-8603:ProDOS:S.WHICH.PROC.txt
================================================================

```
 1000  *SAVE S.WHICH.PROC
 1010         .OP 65802
 1020  *--------------------------------
 1030  PRBYTE .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  *--------------------------------
 1060  WHICH.PROCESSOR
 1070         LDA #$65
 1080         JSR PRBYTE
 1090         BRA .1
 1100         JMP .2
 1110  .1     LDA #"8"
 1120         XBA
 1130         LDA #"C"
 1140         XBA
 1150         JSR COUT
 1160  .2     LDA #$02
 1170         JMP PRBYTE
 1180  *--------------------------------
```

```
===========================================================================
DOCUMENT :AAL-8603:ProDOS:TEST.CKSUMMER.txt
===========================================================================

 1000   .LIF
 1010  *SAVE TEST.CKSUMMER
 1020  *--------------------------------
 1030  *   SIMULATE PRODOS $FB09-FB10 CHECK-SUMMER
 1040  *      (AT $267C IN PRODOS 1.1.1)
 1050  *--------------------------------
 1060  T
 1070         LDA #S1
 1080         STA $0A
 1090         LDA /S1
 1100         STA $0B
 1110         JSR CS
 1120         LDA #S2
 1130         STA $0A
 1140         LDA /S2
 1150         STA $0B
 1160  CS
 1170         JSR PT
 1180         CLC
 1190         LDY #0
 1200         STY X
 1210  .1     LDA ($0A),Y
 1220         JSR B
 1230         AND #$DF
 1240         JSR B
 1250         LDA X
 1260         JSR B
 1270         LDA ($0A),Y
 1280         AND #$DF
 1290         ADC X
 1300         STA X
 1310         JSR B
 1320         ROL X
 1330         LDA X
 1340         JSR B
 1350         JSR $FD8E
 1360         INY
 1370         CPY #8
 1380         BCC .1
 1390         TYA
 1400         ASL
 1410         ASL
 1420         ASL
 1430         ASL
 1440         ORA X
 1450         JSR B
 1460         ADC #$0B
 1470  *--------------------------------
 1480  B      PHA
```

```
1490        PHP
1500        JSR $FDDA
1510        LDA #" "
1520        JSR $FDED
1530        JSR $FDED
1540        PLP
1550        PLA
1560        RTS
1570 *--------------------------------
1580 X      .BS 1
1590 *--------------------------------
1600 S1     .AS -/APPLE ][/
1610 S2     .HS B0.A2.20.4A.FF.38.B0.9E
1620 *--------------------------------
1630 TITLE  .HS 8D8D
1640        .AS -/LDA AND ADC STA ROL/
1650        .HS 8D00
1660 *--------------------------------
1670 PT
1680        LDY #0
1690 .1     LDA TITLE,Y
1700        BEQ .2
1710        JSR $FDED
1720        INY
1730        BNE .1
1740 .2     RTS
1750 *--------------------------------
```

On Dividing a BCD Value by 4.............Bob Sander-Cederlof

The 6502 allows two kinds of addition and subtraction operations, depending on the state of the D-bit in the status register. After a SED (Set D) instruction, the ADC and SBC instructions will operate in decimal mode; after CLD (CLear D), ADC and SBC will operate in binary mode.

In decimal mode the range of values in a byte is from $00 to $99. The left nybble is the ten's digit, and the right nybble is the unit's digit. The decimal mode makes some programs much easier to write, and others more difficult. Having both modes is nice.

In binary mode, if you want to divide by four you just shift the value right two bit-positions. If by 8, shift 3 times. And so on. In decimal mode, you can very easily divide by powers of ten; however, dividing by four is more difficult.

I needed a quick way to tell if a number in decimal mode was divisible by four. After inspecting the binary values of the decimal-mode numbers between 00 and 99 a, I found a way. If the ten's digit is even and the unit's digit 0, 4, or 8, the number is divisible by four. Also, if the ten's digit is odd and the unit's digit is 2 or 6, the number is divisible by four. This can be tested as follows:

```
       LDA VALUE
       AND #$13
       BEQ ...     ...TEN'S EVEN, UNITS=0,4,8
       EOR #$12
       BEQ ...     ...TEN'S ODD, UNITS=2,6
       ...         ...NOT DIVISIBLE
```

Next I needed a way to actually divide by four. Again I started by inspecting the various values involved. Simply shifting right twice does not do the job, except for numbers less than ten. You cannot even divide by two by simply shifting right once, unless the ten's digit is even. Hmmm.... If the ten's digit is odd, I could subtract 6 frist and then shift right once to divide by two. Doing all that twice would result in a division by four. The subtraction must be done in binary mode, not decimal. The subroutine below in lines 1460-1590 will divide the decimal number in VALUE by four, truncating any remainder, and return the quotient in the A-register. Lines 1600-1700 show a shorter way to divide by two, provided you don't mind using the X-register.

To test my subroutines, I wrote some test programs. The first program, lines 1000-1370, runs through the values 00 to 99, printing ten values to a line. Each number that is evenly divisible by four is flagged with an asterisk.  The second program, lines 1720-1990, shows the quotient after calling DIVIDE.BCD.VALUE.BY FOUR.

I am sure there must be lots of other neat tricks possible by combining binary and decimal modes in the 6502.  Do you know some? Send them in, and we will publish the best!

==============================================================================
DOCUMENT :AAL-8604:Articles:Boot.80.txt
==============================================================================

Booting into 80 Columns..........................Bill Morgan

The ProDOS version of the S-C Macro Assembler is carefully written to operate in either 40 or 80 columns. When you boot the disk the assembler starts out in the 40 column mode, because we couldn't take for granted that you would have (or want) the 80 column display. Well it turns out that most people (myself included) are using 80 columns and are getting tired of typing PR#3 every time they start up the assembler.

Marc Wolfgram called up today from Wisconsin to ask how to make the assembler start up in 80 columns, and that finally got me around to finding out how. It's embarassingly easy: just a two-byte patch. Here's the procedure, assuming you're in S-C Macro Assembler ProDOS:

```
UNLOCK SCASM.SYSTEM
BLOAD SCASM.SYSTEM,A$2000,TSYS
$6001:00 C3
BSAVE SCASM.SYSTEM,A$2000,TSYS,L17920
LOCK SCASM.SYSTEM
```

We just changed the IO.INIT call from JMP MON.HOME to JMP $C300, and that's all there is to it! Now the next time you boot up, the assembler will be in 80 column mode. RESET will return you to 40 columns. PR#3 or NEW will restore 80 columns.

Thanks, Marc, for prompting me to find out about this.

==============================================================================
DOCUMENT :AAL-8604:Articles:Front.Page.txt
==============================================================================

$1.80

Volume 6 -- Issue 7                         April, 1986

In This Issue...



65816 Books

The race is on!  "Programming the 65816", by David Eyes from Prentice-Hall, originally scheduled for publication last October, is now expected in late April.  "65816/65802 Assembly Language Programming", by Michael Fischer from Osborne/McGraw- Hill, scheduled for May publication, is now also due in late April.  We have plenty of copies of these books on order, and a long list of patient people waiting for complete information on programming these powerful new chips.  Coming Soon...

More Memory Expansion

We'd like to call your attention to the new ad for Applied Engineering's RamFactor board.  This is a "Slinky" style memory expansion card for any standard slot of an Apple II, II+, or //e. We've been doing some of the firmware for this product, and it's been a delight to work with.

One thing the ad doesn't really emphasize is the power and flexibility of the program switcher firmware.  You can set up the card with a variable number of variable-sized partitions and then switch between them almost instantly.  Any partition can be based on any operating system, or on your own program.  Couple this with the battery backup option (it's really more of an uninterruptible power suppply for the card) and you have what amounts to a hard disk operating at RAM speed!

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries.  Back issues are available for $1.80 each (other countries add $1 per back issue for postage).  A subscription to the newsletter and the Monthly Disk containing all source code is $45 per year in the US, Canada and Mexico, and $87 to other countries.

==================================================================
DOCUMENT :AAL-8604:Articles:IIc.ROM.Bug.txt
==================================================================

A "Gotcha!" in New //c ROMs.................Robert H. Bernard

Apple seems to have installed a bug in the new ROM for the Apple //c which affects DOS 3.3. I am talking about the 3.5 ROM that supports Unidisk 3.5 and AppleTalk.

The new bug manifests itself when you use the control-IxxN command to either serial port. The older //c ROMs accumulated the "xx" number in $47F; the new ones do it in $47E. Location $47E is supposed to be dedicated to slot 6, the slot where the disk drives are. DOS uses $47E to keep track of the current track position for drive 1. So, after doing the serial port command to set line length, the next time DOS tries to look at drive 1 it will have to re-calibrate.

Re-calibration is not a disaster, but it is annoying. A needless and not noiseless waste of time. To avoid it with the new ROMs, you have to save and restore the contents of $47E around any serial port command that involves scanning a numeric field.

I have looked through the entire listing of the 3.5 ROM that came with my upgrade kit, and there does not appear to be any reason why this variable was moved. Location $47F is not used for any new value that I can see.

Even though the Apple //c Technical Reference Manual reserves $47E for the firmware, and ProDOS doesn't use the cell, using a "slot 6" screen-hole for a slot 1 and 2 activity is a serious breach of the protocol for their use that dates back to the earliest Wozdays. I know Apple is dropping (or at least decreasing) their support of DOS 3.3, but this is ridiculous!

==============================================================================
DOCUMENT :AAL-8604:Articles:Msg.Into.Window.txt
==============================================================================

Writing Messages in Windows..............Bob Sander-Cederlof

The idea for the following program came from some similar code in the Cirtech Flipster software. Their "program manager" software displays a series of messages and menus in selected windows using a simple subroutine.

The windows are not quite as sophisticated as you may be used to if you are a Macintosh fan. This program divides the screen up vertically, with each window running the full screen width. Calls to the program specify which window to write a message into. The JSR MSG.IN.WINDOW is followed by a single byte specifying which window to use, the ASCII text of the message, and a final 00 byte signifying the end of message. MSG.IN.WINDOW first sets up the window, then clears it, then displays the message in it, and then returns to continue execution right after the 00 byte. MSG.IN.WINDOW does not make any provision for saving the previous contents of the screen inside the window and restoring it later. As I said, this is much simpler than Mac windows.

The Apple monitor has built-in window capability, with the current window being defined by four bytes in page zero. $20 is called LEFT, and defines the starting column of a screen line. This is normally 0, meaning the first column. $21 is called WIDTH, and specifies how many characters are in each line. This is usually 40 ($28), but may be 80 ($50) in a //c or enhanced //e in 80-column mode. MSG.IN.WINDOW does not make any changes to LEFT or WIDTH, although you could modify it to do so.

$22 is called WNDTOP, and specifies the top line of the working window. This is usually 0, meaning to start at the top of the screen. It could be as large as 23 ($17), meaning the bottom line of the screen. $23 is called WNDBOT, and specifies the bottom line of the working window. The number in WNDBOT is actually the number of the next line below the working window, and is usually 24 ($18) to specify a window that goes all the way to the bottom of the screen. MSG.IN.WINDOW stores new values in WNDTOP and WNDBOT, according to a table of line numbers called WINDOW.DATA.

My WINDOW.DATA table lists six different windows, but of course you could have as many as you wish. They can even overlap. The table I used contains the line numbers 0, 24, 0, 3, 9, 18, 20, and 24. This corresponds to the following windows:

```
     Index     WNDTOP WNDBOT Window
       0          0     24    0-23  <full screen>
       1       <better not use!!!>
       2          0      3    0-2
       3          3      9    3-8
```

| | | | |
|---|---|---|---|
| 4 | 9 | 18 | 9-17 |
| 5 | 18 | 20 | 18-19 |
| 6 | 20 | 24 | 20-23 |

Lines 1080-1130 in the listing below detail the calling sequence for MSG.IN.WINDOW. The test program in lines 1500 and following shows some actual calls, with a "wait for any keystroke" between messages so you can see it happen.

Lines 1140-1180 save the caller's return address, placed on the stack by the JSR MSG.IN.WINDOW. This address will be used to pick up the calling parameters, and then used to return to the calling program. The subroutine in lines 1400-1460 increments the pointer and picks up the next byte from the calling sequence.

When we are finished displaying the message, the pointer will be pointing at the terminal 00 byte. Placing the pointer address back on the stack lets us use an RTS opcode to return to the caller. This is done in lines 1340-1390.

Lines 1200-1250 pick up the window index from the first byte following the JSR instruction. This indexes the WINDOW.DATA table, so two entries from that table are moved into WNDTOP and WNDBOT. The the monitor HOME subroutine can be called to clear the window and place the cursor in the top-left corner of the window.

Lines 1270-1330 display the message, if any. If there is no message, there still must be a terminal 00 byte. By judicious use of 8D (return) and 8A (linefeed) characters, you can display the message any way you like. If the message is too large for the window, lines will be scrolled out the top of the window and lost.

The MSG.IN.WINDOW subroutine illustrates a commonly used technique of placing messages to be printed "in-line", like PRINT "message" statements in Applesoft. I personally prefer to collect all my messages together, and use a message number in a register to select which one to print. One problem with my preferred method is that my programs are then easier to disassemble ... if that is a problem. The 6502 was not designed for easy transfer of calling parameters which follow the JSR. (The 65816 makes this kind of code easier, with its stack-relative address mode.)

=======================================================================
DOCUMENT :AAL-8604:Articles:NewDOSInit.Boot.txt
=======================================================================

Faster Boot and More Space for DOS 3.3....Bob Sander-Cederlof

A freshly initialized DOS 3.3 disk has 496 free sectors, less whatever is used by your HELLO program. There are 16 more sectors that are either never used or which are wasted, in tracks 0 and 2. The following program modifies the code which writes the DOS image and the code which reads it back during boot, so that the entire image fits in tracks 0 and 1. A further change makes the space in track 2 available for normal files.

The new boot procedure actually is faster than the standard one, and all the new code takes less space than that which is replaced. All you give up is the ability to boot into machines with less than 48K. Does anyone still have one?

Standard DOS 3.3 stores the DOS image in two pieces. The code destined for $B600-BFFF is on track 0, sectors 0 through 9. The code for $9D00-B500 follows, from track 0/sector 10 through track 2/sector 4. Sectors 5-15 of track 2 are not used. The information stored in sectors 3 and 4 of track 2 (aimed at $B400-B5FF) is useless, because all this space is variables for DOS which do not need to be initialized. The same goes for sector 5 of track 0. The contents of sectors 10 and 11 of track 0 is not used on a "slave" disk, which is what you get with the INIT command. My disks have to stay slave disks, because we are going to reshuffle everything around so all the unused sectors end up in track 2.

My new layout stores $9D00-9DFF in track 0/sector 5, and $9E00-B3FF in track 0/sector 10 through track 1/sector 15. The following table summarizes the old and new layouts.

| Sector | Track 0 | | Track 1 | | Track 2 | |
|---|---|---|---|---|---|---|
| | Old | New | Old | New | Old | New |
| 0 | B6 | B6 | A1 | A4 | B1 | .. |
| 1 | B7 | B7 | A2 | A5 | B2 | .. |
| 2 | B8 | B8 | A3 | A6 | B3 | .. |
| 3 | B9 | B9 | A4 | A7 | B4 | .. |
| 4 | BA | BA | A5 | A8 | B5 | .. |
| 5 | BB | 9D | A6 | A9 | .. | .. |
| 6 | BC | BC | A7 | AA | .. | .. |
| 7 | BD | BD | A8 | AB | .. | .. |
| 8 | BE | BE | A9 | AC | .. | .. |
| 9 | BF | BF | AA | AD | .. | .. |
| 10 | .. | 9E | AB | AE | .. | .. |
| 11 | .. | 9F | AC | AF | .. | .. |
| 12 | 9D | A0 | AD | B0 | .. | .. |
| 13 | 9E | A1 | AE | B1 | .. | .. |
| 14 | 9F | A2 | AF | B2 | .. | .. |
| 15 | A0 | A3 | B0 | B3 | .. | .. |

I published the complete commented disassembly of the code which writes the DOS image on a disk and the code for the second stage boot in AAL way back in October, 1981. The second stage boot code begins at $B700, and the DOS writer starts at $B74A. They both use a subroutine at $B793 to read/write a range of sectors. I preserved the starting points for these two routines in the program which follows, but there is a lot of new empty space. If you are interested, you could go ahead and shove all the code segments together, patch all the calls for the new locations, and get one big area of free space for adding new features.

I was able to save coding space in several ways. First, by deciding that I would not worry about running in less than 48K. Second, that I could eliminate the extra code used to clobber the language card. This is a very common patch anyway, because most of us do not want to have to keep re-loading the language card area just because we re-boot DOS. Third, by eliminating the redundant calls to $FE89 and $FE93. The first stage boot does both of these just before jumping to the second stage boot, so there is no reason to do them again. And fourth, by being more efficient. If you want to, you can save even more by doing away with the subroutine at $B7C2: part of it is redundant, and the rest can be combined with the code at $B74A.

The standard DOS boot first loads $B600-BFFF from track 0, and then skips out to track 2 to read the rest hind-end-first. The track steps are 0-1-2-1-0. My new version starts in track 0, reads it all, then reads all of track 1, and it is done. The track steps are simply 0-1. It is a lot faster. However, the overall boot time is not significantly faster, due to the time spent finding track 0 in the first place, and the time spent loading the HELLO program.

Lines 1060-1140 install the new code. The entire $B7 page is replaced, as well as a single byte at $AEB3. This byte changes the VTOC on the newly initialized disk so that track 2 is available. While I was looking at this area, I noticed that the VTOC written on the new disk is not necessarily correct. DOS does not create an entirely new VTOC for the new disk. The bitmap area is new, and several other bytes are set up. However, DOS does not store any values in the bytes which tell how many tracks, sectors per track, sector size, and T/S entries per T/S list. This means that if the last access to a disk prior to initializing a new one was to a non-standard disk, the VTOC may be incorrect on the new disk. If I load a file from a large volume on my Sider, and then INIT a floppy, the floppy's VTOC indicates 32 sectors per track and 50 tracks! Ouch! Beware!

Lines 1180-1480 are the second stage boot code. The first stage boot is located at $B601, and actually executed at $801. It loads in sectors 0-9 of track 0 into $B600-BFFF, calls $FE89 and $FE93 to set the standard 40-column input hooks, and then jumps to $B700 with the slot*16 in the X-register. My stage two begins by copying the information which came from sector 5, now found at $BB00-BBFF, to the place it belongs at $9D00-9DFF (lines 1270-1320). Next I set up a call to my RWFT subroutine.

RWFT stands for Read/Write From Table. I have a table that describes all of the segments which must be loaded from the disk during boot, or written during initialization. Stage two boot must read the same things written by initialization, but init-ing requires first writing the stuff which will be loaded by stage one boot. Stage two boot calls RWFT with A=1 (read opcode for the IOB) and Y=2 (skipping the first two entries in the table). Initialization calls RWFT with A=2 (write opcode) and Y=0 (start at the beginning of the table).

RWFT gets four items out of the table for each step. The page number and sector number indicate the end of the range to be read or written. The count tells how many pages (or sectors) need to be read or written. All of the sectors must be in the track specified by the table entry. After one range has been read, RWFT steps to the next. The table terminates when the page address of 0 is found.

For some reason the code at $AEFF looks like this:

```
       AEFF-  JSR $B7C2
       AF02-  JSR $B74A
```

Both of these subroutines are never called from any other place, so they could be combined into one. Doing so would save several bytes. Furthermore, at least with my new RWFT program, lines 2120 and 2130 could be deleted, saving six more bytes.

There are still more ways to increase the storage on standard floppies, as you probably know. You can shorten the catalog, make a few other patches, and use some sectors in track 17 ($11).

You can usually use more than 35 tracks, since most drives will handle at least 36 and many a full 40. This also only takes a few simple patches. At $AEB5 you normally find a value $8C. Add 4 to this value for each additional track. This controls the loop that builds the bitmap of available sectors in the VTOC. The byte at $BEFE controls how many tracks the formatter in RWTS lays down. It is normally $23 (decimal 35), so add one for each additional track. Just before you start the INIT command, change the byte at $B3EF. This is normally $23, the number of tracks. Add 1 for each additional track. You have to be sure to do this last patch just prior to the INIT, because reading or writing another disk will cause it to be changed back.

Incidentally, this reminds me of the potential bug I mentioned above regarding writing out an incorrect VTOC. Once today I tried to catalog a disk that had been only partially initialized. The tracks had been written, but no VTOC or catalog sectors were. Of course I got an I/O ERROR. Next I decided to INIT that same disk. It went through the formatting stage, then bombed out with an I/O error when trying to write the catalog. Looking at the VTOC on this disk, the bytes for number of tracks, et cetera, were all zero!.

Now back to extra tracks. After making a disk with the extra tracks, you really need to check them to be sure your drive handles them. Use a disk zap program and try to write on the last track. Then try to write on the previous track. If your drive will go out that far, you will be successful. If you get an error trying to find the next to the last track, keep backing up until you find a track that does work. All the ones in between were written in the same location on the disk surface as the last track. If there were any missing tracks, you need to reformat the disk with fewer tracks.

And interesting side not to this discussion is that you could format a disk with LESS than 35 tracks if you wish. Just so you at least include track 17 ($11), you can reduce the values at $BEFE, $B3EF, and $AEB5 and stop short of a full disk. Some copy protection schemes do this, along with other tricks, to frustrate the making of copies.

==================================================================
DOCUMENT :AAL-8604:Articles:Rest.Clob.Cata.txt
==================================================================

Tools for Restoring Lost Catalogs.........Bob Sander-Cederlof

From time to time it happens. One way or another I manage to clobber a catalog track on a disk. I have done it three times to Volume 1 in the DOS partition on my 10-megabyte Sider. (All it takes is "INIT HELLO,V1", forgetting that the last slot I accessed was the Sider's.)

All of the other tracks are still intact, but there is no way to get to them because the catalog is totally wiped out. One solution would be to have an accurate backup floppy for each Sider volume. This should be especially easy for Volume 1, because it is mostly standard Sider utilities. Mostly.... I have modifed several of them, and somehow I almost always have several programs-under-development that end up in V1. Of course, I could just as easily destroy the catalog track on any other volume, or any floppy for that matter.

It is for mistakes like mine that the program FIXCAT in "Bag of Tricks" was invented. FIXCAT looks over a diskette, finds all the sectors which look like they contain track/sector lists, and tries to piece together a new catalog track. Even though it is fairly automatic, I find it very difficult to use. I am always getting confused between old (deleted) copies of files and the current ones, and my disks usually have at least 2 or 3 dozen active files.

Recently it happened again. In fact, while I was working on one of the other articles in this issue of AAL. I decided to write a couple of utilities to help me make more effective use of FIXCAT. My new tools turn out to be useful even without FIXCAT, and you might enjoy just playing with them.

I assume you have a copy of "Beneath Apple DOS", or some other reference work which explains the format of DOS disks, catalog tracks, and track/sector lists.

The first tool I wrote looks through the tracks and sectors of a damaged disk for any sectors containing what could be track/sector lists. When one is found, I display the location of the supposed TS-list, all of the track/sectors in the list, and the first 64 bytes of the first data sector of the supposed file. Here is an example of the display:

```
03-5:  03-4 03-3 03-2
       07 02 09 E8 03 81 2E 4C 49 46 00 16 F2 03 2A C0
...h...LIF..r.*@
       06 08 53 41 56 45 81 42 43 44 81 4D 41 47 49 43
..SAVE.BCD.MAGIC
       00 08 FC 03 2A C0 20 2D 00 05 06 04 54 00 0B 10 ..|.*@
-....T...
```

```
       04 87 4C 44 41 81 23 30 00 0C 1A 04 2E 31 85 53
..LDA.#0.....1.S
```

The first 64 bytes are displayed in both hexadecimal and in ASCII, with periods being substituted for unprintable characters.

Having this information on paper before starting up FIXCAT is a big help. I can peacefully analyze the data at my desk, without the fear and panic associated with making "life and death" decisions at the keyboard. The first few bytes of a file will usually reveal what type of file it is.

If it is a source code file for the S-C Macro Assembler, Integer BASIC, or Applesoft, it will begin with a two-byte length for the file. Binary files begin with the load address, then the length. Text files start right in with data in ASCII, normally with all the high bits on. Since I almost always have a line near the beginning of my source files which contains the file name, I can usually read that file name in the dump of the first 64 bytes.

The FIND.TS.LISTS program is fairly short and simple. Starting from the bottom, the subroutine READTS at lines 2370-2430 calls on RWTS to read a particular track and sector. I elected to use my own IOB, rather than the one inside DOS at $B7E8. For simplicity's sake I assembled in the slot, drive, and volume information in my IOB. READTS only has to store the desired track and sector numbers, and call RWTS. I limited error handling to just re-calling RWTS, in the hopes of eventually succeeding. Should this begin to be a problem, I could print out an error message and either quit or continue with the next sector.

The subroutine READ.NEXT.SECTOR, lines 2200-2350, is used to scan through the disk from beginning to end. TS-lists cannot be in track 0, so I start with track 1. Since DOS allocates sectors in a track starting with sector $0F and going backwards to sector $00, I decided to scan the same way. This makes the files found list more closely to the same order as they were in the original catalog. I first advance the track/sector to the next one, then read it. Thus after reading, CUR.TRACK and CUR.SECTOR are pointing to the one we just read.

Now back to the top. Lines 1100-1130 start CUR.TRACK and CUR.SECTOR at 0. The first call to READ.NEXT.SECTOR will advance them to track 1, sector $0F. Successive calls will read the rest of track 1, then advance to track 2, and so on until we have finished track $22. When we try to read track $23, which does not exist, READ.NEXT.SECTOR will return with carry set and our program will end.

Lines 1170-1290 examine the data in the sector just read to see if it might be a track/sector list. The method I use is to require that there be at least one TS-pair, at BUF+12. I also require that all of the bytes beyond BUF+12 are within the range of valid track-sector pairs. If any bytes are out of range, I assume the current sector is not a TS-list. My tests seem to be adequate, because with every disk I have used it on it found all and only the TS-lists.

Having found TS-list, I call DISPLAY.TS.LIST to display it. Lines 1450-1540 display the location of the TS-list. The subroutine PR.TS prints the track and sector numbers from the A- and X-registers in the form "TT-S". Lines 1550-1720 list the TS-pairs in the TS-list, stopping at the first pair with a track number of zero. Up to 8 pairs are listed on a line.

Lines 1330-1430 read the first data sector of the supposed file, and display the first 64 bytes in hex and ASCII. This display is done by calling DISPLAY.NEXT.16 four times.

As it happens, I did have a fairly recently made backup of the clobbered disk. I thought I should also run my program against this good disk, and comparing the two displays would enable me to pinpoint each active file. However, what I really want from the GOOD disk is the information in the CATALOG. I decided to modify FIND.TS.LISTS to be driven from the catalog track, rather than from a search for TS-lists. The result was another useful tool, BIG.CATALOG.DISPLAY.

BIG.CATALOG.DISPLAY has the same kind of output that FIND.TS.LISTS does, except that it also lists the file type, file name, and sector count from the catalog. Information is included for deleted files for which entries are found in the catalog, as well as all the active files.

The subroutines DISPLAY.TS.LIST, DISPLAY.NEXT.16, SEVEN.SPACES, PR.TS, and READTS are used without any changes from the FIND.TS.LISTS program. Instead of READ.NEXT.SECTOR, I have now READ.NEXT.CATALOG.SECTOR. This starts at track $11, sector $0F, and works back as far as sector $01. A better way might be to follow the actual chain, beginning in the VTOC sector, but the current scheme is easier and works with most of my disks.

Lines 1140-1180 set up the initial catalog track and sector. Lines 1190-1210 read the catalog sector. If the returned status is positive we did read a sector, and continue processing; if not, we are finished. Lines 1220-1250 set up the buffer address in the IOB for reading TS-lists and data sectors: we do not want to read them over the top of the catalog sector we are working with.

Lines 1270-1320 set up a loop for processing each of the seven file entries in the current catalog sector. The "NEXT" part of the loop is at lines 1350-1440. Each catalog entry takes 35 bytes, so lines 1350-1440 add 35 to the pointer.

DISPLAY.DATA.FOR.ONE.FILE first checks for a zero entry, meaning the end of the catalog. A catalog is initialized to all zeroes, so as soon as we find a zero entry we know there are no more files. Next, at lines 1520-1550, I check for a deleted file. If the track number is negative, it is a deleted file. The actual track number of a deleted file is saved on top of the 30th character of the file name, so I pick it up there. Lines 1560-1590 save the track and sector of the TS-list, so I can read it later.  Lines 1600-1650 display the file type as a hex value, followed by two dashes.

Lines 1660-1700 print the first 29 characters of the file name.  I don't print the last character because for a deleted file it will have been clobbered by saving the track number there.  Probably what I should do here is print either the last character for an active file, or some special symbol for a deleted file.  You can add that code if you like.

Lines 1710-1770 pick up the file size, in number of sectors, and print it as a hex value.  The sector count includes the sector for the TS-list.

Lines 1780-1860 read the track/sector list for the file.  If either the track number or the sector number is out of range, nothing is read and we skip any further processing for this file.

Lines 1870-1940 read in the first data sector for the file.  Again, if either the track or sector number is out of range, we don't try to read it.  Finally, lines 1950-2000 display the first 64 bytes of the file.

I hope you find these new tools as useful as I have.  Of course, I could hope you will never NEED them, but that would prabably be a vain hope.  I also hope you have "Bag of Tricks" or some similar utility to put it all back together after you get the information my tools provide.  And if I ever clobber Volume 1 on my Sider again (perish the thought), I intend to modify my copies of DOS so they will not allow me to INIT a volume on the Sider.

```
================================================================================
DOCUMENT :AAL-8604:DOS3.3:BCD.MAGIC.txt
================================================================================

 1000  *SAVE BCD MAGIC
 1010  *--------------------------------
 1020  CROUT  .EQ $FD8E
 1030  PRBYTE .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  *--------------------------------
 1060  VALUE  .EQ 0
 1070  *--------------------------------
 1080  T
 1090         LDA #0       FOR VALUE = 0 TO $FF
 1100  .1     STA VALUE
 1110         LDA #" "
 1120         JSR COUT
 1130         LDA VALUE
 1140         JSR PRBYTE
 1150  *--------------------------------
 1160         JSR IS.BCD.VALUE.DIVISIBLE.BY.FOUR
 1170         BEQ .2       ...YES
 1180         LDA #" "     ...NO
 1190         .HS 2C
 1200  .2     LDA #"*"
 1210         JSR COUT
 1220  *--------------------------------
 1230         LDA #" "     SEPARATE ITEMS IN CHART
 1240         JSR COUT
 1250         LDA VALUE    NEW LINE AFTER TEN VALUES
 1260         AND #$0F
 1270         CMP #9
 1280         BNE .3
 1290         JSR CROUT
 1300  *---NEXT VALUE-------------------
 1310  .3     SED          MUST DO ARITHMETIC
 1320         LDA VALUE       IN DECIMAL MODE
 1330         CLC
 1340         ADC #1
 1350         CLD          BACK TO BINARY
 1360         BCC .1       ...UNTIL WRAP-AROUND
 1370         RTS
 1380  *--------------------------------
 1390  IS.BCD.VALUE.DIVISIBLE.BY.FOUR
 1400         LDA VALUE    RETURN .EQ. STATUS IF YES
 1410         AND #$13            .NE. STATUS IF NOT
 1420         BEQ .1
 1430         EOR #$12
 1440  .1     RTS
 1450  *--------------------------------
 1460  DIVIDE.BCD.VALUE.BY.FOUR
 1470         LDA VALUE
 1480         JSR DIVIDE.BCD.VALUE.BY.TWO
```

```
DIVIDE.BCD.VALUE.BY.TWO
       PHA
       AND #$10
       BEQ .1
       PLA
       SBC #6
       LSR
       RTS
.1     PLA
       LSR
       RTS
*-------------------------------
SHORTER.DIV.BY.TWO
       LSR
       TAX
       AND #8
       BEQ .1
       DEX
       DEX
       DEX
.1     TXA
       RTS
*-------------------------------
D
       LDA #0       FOR VALUE = 0 TO $FF
.1     STA VALUE
       LDA #" "
       JSR COUT
       LDA VALUE
       JSR PRBYTE
       LDA #"."
       JSR COUT
*-------------------------------
       JSR DIVIDE.BCD.VALUE.BY.FOUR
       JSR PRBYTE
*-------------------------------
       LDA #" "     SEPARATE ITEMS IN CHART
       JSR COUT
       LDA VALUE    NEW LINE AFTER TEN VALUES
       AND #$0F
       CMP #9
       BNE .3
       JSR CROUT
*---NEXT VALUE------------------
.3     SED          MUST DO ARITHMETIC
       LDA VALUE       IN DECIMAL MODE
       CLC
       ADC #1
       CLD          BACK TO BINARY
       BCC .1       ...UNTIL WRAP-AROUND
       RTS
```

==============================================================================
DOCUMENT :AAL-8604:DOS3.3:DOS33.B700.B7FF.txt
==============================================================================

```
 1000  *SAVE S.B700-B7FF DOS 3.3
 1010  *--------------------------------
 1020  FMP.SUBCOD .EQ $B5BC
 1030  FMW.VOLUME .EQ $B5F9
 1040  RWTS       .EQ $BD00
 1050  *--------------------------------
 1060  INSTALL
 1070         LDY #0       COPY NEW CODE INTO DOS
 1080  .1     LDA NEW.B700,Y    $B700...B7FF
 1090         STA $B700,Y
 1100         INY
 1110         BNE .1
 1120         LDA #8       PATCH TO INCLUDE TRACK 2
 1130         STA $AEB3    AS FREE SPACE
 1140         RTS
 1150  *--------------------------------
 1160  NEW.B700  .PH $B700
 1170  *-------------------------------
 1180  BOOT.STAGE2
 1190         STX IOB.SLOT16
 1200         STX IOB.PRVSLT
 1210         TXA          SLOT*16
 1220         LSR          GET SLOT #
 1230         LSR
 1240         LSR
 1250         LSR
 1260         TAX          X = SLOT NUMBER
 1270  *---COPY BB00-FF TO 9D00-FF------
 1280         LDY #0
 1290  .1     LDA $BB00,Y
 1300         STA $9D00,Y
 1310         DEY
 1320         BNE .1
 1330  *---SET CURRENT TRACKS @ 0-------
 1340         TYA          A = Y = 0
 1350         STA $4F8,X
 1360         STA $478,X
 1370  *---BUILD RWFT CALL-------------
 1380         INY          Y = 1
 1390         STY IOB.PRVDRV
 1400         STY IOB.DRIVE     DRIVE = 1
 1410         TYA               A = 1 (READ OPCODE)
 1420         INY               Y = 1 (RWFT INDEX)
 1430         JSR RWFT
 1440  *---COLD START DOS--------------
 1450         LDX #$FF
 1460         TXS          EMPTY STACK
 1470         STX IOB.VOLUME
 1480         JMP $9D84    DOS HARD ENTRY
```

```
1490  *-------------------------------
1500         .BS $B74A-*       <<<FILLER>>>
1510  *--------------------------------
1520  *      WRITE DOS IMAGE ON TRACKS 0-2
1530  *--------------------------------
1540  WRITE.DOS.IMAGE
1550         LDA #2       WRITE OPCODE FOR RWTS
1560         LDY #0       RWFT INDEX
1570  *-------------------------------
1580  *   READ/WRITE FROM TABLE
1590  *-------------------------------
1600  RWFT
1610         STA IOB.OPCODE
1620  .1     STY RWFT.INDEX
1630         LDA RWFT.ADDR,Y
1640         BEQ .3       ...END OF RWFT TABLE
1650         STA IOB.BUFFER+1
1660         LDA RWFT.TRACK,Y
1670         STA IOB.TRACK
1680         LDA RWFT.SECTOR,Y
1690         STA IOB.SECTOR
1700         LDA RWFT.COUNT,Y
1710         STA RWFT.N
1720  .2     LDA /IOB
1730         LDY #IOB
1740         JSR ENTER.RWTS
1750         BCS .2       ...TRY AGAIN IF ERROR
1760         DEC IOB.SECTOR    NEXT SECTOR
1770         DEC IOB.BUFFER+1  NEXT PAGE
1780         DEC RWFT.N
1790         BNE .2
1800         LDY RWFT.INDEX
1810         INY
1820         BNE .1       ...ALWAYS
1830  .3     RTS
1840  *-------------------------------
1850  RWFT.N     .BS 1
1860  RWFT.INDEX .BS 1
1870  *-------------------------------
1880  RWFT.ADDR   .HS BF.9D.A3.B3.00
1890  RWFT.TRACK  .HS 00.00.00.01
1900  RWFT.SECTOR .HS 09.05.0F.0F
1910  RWFT.COUNT  .HS 0A.01.06.10
1920  *-------------------------------
1930         .BS $B7B5-*       <<<FILLER>>>
1940  *--------------------------------
1950  *      ENTER RWTS
1960  *--------------------------------
1970  ENTER.RWTS
1980         PHP          SAVE STATUS ON STACK
1990         SEI          DISABLE INTERRUPTS
2000         JSR RWTS     CALL RWTS
2010         BCS .1       ERROR RETURN
2020         PLP          RESTORE STATUS
```

```
2030        CLC          SIGNAL NO RWTS ERROR
2040        RTS          RETURN TO CALLER
2050 .1     PLP          RESTORE STATUS
2060        SEC          SIGNAL RWTS ERROR
2070        RTS          RETURN TO CALLER
2080 *---------------------------------
2090 *      SET UP RWTS TO WRITE DOS
2100 *---------------------------------
2110 SETUP.WRITE.DOS
2120        LDA FMP.SUBCOD  IMAGE ADDRESS
2130        STA IOB.BUFFER+1
2140        LDA #0
2150        STA IOB.BUFFER
2160        LDA FMW.VOLUME  VOLUME #
2170        EOR #$FF        UNCOMPLEMENT IT
2180        STA IOB.VOLUME
2190        RTS
2200 *---------------------------------
2210 *      CLEAR 256 BYTES STARTING AT ($42,43)
2220 *---------------------------------
2230 ZERO.CURRENT.BUFFER
2240        LDA #0
2250        TAY
2260 .1     STA ($42),Y
2270        INY
2280        BNE .1
2290        RTS
2300 *---------------------------------
2310        .BS $B7E8-*       <<<FILLER>>>
2320 *---------------------------------
2330 *      IOB FOR RWTS CALLS
2340 *---------------------------------
2350 IOB
2360 IOB.TYPE   .HS 01   0--MUST BE $01
2370 IOB.SLOT16 .HS 60   1--SLOT # TIMES 16
2380 IOB.DRIVE  .HS 01   2--DRIVE # (1 OR 2)
2390 IOB.VOLUME .HS 00   3--DESIRED VOL # (0 MATCHES ANY)
2400 IOB.TRACK  .BS 1    4--TRACK # (0 TO 34)
2410 IOB.SECTOR .BS 1    5--SECTOR # (0 TO 15)
2420 IOB.PNTDCT .DA DCT  6--ADDRESS OF DCT
2430 IOB.BUFFER .BS 2    8--ADDRESS OF DATA
2440 IOB.SECTSZ .DA 256 10--# BYTES IN A SECTOR
2450 IOB.OPCODE .BS 1   12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
2460 IOB.ERROR  .BS 1   13--ERROR CODE: 0, 8, 10, 20, 40, 80
2470 IOB.ACTVOL .BS 1   14--ACTUAL VOLUME # FOUND
2480 IOB.PRVSLT .HS 60  15--PREVIOUS SLOT #
2490 IOB.PRVDRV .HS 01  16--PREVIOUS DRIVE #
2500            .BS 2
2510 DCT    .HS 0001EFD8
2520        .BS 1
2530 *--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8604:DOS3.3:S.BigCatDisp.txt
==============================================================================

 1000  *SAVE S.BIG CATALOG DISPLAY
 1010  *-------------------------------
 1020  CAT.SECTOR .EQ 0
 1030  CAT.TRACK  .EQ 1
 1040  CNTR       .EQ 2
 1050  PNTR       .EQ 3,4
 1060  TS.TRACK   .EQ 5
 1070  TS.SECTOR  .EQ 6
 1080  *--------------------------------
 1090  COUT   .EQ $FDED
 1100  CROUT  .EQ $FD8E
 1110  PRBYTE .EQ $FDDA
 1120  ENTER.RWTS .EQ $3D9
 1130  *-------------------------------
 1140  BIG.CATALOG.DISPLAY
 1150         LDA #15
 1160         STA CAT.SECTOR
 1170         LDA #17
 1180         STA CAT.TRACK
 1190  .1     JSR READ.NEXT.CATALOG.SECTOR
 1200         BPL .2       GOT A SECTOR
 1210  .4     RTS
 1220  .2     LDA #BUF
 1230         STA IOB.BUFFER
 1240         LDA /BUF
 1250         STA IOB.BUFFER+1
 1260  *-------------------------------
 1270         LDA #CAT+11
 1280         STA PNTR
 1290         LDA /CAT+11
 1300         STA PNTR+1
 1310         LDA #7
 1320         STA CNTR
 1330  .3     JSR DISPLAY.DATA.FOR.ONE.FILE
 1340         BCS .4       ...END OF CATALOG
 1350         LDA PNTR
 1360         ADC #35
 1370         STA PNTR
 1380         LDA PNTR+1
 1390         ADC #0
 1400         STA PNTR+1
 1410         DEC CNTR
 1420         BNE .3
 1430         JSR CROUT
 1440         JMP .1
 1450  *-------------------------------
 1460  DISPLAY.DATA.FOR.ONE.FILE
 1470         LDY #0
 1480         LDA (PNTR),Y
```

```
1490        BNE .1
1500        SEC
1510        RTS
1520 .1     BPL .15
1530        LDY #32
1540        LDA (PNTR),Y      REAL TRACK OF DELETED FILE
1550        LDY #0
1560 .15    STA TS.TRACK
1570        INY
1580        LDA (PNTR),Y
1590        STA TS.SECTOR
1600        INY
1610        LDA (PNTR),Y      GET FILE TYPE
1620        JSR PRBYTE
1630        LDA #"-"
1640        JSR COUT
1650        JSR COUT
1660 .2     INY
1670        LDA (PNTR),Y      PRINT FILE NAME
1680        JSR COUT
1690        CPY #31           DON'T PRINT LAST CHAR OF NAME
1700        BCC .2
1710        INY
1720        INY
1730        LDA (PNTR),Y
1740        JSR PRBYTE
1750        INY
1760        LDA (PNTR),Y
1770        JSR PRBYTE
1780 *---READ T/S LIST----------------
1790        LDX TS.SECTOR
1800        CPX #16
1810        BCS .9
1820        LDY TS.TRACK
1830        CPY #35
1840        BCS .9
1850        JSR READTS
1860        JSR DISPLAY.TS.LIST
1870 *---READ FIRST DATA SECTOR-------
1880        LDY BUF+12
1890        CPY #35
1900        BCS .9
1910        LDX BUF+13
1920        CPX #16
1930        BCS .9
1940        JSR READTS
1950 *---DISPLAY FIRST 64 BYTES-------
1960        LDY #0
1970        JSR DISPLAY.NEXT.16
1980        JSR DISPLAY.NEXT.16
1990        JSR DISPLAY.NEXT.16
2000        JSR DISPLAY.NEXT.16
2010 .9     JSR CROUT
2020        CLC
```

```
2030        RTS
2040 *--------------------------------
2050 DISPLAY.TS.LIST
2055   .LIST OFF
2060        JSR CROUT
2070        LDA TS.TRACK
2080        LDX TS.SECTOR
2090        JSR PR.TS
2100        LDA #":"
2110        JSR COUT
2120        LDA #" "
2130        JSR COUT
2140        JSR COUT
2150        LDY #0
2160 .1     LDA BUF+13,Y      SECTOR
2170        TAX
2180        LDA BUF+12,Y      TRACK
2190        BEQ .2       ...END OF LIST
2200        JSR PR.TS
2210        LDA #" "
2220        JSR COUT
2230        TYA
2240        AND #$0F
2250        CMP #$0E
2260        BNE .3
2270        JSR SEVEN.SPACES
2280 .3     INY
2290        INY
2300        CPY #-12
2310        BCC .1
2320 .2     RTS
2325    .LIST ON
2330 *--------------------------------
2340 DISPLAY.NEXT.16
2345    .LIST OFF
2350        JSR SEVEN.SPACES
2360 .1     LDA BUF,Y
2370        JSR PRBYTE
2380        LDA #" "
2390        JSR COUT
2400        INY
2410        TYA
2420        AND #$0F
2430        BNE .1
2440        TYA
2450        SEC
2460        SBC #16
2470        TAY
2480 .2     LDA BUF,Y
2490        ORA #$80
2500        CMP #$A0
2510        BCS .3
2520        LDA #"."
2530 .3     JSR COUT
```

```
2540        INY
2550        TYA
2560        AND #$0F
2570        BNE .2
2580        RTS
2585   .LIST ON
2590  *--------------------------------
2600  SEVEN.SPACES
2605   .LIST OFF
2610        JSR CROUT
2620        LDA #" "
2630        LDX #7
2640  .4    JSR COUT
2650        DEX
2660        BNE .4
2670        RTS
2675   .LIST ON
2680  *--------------------------------
2690  PR.TS
2695   .LIST OFF
2700        JSR PRBYTE
2710        LDA #"-"
2720        JSR COUT
2730        TXA
2740        ORA #"0"
2750        CMP #$BA
2760        BCC .1
2770        ADC #6
2780  .1    JMP COUT
2785   .LIST ON
2790  *--------------------------------
2800  *   READ NEXT CATALOG SECTOR
2810  *--------------------------------
2820  READ.NEXT.CATALOG.SECTOR
2830        LDA #CAT
2840        STA IOB.BUFFER
2850        LDA /CAT
2860        STA IOB.BUFFER+1
2870        LDX CAT.SECTOR
2880        LDY CAT.TRACK
2890        JSR READTS
2900        DEC CAT.SECTOR
2910        RTS
2920  *--------------------------------
2930  READTS STX IOB.SECTOR
2935    .LIST OFF
2940        STY IOB.TRACK
2950  .2    LDA /IOB
2960        LDY #IOB
2970        JSR ENTER.RWTS
2980        BCS .2       ...TRY AGAIN IF ERROR
2990        RTS
2995    .LIST ON
3000  *---------------------------------
```

```
3010  *      IOB FOR RWTS CALLS
3020  *---------------------------------
3030  IOB
3035     .LIST OFF
3040  IOB.TYPE   .HS 01   0--MUST BE $01
3050  IOB.SLOT16 .HS 60   1--SLOT # TIMES 16
3060  IOB.DRIVE  .HS 01   2--DRIVE # (1 OR 2)
3070  IOB.VOLUME .HS 00   3--DESIRED VOL # (0 MATCHES ANY)
3080  IOB.TRACK  .BS 1    4--TRACK # (0 TO 34)
3090  IOB.SECTOR .BS 1    5--SECTOR # (0 TO 15)
3100  IOB.PNTDCT .DA DCT  6--ADDRESS OF DCT
3110  IOB.BUFFER .DA BUF  8--ADDRESS OF DATA
3120  IOB.SECTSZ .DA 256 10--# BYTES IN A SECTOR
3130  IOB.OPCODE .HS 01  12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
3140  IOB.ERROR  .BS 1   13--ERROR CODE: 0, 8, 10, 20, 40, 80
3150  IOB.ACTVOL .BS 1   14--ACTUAL VOLUME # FOUND
3160  IOB.PRVSLT .HS 60  15--PREVIOUS SLOT #
3170  IOB.PRVDRV .HS 01  16--PREVIOUS DRIVE #
3175     .LIST ON
3180  *--------------------------------
3190  DCT    .HS 0001EFD8
3200  *--------------------------------
3210  BUF    .BS 256
3220  CAT    .BS 256
3230  *--------------------------------
3235     .LIST OFF
```

```
==========================================================================
DOCUMENT :AAL-8604:DOS3.3:S.Find.TS.Lists.txt
==========================================================================

 1000  *SAVE S.FIND T/S LISTS
 1010  *---------------------------------
 1020  CUR.SECTOR .EQ 0
 1030  CUR.TRACK  .EQ 1
 1040  *----------------------------------
 1050  COUT   .EQ $FDED
 1060  CROUT  .EQ $FD8E
 1070  PRBYTE .EQ $FDDA
 1080  ENTER.RWTS .EQ $3D9
 1090  *---------------------------------
 1100  FIND.TS.LISTS
 1110         LDA #0
 1120         STA CUR.SECTOR
 1130         STA CUR.TRACK
 1140  .1     JSR READ.NEXT.SECTOR
 1150         BCC .2       GOT A SECTOR, CHECK IT
 1160         RTS          END OF DISK, QUIT
 1170  *---CHECK IF THIS IS T/S LIST----
 1180  .2     LDA BUF+12   TRACK # FOR FIRST DATA SECTOR
 1190         BEQ .1       ...NO, TRY NEXT ONE
 1200         LDY #12
 1210  .3     LDA BUF,Y
 1220         CMP #35
 1230         BCS .1       ...NOT VALID TRACK
 1240         INY
 1250         LDA BUF,Y
 1260         CMP #16
 1270         BCS .1       ...NOT VALID SECTOR
 1280         INY
 1290         BNE .3       ...MORE IN SECTOR TO CHECK
 1300  *---DISPLAY THE T/S LIST---------
 1310         JSR DISPLAY.TS.LIST
 1320  *---READ FIRST DATA SECTOR-------
 1330         LDY BUF+12
 1340         LDX BUF+13
 1350         JSR READTS
 1360  *---DISPLAY FIRST 64 BYTES-------
 1370         LDY #0
 1380         JSR DISPLAY.NEXT.16
 1390         JSR DISPLAY.NEXT.16
 1400         JSR DISPLAY.NEXT.16
 1410         JSR DISPLAY.NEXT.16
 1420         JSR CROUT
 1430         JMP .1
 1440  *---------------------------------
 1450  DISPLAY.TS.LIST
 1460         JSR CROUT
 1470         LDA CUR.TRACK
 1480         LDX CUR.SECTOR
```

```
1490        JSR PR.TS
1500        LDA #":"
1510        JSR COUT
1520        LDA #" "
1530        JSR COUT
1540        JSR COUT
1550        LDY #0
1560 .1     LDA BUF+13,Y      SECTOR
1570        TAX
1580        LDA BUF+12,Y      TRACK
1590        BEQ .2       ...END OF LIST
1600        JSR PR.TS
1610        LDA #" "
1620        JSR COUT
1630        TYA
1640        AND #$0F
1650        CMP #$0E
1660        BNE .3
1670        JSR SEVEN.SPACES
1680 .3     INY
1690        INY
1700        CPY #-12
1710        BCC .1
1720 .2     RTS
1730 *---------------------------------
1740 DISPLAY.NEXT.16
1750        JSR SEVEN.SPACES
1760 .1     LDA BUF,Y
1770        JSR PRBYTE
1780        LDA #" "
1790        JSR COUT
1800        INY
1810        TYA
1820        AND #$0F
1830        BNE .1
1840        TYA
1850        SEC
1860        SBC #16
1870        TAY
1880 .2     LDA BUF,Y
1890        ORA #$80
1900        CMP #$A0
1910        BCS .3
1920        LDA #"."
1930 .3     JSR COUT
1940        INY
1950        TYA
1960        AND #$0F
1970        BNE .2
1980        RTS
1990 *---------------------------------
2000 SEVEN.SPACES
2010        JSR CROUT
2020        LDA #" "
```

```
2030        LDX #7
2040 .4     JSR COUT
2050        DEX
2060        BNE .4
2070        RTS
2080 *---------------------------------
2090 PR.TS
2100        JSR PRBYTE
2110        LDA #"-"
2120        JSR COUT
2130        TXA
2140        ORA #"0"
2150        CMP #$BA
2160        BCC .1
2170        ADC #6
2180 .1     JMP COUT
2190 *---------------------------------
2200 *   READ NEXT SECTOR
2210 *---------------------------------
2220 READ.NEXT.SECTOR
2230        LDX CUR.SECTOR
2240        LDY CUR.TRACK
2250        DEX          NEXT SECTOR
2260        BPL .1       ...SAME TRACK
2270        LDX #15      ...NEXT TRACK
2280        INY
2290        CPY #35
2300        BCS .2       ...END OF DISK
2310 .1     STY CUR.TRACK
2320        STX CUR.SECTOR
2330        JSR READTS
2340        CLC
2350 .2     RTS
2360 *---------------------------------
2370 READTS STX IOB.SECTOR
2380        STY IOB.TRACK
2390 .2     LDA /IOB
2400        LDY #IOB
2410        JSR ENTER.RWTS
2420        BCS .2       ...TRY AGAIN IF ERROR
2430        RTS
2440 *----------------------------------
2450 *      IOB FOR RWTS CALLS
2460 *----------------------------------
2470 IOB
2480 IOB.TYPE   .HS 01   0--MUST BE $01
2490 IOB.SLOT16 .HS 60   1--SLOT # TIMES 16
2500 IOB.DRIVE  .HS 01   2--DRIVE # (1 OR 2)
2510 IOB.VOLUME .HS 00   3--DESIRED VOL # (0 MATCHES ANY)
2520 IOB.TRACK  .BS 1    4--TRACK # (0 TO 34)
2530 IOB.SECTOR .BS 1    5--SECTOR # (0 TO 15)
2540 IOB.PNTDCT .DA DCT  6--ADDRESS OF DCT
2550 IOB.BUFFER .DA BUF  8--ADDRESS OF DATA
2560 IOB.SECTSZ .DA 256 10--# BYTES IN A SECTOR
```

```
IOB.OPCODE .HS 01  12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
IOB.ERROR  .BS 1   13--ERROR CODE: 0, 8, 10, 20, 40, 80
IOB.ACTVOL .BS 1   14--ACTUAL VOLUME # FOUND
IOB.PRVSLT .HS 60  15--PREVIOUS SLOT #
IOB.PRVDRV .HS 01  16--PREVIOUS DRIVE #
*--------------------------------
DCT    .HS 0001EFD8
*--------------------------------
BUF    .BS 256
*--------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8604:DOS3.3:S.Msg.Into.Wind.txt
==============================================================================

 1000  *SAVE S.MSG INTO WINDOW
 1010  *--------------------------------
 1020  HOME   .EQ $FC58
 1030  COUT   .EQ $FDED
 1040  *--------------------------------
 1050  PNTR   .EQ $00,01
 1060  WNDTOP .EQ $22
 1070  WNDBOT .EQ $23
 1080  *--------------------------------
 1090  *   CALL:   JSR MSG.IN.WINDOW
 1100  *           .DA #<window number>
 1110  *           .AS text of message
 1120  *           .HS 00    <end of msg flag>
 1130  *--------------------------------
 1140  MSG.IN.WINDOW
 1150         PLA          GET RETURN ADDRESS INTO PNTR
 1160         STA PNTR     LO BYTE
 1170         PLA
 1180         STA PNTR+1   HI BYTE
 1190  *---SETUP WINDOW TOP & BOTTOM----
 1200         JSR GET.NEXT.CALL.BYTE
 1210         TAX          WINDOW INDEX
 1220         LDA WINDOW.DATA,X
 1230         STA WNDTOP
 1240         LDA WINDOW.DATA+1,X
 1250         STA WNDBOT
 1260         JSR HOME     CLEAR THE WINDOW
 1270  *---DISPLAY MESSAGE, IF ANY------
 1280         LDY #0
 1290  .1     JSR GET.NEXT.CALL.BYTE
 1300         BEQ .2       END OF MESSAGE
 1310         ORA #$80     ...JUST IN CASE
 1320         JSR COUT
 1330         JMP .1
 1340  *---RETURN TO CALLER-------------
 1350  .2     LDA PNTR+1   HI BYTE
 1360         PHA
 1370         LDA PNTR     LO BYTE
 1380         PHA
 1390         RTS
 1400  *--------------------------------
 1410  GET.NEXT.CALL.BYTE
 1420         INC PNTR     LO BYTE
 1430         BNE .1
 1440         INC PNTR+1   HI BYTE
 1450  .1     LDA (PNTR),Y
 1460         RTS
 1470  *--------------------------------
 1480  WINDOW.DATA
```

```
       .DA #0,#24,#0,#3,#9,#18,#20,#24
*--------------------------------
T
       JSR MSG.IN.WINDOW
       .DA #2       TOP WINDOW
       .AS -/TOP LINE OF THE SCREEN/
       .HS 8D
       .AS -/SECOND LINE OF THE SCREEN/
       .HS 8A
       .AS -/...AND THE THIRD/
       .HS 00       END MSG
       JSR W
       JSR MSG.IN.WINDOW
       .DA #6       BOTTOM WINDOW
       .AS -/LINE 21/
       .HS 8A
       .AS -/...LINE 22/
       .HS 8A.8A
       .AS -/...AND LINE 24/
       .HS 00       END MSG
       JSR W
       JSR MSG.IN.WINDOW
       .DA #0       FULL SCREEN
       .AS -/MY MESSAGE/
       .HS 00       END MSG
       RTS
*--------------------------------
W      LDA $C000    WAIT FOR KEY BEFORE CONTINUING
       BPL W
       STA $C010
       RTS
*--------------------------------
```

================================================================================
DOCUMENT :AAL-8604:ProDOS:BCD.MAGIC.txt
================================================================================

```
 1000  *SAVE BCD.MAGIC
 1010  *--------------------------------
 1020  CROUT  .EQ $FD8E
 1030  PRBYTE .EQ $FDDA
 1040  COUT   .EQ $FDED
 1050  *--------------------------------
 1060  VALUE  .EQ 0
 1070  *--------------------------------
 1080  T
 1090         LDA #0       FOR VALUE = 0 TO $FF
 1100  .1     STA VALUE
 1110         LDA #" "
 1120         JSR COUT
 1130         LDA VALUE
 1140         JSR PRBYTE
 1150  *--------------------------------
 1160         JSR IS.BCD.VALUE.DIVISIBLE.BY.FOUR
 1170         BEQ .2       ...YES
 1180         LDA #" "     ...NO
 1190         .HS 2C
 1200  .2     LDA #"*"
 1210         JSR COUT
 1220  *--------------------------------
 1230         LDA #" "     SEPARATE ITEMS IN CHART
 1240         JSR COUT
 1250         LDA VALUE    NEW LINE AFTER TEN VALUES
 1260         AND #$0F
 1270         CMP #9
 1280         BNE .3
 1290         JSR CROUT
 1300  *---NEXT VALUE-------------------
 1310  .3     SED          MUST DO ARITHMETIC
 1320         LDA VALUE       IN DECIMAL MODE
 1330         CLC
 1340         ADC #1
 1350         CLD          BACK TO BINARY
 1360         BCC .1       ...UNTIL WRAP-AROUND
 1370         RTS
 1380  *--------------------------------
 1390  IS.BCD.VALUE.DIVISIBLE.BY.FOUR
 1400         LDA VALUE    RETURN .EQ. STATUS IF YES
 1410         AND #$13            .NE. STATUS IF NOT
 1420         BEQ .1
 1430         EOR #$12
 1440  .1     RTS
 1450  *--------------------------------
 1460  DIVIDE.BCD.VALUE.BY.FOUR
 1470         LDA VALUE
 1480         JSR DIVIDE.BCD.VALUE.BY.TWO
```

```
DIVIDE.BCD.VALUE.BY.TWO
       PHA
       AND #$10
       BEQ .1
       PLA
       SBC #6
       LSR
       RTS
.1     PLA
       LSR
       RTS
*--------------------------------
SHORTER.DIV.BY.TWO
       LSR
       TAX
       AND #8
       BEQ .1
       DEX
       DEX
       DEX
.1     TXA
       RTS
*--------------------------------
D
       LDA #0       FOR VALUE = 0 TO $FF
.1     STA VALUE
       LDA #" "
       JSR COUT
       LDA VALUE
       JSR PRBYTE
       LDA #"."
       JSR COUT
*--------------------------------
       JSR DIVIDE.BCD.VALUE.BY.FOUR
       JSR PRBYTE
*--------------------------------
       LDA #" "     SEPARATE ITEMS IN CHART
       JSR COUT
       LDA VALUE    NEW LINE AFTER TEN VALUES
       AND #$0F
       CMP #9
       BNE .3
       JSR CROUT
*---NEXT VALUE-------------------
.3     SED          MUST DO ARITHMETIC
       LDA VALUE       IN DECIMAL MODE
       CLC
       ADC #1
       CLD          BACK TO BINARY
       BCC .1       ...UNTIL WRAP-AROUND
       RTS
```

```
================================================================
DOCUMENT :AAL-8604:ProDOS:S.MSG.INTO.WNDW.txt
================================================================
```

```
1000  *SAVE S.MSG.INTO.WNDW
1010  *--------------------------------
1020  HOME   .EQ $FC58
1030  COUT   .EQ $FDED
1040  *--------------------------------
1050  PNTR   .EQ $00,01
1060  WNDTOP .EQ $22
1070  WNDBOT .EQ $23
1080  *--------------------------------
1090  *   CALL:   JSR MSG.IN.WINDOW
1100  *           .DA #<window number>
1110  *           .AS text of message
1120  *           .HS 00    <end of msg flag>
1130  *--------------------------------
1140  MSG.IN.WINDOW
1150         PLA          GET RETURN ADDRESS INTO PNTR
1160         STA PNTR     LO BYTE
1170         PLA
1180         STA PNTR+1   HI BYTE
1190  *---SETUP WINDOW TOP & BOTTOM----
1200         JSR GET.NEXT.CALL.BYTE
1210         TAX          WINDOW INDEX
1220         LDA WINDOW.DATA,X
1230         STA WNDTOP
1240         LDA WINDOW.DATA+1,X
1250         STA WNDBOT
1260         JSR HOME     CLEAR THE WINDOW
1270  *---DISPLAY MESSAGE, IF ANY------
1280         LDY #0
1290  .1     JSR GET.NEXT.CALL.BYTE
1300         BEQ .2       END OF MESSAGE
1310         ORA #$80     ...JUST IN CASE
1320         JSR COUT
1330         JMP .1
1340  *---RETURN TO CALLER-------------
1350  .2     LDA PNTR+1   HI BYTE
1360         PHA
1370         LDA PNTR     LO BYTE
1380         PHA
1390         RTS
1400  *--------------------------------
1410  GET.NEXT.CALL.BYTE
1420         INC PNTR     LO BYTE
1430         BNE .1
1440         INC PNTR+1   HI BYTE
1450  .1     LDA (PNTR),Y
1460         RTS
1470  *--------------------------------
1480  WINDOW.DATA
```

```
1490        .DA #0,#24,#0,#3,#9,#18,#20,#24
1500 *--------------------------------
1510 T
1520        JSR MSG.IN.WINDOW
1530        .DA #2      TOP WINDOW
1540        .AS -/TOP LINE OF THE SCREEN/
1550        .HS 8D
1560        .AS -/SECOND LINE OF THE SCREEN/
1570        .HS 8A
1580        .AS -/...AND THE THIRD/
1590        .HS 00      END MSG
1600        JSR W
1610        JSR MSG.IN.WINDOW
1620        .DA #6      BOTTOM WINDOW
1630        .AS -/LINE 21/
1640        .HS 8A
1650        .AS -/...LINE 22/
1660        .HS 8A.8A
1670        .AS -/...AND LINE 24/
1680        .HS 00      END MSG
1690        JSR W
1700        JSR MSG.IN.WINDOW
1710        .DA #0      FULL SCREEN
1720        .AS -/MY MESSAGE/
1730        .HS 00      END MSG
1740        RTS
1750 *--------------------------------
1760 W      LDA $C000   WAIT FOR KEY BEFORE CONTINUING
1770        BPL W
1780        STA $C010
1790        RTS
1800 *--------------------------------
```

==============================================================================
DOCUMENT :AAL-8605:Articles:Bartletts.Searc.txt
==============================================================================

Recovering & Repairing Lost Programs......Peter Bartlett, Jr.
Eldridge, Iowa

As a long-time user of the S-C Macro Assembler, I have learned a few tricks to save a lot of aggravation. Sometimes I mistakenly erase the source program I have in memory with the "NEW" or "LOAD" command. The program is not actually gone; instead, the pointer to the start of the program is changed.

At one time, I would adjust the source pointer by hand until my program was restored, but this was slow and painful. So like all good hackers I now have a little program to find the start of a program and adjust the pointer automatically.

My "Find.Start" program searches through memory for a source line numbered 1000 and resets the source pointer to that line. The search begins at HIMEM and proceeds down until it finds line 1000 or address $800.

The program itself is a simple search for the two-byte hex equivalent of 1000. On entry, the program starts the search at HIMEM and sets the "DONE.ONCE" flag so subsequent re-entries pick up the search where it last left off.

After the program stops, you can run it again to find the next lower source line numbered 1000. If several programs have been loaded into memory, you can run "Find.Start" several times to point to the start of each one.

The only way to start the search from HIMEM again is to re-load the program. It's not elegant, but does it really need to be?

In many instances, the next step is to re-construct the scrambled part of a program. This usually seems impossible, because the program's internal pointers will probably be scrambled and cause weird problems when editing.

Instead of fighting with the program (or hand-patching as I used to do), just use the handy "TEXT" command built into the assembler to create a text version of your program. Then enter the "AUTO" mode and "EXEC" the text version of your program back into memory. This will rectify all the internal pointers and leave you free to edit your program back into shape.

Perhaps that last paragraph is obvious, but I didn't think of it until recently. And we've had the "TEXT" command available for a long time!

==============================================================================
DOCUMENT :AAL-8605:Articles:Division.By7.txt
==============================================================================

More and Better Division by Seven.........Bob Sander-Cederlof

I can think of at least three good reasons we need a good subroutine for dividing by seven. We need it in computations involving the day of week. We need it in hi-res graphics programs to calculate the byte and bit for a particular pixel between 0 and 279 for normal hi-res, or between 0 and 559 for double hi-res. Lastly, the new protocol converter interface used in connection with the Unidisk 3.5 works with packets of up to 767 bytes which are made up of a number of 7-byte groups.

In looking through the assembly listing of the new //c ROMs, which come with the Unidisk 3.5 update, I noticed a divide-by- seven subroutine at $CB45-CBAF. The code divides the buffer size, which can be up to $2FF, by seven, and saves both the quotient and the remainder. The code looks too large and too slow and too complicated ... in other words, it looks like a challenging assignment. My transposition of the //c code follows, and as I count cycles it takes from 133 to 268 cycles depending on the value of the dividend. The code and tables take 71 bytes in the //c ROM.

While I was musing on the possibilities, Michael Hackney called me from Troy, New York. He wondered if we were interested in publishing his fast 65802 routine for dividing by seven. Michael uses his in a speedy double hi-res program. He divides values up to 559 ($22F) by seven, keeping both the quotient and remainder, in 66 cycles. Michael's subroutine itself is short (37 bytes), but he uses a 140-byte table to achieve the speed. Adding another 84 bytes to the tables extends the range to handle dividends up to 895 ($37F).

(In all the times and lengths given here, I am not counting the JSR-RTS cycles nor the RTS byte. I assume the code is critical enough that it would be placed in-line in actual use, rather than made into a JSR-called subroutine. I am also not counting any overhead I added to switch from 65802 mode to 6502 and back, as this was only added due to my test program being in 65802 mode. All of the subroutines use page zero for variable and temporary storage. They would be longer and slightly slower if the variables and temporaries were not in page zero.)

Yesterday I spent the whole day dividing by seven. I came up with two new subroutines: one for the 65802, and one for a normal 6502. They are both small and fast. First I tackled the 65802 version, and based in on multiplying by 1/7 as a binary fraction. This one came out 39 bytes long, executing in 64 cycles. This one used a fudge factor; the largest dividend it can handle is 594 ($252). By using alternate code to extend the precision, numbers up to 895 ($37F) can be handled. This one takes the same number of bytes, but 9 cycles longer.

Finally, I wrote a normal 6502 version. Strangely enough, it came out only 60 bytes long and only 76 cycles! Makes me wonder if I couldn't do better in the 65802, given another day or two. The 6502 version handles dividends up to 1023 ($3FF). It would be two bytes shorter if the range was restricted to $2FF.

Here is a table summarizing the size, timing, and dividend range for the various subroutines:

| | bytes | cycles | dividend |
|---|---|---|---|
| //c ROM | 71 | 133-268 | 0-$2FF |
| Hackney 65802 | 177 | 66 | 0-$22F |
| RBSC 65802-1 | 39 | 64 | 0-$252 |
| RBSC 65802-2 | 39 | 73 | 0-$37F |
| RBSC 6502 | 60 | 76 | 0-$3FF |

The listing which follows includes all five versions, plus a testing program. The testing program runs through the entire range from $3FF down to 0. After doing the division by the selected method, a check subroutine tests for a valid remainder (a number less than 7); it further tests that the quotient*7 +remainder = the original dividend. If not, the dividend, quotient, and remainder are all printed in hexadecimal. If they are correct, the next dividend is tried. A keyboard pausing subroutine allows you to stop the display momentarily and/or abort the test run.

Lines 1020-1060 control some conditional assembly which select which division method to use. By change the value of VERSION in line 1020 I can assemble any one of the four routines. I used the "CON" listing option in line 1180 (which is not itself listed: it is "1180    .LIST CON") so that you can see what the un-assembled lines of code are. Other conditional code at lines 1720-1860 and 4010-4050 selects options mentioned above.

Lines 1200-1540 control each test run. I wrote this program using 65802 instructions, although it would not be difficult to re-write it for a plain 6502. Lines 1210-1220 enter the 65802 Native Mode, and lines 1520-1530 leave it. It is VERY IMPORTANT to be sure you do not exit a program and return to normal Apple software while still in the Native Mode. The most fantastic things can happen if you forget!

Lines 1580-1950 are my 65802 version. This entire subroutine is executed in the 65802 native mode, with the M-bit set so the A-register operations are 16-bits. The value 1/7 in binary is .001001001001001...forever. Multiplying by than number should give the same answer as dividing by seven. It also has the surprising side effect that the three bits after the "quotient" portion of the product will be equal to the "remainder". The values of the fractions from 0/7 to 6/7 are just nice that way:

| fraction | repeating decimal | same value in hex | the first three bits |
|---|---|---|---|
| 0/7 | .000000 | .000 | 000 |

```
        1/7    .142857..   .249..        001
        2/7    .285714..   .492..        010
        3/7    .428571..   .6DB..        011
        4/7    .571428..   .924..        100
        5/7    .714285..   .B6D..        101
        6/7    .857142..   .DB6..        110
```

Wow!  Isn't that neat?  More justification for the numerologists who claim that seven is the "perfect" number.

Now it remains to find the most efficient way to multiply by that fraction.  The method I came up with first forms the product for .01000001 (lines 1600-1670).  Then I divide that result by 8, which is the product for .00001000001 (lines 1680- 1700).  Adding the two products in line 1710 gives me the product for .01001001001 (approximately 2/7).  Dividing that by two gives me an approximation for the division by seven.  The code that follows in lines 1720-1800 is not assembled, because of the ".DO 0" line.  What it does is extend the multiplication to include one more partial product.  The shortest way I could think of to get that little number is demonstrated in the code you see.  The extra precision makes my subroutine work for dividends up to $37F.  It fails above that value because of overflow during the multiplication.  If I leave out the extra precision, the subroutine gets the wrong answers for some numbers at each end of the range.  By adding a "fudge factor" (a trick learned in college laboratory assignments to force experimental results to fit the laws of science), I can make all the dividends up to $252 work.  The fudge factor adds $000A for values in the A-register of $8800 or more, and only $0008 for values below $8800.

Line 1870 is the division by two mentioned above.  Lines 1880-1940 shift the first three bits of the remainder over to the correct position in the lower byte of the A-register.  As I was writing the previous sentence, it suddenly struck me that the second set of three bits might be the same as the first set, if my multiplications happened to be precise enough.  I went back to the assembler, changed line 1720 to ".DO 1" so the more precise version would assemble, and then replaced lines 1910-1930 with "1910    AND #7".  Guess what!  It worked!  One byte shorter and four cycles faster!  That makes it 38 bytes long, and only 69 cycles.

Next is my 6502 version, lines 1970-2370.  The first four lines simply save the current state of the M and X bits, and the mode, and switch to 6502 emulation mode.  They are matched by lines 2340-2360, which restore the mode and state.  These will work regardless of what mode and state the machine was in when the subroutine was called.  Since the subroutine would normally only be used in a 6502, you would leave out lines 1980-2010 and 2340-2360.  I did not count them when timing the code.  Back in December of 1984 I wrote in these pages of a nifty way to divide a one-byte value by seven.  I used that method here, for dividing the low-order byte of the dividend.  I then computed the remainder by multiplying the quotient by 7 and subtracting it from the dividend.  Saving that quotient and remainder, I used a table lookup to determine the quotient and remainder of the high-order byte of the number. Since it could only have the values 0-3, the tables are very short. Then I add the two remainders together, modulo 7; and the two quotients, remembering the carry from the remainder if any.

Lines 2030-2170 are essentially the same as published in that December issue of AAL, except for the addition of lines 2130, 2140, and 2160. With those two lines I am saving a few steps in the multiplication by seven that I must do. Lines 2190-2200 finish the multiplication by seven, by adding the *2 and *4 values saved above. Lines 2210-2200 form the complement of the value, so I can subtract by adding. Normally a complement is formed by:

```
       EOR #$FF
       CLC
       ADC #1
```

I do the same with two less bytes and cycles here by preceding the addition at line 2230 with SEC rather than the usual CLC. I saved a byte and two cycles by storing one less than the actual remainder in the table of remainders at line 2400.

Lines 2420-2640 are called to print out the results when they don't meet expectations. Notice lines 2430-2460 and 2610-2630, which make sure I am in the correct state and mode. The monitor routines will not work correctly in 16-bit state, and may not work correctly in 65802 Native mode.

Lines 2660-2920 check the results. The subroutine returns with carry clear if the quotient and remainder are correct, or carry set if they are not. I check both by multiplying the quotient by seven and adding the remainder to see if the result equals the dividend, and I also make sure the remainder is less than seven. It is possible to get an answer with the quotient one less than it should be and a remainder of 7, so I had to test the remainder.

The PAUSE routine checks to see if any key has been typed. If so, and if it is not a <RETURN>, it waits until another key is typed. Note that I had to set 8-bit mode, to prevent the softswitch at $C011 from being switched. This also makes the CMP work properly. Otherwise the LDA $C000 would get two copies of the same character in the two halves of the A-register.

Lines 3060-3540 are essentially the code from the new //c ROMs. I re-arranged it a little, to make a stand-alone routine within my test-bed, and I changed labels and variable names. Apple uses two sets of tables. One gives quotients and remainders for 0, $100, and $200 (the high byte of the dividend). The other gives quotients and remainders for 0, $08, $10, $20, $40, and $80. A loop runs 5 times to add in the quotients and remainders for bits 3-7 of the dividend, and then fakes one more trip to add in the value of bits 0-2. Not efficient!

Michael Hackney's code is in lines 3560-4080. I'll quote from his letter.

"Apple hi-res graphics characteristically involve various calculations to determine the exact display address from a given X,Y pair. Typically, the vertical position (Y) base address is found by table look-up. The horizontal, or X, position is determined by dividing by 7 (since there are seven pixel bits per byte in the hi-res screen). The integer portion of the division is the byte offset from the base address, and the remainder is the position in the byte. Brute calculation (which is slow for graphics routines) or table lookup (which takes a lot of space) is used to do the division. Table lookup is usually used in good graphics programs. Hi-res graphics require two 280-byte tables, one for quotient and one for remainder. Double hi-res requires tables twice as big. My interest in 65802/816 double-he-res graphics drivers has prompted me to find a serviceable divide-by-seven which is quick and doesn't require more than one page of memory.

"The 65802/816 16-bit operations are ideally suited for this task. Larger numbers can be easily manipulated and table lookup can retrieve 2 bytes of data at once. My routine uses both of these techniques to perform its duty. It divides the original number by eight before doing any table lookup (this keeps the table smaller). The it mulitplies both the quotient and remainder retrieved from the table by 8. The resulting remainder is added to the original lower three bits (the ones shifted out when I divided by 8), and I look into the table again. The first quotient is added to the second quotient, and it is finished. The table only takes 140 bytes, storing quotients and remainders for numbers up to 69. Everything fits in a page with room to spare.

"As an extra bonus, I included a small routine which generates the table in situ. The area occupied by the table generator can be used for data storage once the table is built. It takes longer to load a table from disk than it does to compute one, and the generator dissappears after use, so this is the best way to do it."

In order to get the greatest speed, Michael's table should all reside entirely in the same page of memory. That is why I included line 4100, which justifies the table to the beginning of the next page.

So here you have four great answers to the challenge. Now it's your turn!

=====================================================================
DOCUMENT :AAL-8605:Articles:Front.page.txt
=====================================================================

$1.80

Volume 6 -- Issue 8                        May, 1986

In This Issue...



Enhancing Applesoft with the Toolbox Series

A number of years ago, when Roger Wagner Publishing was still called Southwestern Data Systems, he published Peter Meyer's program "The Routine Machine". The system evolved into four packages: Wizard's Toolbox, Database Toolbox, Video Toolbox, and Chart'n Graph Toolbox. Each "Toolbox" contains a large assortment of assembly language routines which enhance the capabilities of Applesoft. The "Workbench" (included with each Toolbox) allows programmers to add any assortment of these routines to their Applesoft programs at any time. The routines are all called by using the ampersand (&) statement.

Roger will make a special deal for Apple Assembly Line sub- scribers: he'll send a free copy of the "Trial-size Toolbox" (normally $3) to anyone who mentions reading about the package here. The disk includes eight ampersand commands, including a charting command-set with 12 sub-commands, a fixed-length input command, and a print with word-wrap command. All are usable under either DOS 3.3 or ProDOS. Also on the disk is the text of a 50-page manual. The manual includes a tutorial for the toolbox system, a complete explanation of the commands included on the sampler disk, and a comprehensive listing of every command in each of our Toolbox packages. For the free sampler write to Roger Wagner Publishing, Box 582, Santee, CA 92071.

The Toolbox packages are normally $39.95 each. We'll sell them here at S-C for $36 each, or $140 for the complete set.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $45 per year in the US, Canada and Mexico, and $87 to other countries.

=============================================================================
DOCUMENT :AAL-8605:Articles:UniDisk.RWTS.txt
=============================================================================

DOS 3.3 for the UniDisk 3.5 (RWTS 3.5)..............Bill Morgan

We finally got one of Apple's new UniDisk 3.5 drives for the //e, and let me tell you it's very nice. This small but large addition to our favorite computer is about half the volume of a Disk II, but each disk stores almost six times as much infor- mation. It's even a bit faster than the 5.25" drives, about 1.3 times the speed.

Of course there's a catch. In line with Apple's policy of supporting ProDOS only, the new device doesn't use DOS 3.3, at least not as far as Apple is concerned. There are already several different UniDisk versions of DOS, and we're about to build our own right here. It's really quite easy.

There are two parts to the problem: intercepting and handling RWTS calls to the UniDisk slot, and formatting a 3.5" disk with a DOS VTOC and Catalog.

There are a variety of ways to take over a call to RWTS. When we call RWTS at $3D9 it jumps on to $B7B5, where interrupts are disabled before calling the real RWTS entry at $BD00. Some programs take control at $B7B7 and others at $BD00. I looked at the code at $BD00 and saw that it does a little housekeeping and then at $BD10 loads the accumulator with the slot*16 value from the IOB. That looks like the ideal time to check to see if this call is for my slot, so $BD12 is where I patch in the jump to my code. If you are using several nonstandard devices with DOS 3.3 (Sider or other hard disk, RAM disk, other drives) you will need to keep track of who's patching into RWTS where.

Now we come to the question of where to put our version of RWTS. There's certainly no room inside DOS for almost a page of code plus two pages of buffer. I thought I could probably squeeze the code into page three, but that still left that buffer (not to mention the crowd already living at that popular address!) It occurred to me to throw INIT away and put the code inside the existing RWTS at $BEAF, but what about the buffer? I finally decided to use the time-honored technique of moving the DOS buffers and HIMEM down and installing my program and buffer in there. That's also crowded, but where isn't? The first working version of RWTS 3.5 ran at $9900, with the buffer at $9B00-9CFF. The installation routine checked to see if anyone else was using the space and returned an error if so. Applesoft and the S-C Macro Assembler got along with this arrangement just fine, so I spent some time polishing the program and started to write this article.

That's when I was forcibly reminded that the S-C Word Processor sets its own HIMEM and is firmly convinced that $9900-99FF is the buffer for characters deleted off the screen. In other words, the first time I tried to save some text to the UniDisk it blew sky high. I had decided to live without the Word Processor on the UniDisk for the time being when I noticed a couple of interesting things in Beneath Apple DOS. There is a 342-byte buffer inside RWTS at $BB00-BC55, and the code immediately after that buffer is called only by INIT! There really are two full pages of available buffer space inside DOS along with room for the code.

So this edition of RWTS 3.5 runs at $BEAF, with its buffer at $BB00-BCFF. I did hit one more snag when I went to use that buffer area; $BCDF-BCFF is officially unused, which means it's a popular place for other patches. My system has part of our fast LOAD/BLOAD patch (AAL April 83) there, so I had to shave a few more bytes out of my program to make room to move the LOAD patch up to $BF97-BFB7. You may have to make some such adjustment, so be sure to check for some other patch at $BCDF.

The UniDisk 3.5 uses a new software interface, called the Protocol Converter. The PC is a sort of serial bus, which can have several devices daisy-chained to the same controller. We program the PC with a calling structure very similar to the ProDOS MLI calls. Here's an example:

```
CALL  JSR DISPATCH
      .DA #1        read command
      .DA PARMLIST
      BCS ERROR
      ... whatever code

PARMLIST
      .DA #3        3 parameters
      .DA #1        unit number
      .DA BUFFER    buffer address
      .DA <BLOCK    block number (3 bytes)
```

That's all it takes to read a 512-byte block into our buffer. Notice that this standard specifies a 3-byte block number: all current devices use only two bytes of the block number, but they're allowing for expansion beyond 32 megabytes. The unit number isn't the same as a ProDOS unit; this is the position of the device in the PC chain. We need to look up the value of DISPATCH in the card. The byte at $CsFF (s = slot) contains the offset into the ROM of the ProDOS driver entry and the Protocol Converter entry is defined to be 3 bytes after that. For example, in my UniDisk 3.5 controller in slot 5 the byte at $C5FF is $0A. That means that the ProDOS entry to the card is $C50A and the PC entry is $C50D.

There's a quick look at the Protocol Converter. We haven't seen much information published about it yet. The new //c Technical Reference Manual has a good section, including a ROM listing, but the //e UniDisk 3.5 includes no programmer's documentation. Bob is planning a more extensive article on its programming for next month's AAL. Stay tuned...

Apple's new memory expansion card has a PC interface and this RWTS will work with that card as well, but some modification will be needed to use more than one PC at a time. The installation code could scan all slots looking for PCs and build a table of valid slots and entry addresses. Then the initial code at MY.RWTS could search that table and plug the appropriate PC.DISPATCH address into the calls.

The Protocol Converter sees the UniDisk as 1600 blocks of 512 bytes each, for a total of 819,200 (800K) bytes of storage. We have no way to find out about actual tracks and sectors on the disk; this drive seems to use the Macintosh scheme of a variable number of blocks per track. Therefore, we're going to translate DOS's tracks and sectors into some block number and ask the PC for that block, not worrying about where it actually comes from.

The VTOC on a DOS disk has room for 50 tracks of 32 sectors each. That adds up to 400K, or exactly half a UniDisk, so we should be able to set things up with 2 logical drives of 400K each. The number of tracks per disk and the number of sectors per track are both stored as parameters in the VTOC as well, just to make things easier. Two drives per disk means that we can put drive one in the lower 800 blocks and drive two in the upper 800. Figuring that 32 sectors per track means 16 blocks per track and two sectors per block gives us this equation:

BLOCK = (DRIVE-1)*800 + TRACK*16 + SECTOR/2

An even-numbered sector is in the lower half of a block, odd in the upper half.

Since each sector is half of one block on the disk, we can't just write one sector. We have to read a block, copy the new information into half of the buffer, then write that block back out. This takes extra time, but simplifies some of the control logic because every call does a read first.

That first working version of RWTS 3.5 did a new read for every read call, and a new read and write for every write. Well that proved to be much too slow, even slower than the old Disk II. Then I realized that nearly all DOS operations are reading or writing consecutive sectors in a file, so I must be spending a lot of time reading a block that was already in my buffer just to get the sector in the other half of the block. Sure enough, the performance almost doubled when I started keeping track of which block was in the buffer and skipping re-reads of the same block. It does seem to be a good idea to make a special case of the VTOC sector and always re-read that one, just in case we change disks after writing the VTOC as the last operation on the old disk.

Line by Line

In the INSTALL routine we first make sure there is a Protocol Converter in the slot this RWTS expects. If so, we patch in the JMP to our code near the beginning of the normal RWTS and disable INIT by patching an RTS instruction at the beginning of the command handler. MOVE then puts our routine into place at $BEAF and looks up the PC entry point into the ROMS and installs that address into the instructions that call the interface card. NO.PC provide an error message if we can't find a PC. The ID.TABLE has the bytes which mark a PC interface, interspersed with $FFs so we can use the same index for the ROM and the table.

The meat of the program begins at MY.RWTS. We enter here with slot*$10 in the A register so we can check to see if we need to handle this call. If not we execute the instructions we overwrote with the JMP and go back to the normal RWTS. If is is our call, the first thing we do at MINE is check to see if we handled the last RWTS call as well. If so, all is well, but if normal RWTS was used last then it clobbered the buffer at $BB00. We therefore trash LAST.BLOCK so the tests down at CHECK.FOR.RE.READ will be forced to read a new block.

SET.BLOCK tranforms the requested track and sector into a block number, in the process setting carry to indicate whether we want the high or low half of the block. SET.POINTERS then creates two pointers for MY.BUFFER and IOB.BUFFER, using that carry bit along the way. At SET.DRIVE we check which drive is called for and modify BLOCK to read the other half of the diskette if it says drive 2. While we're at it, we plug the drive number into the volume number found, so it will appear as the volume number in a CATALOG. SET.COMMAND gets the command and makes sure it's either READ or WRITE. Anything else becomes a NOP.

At CHECK.FOR.RE.READ we compare the block number requested with the number of the block in the buffer and if they're different we go on to read the new block. If we already have the block we need, CHECK.FOR.VTOC double-checks to see if it's a VTOC we're reading. If so, we need to re-read it anyway, in case it's now a different disk in the drive. Once all that rigamarole is out of the way, the eight bytes at READ are all it takes to actually read the block!

At SKIP.READ we get the command again. (I just noticed that we can move the SET.COMMAND code to this point, since doing an extra READ won't hurt anything, even if the command is bad. That way we can eliminate MY.COMMAND and its STA and LDA instructions. Furthermore, changing the CMP #2 to an LSR and changing the BEQ to a BCC shaves out another byte, for a total of five fewer bytes. There's always more space to be found!) If the command is a READ then READ.MOVE.BUFFER copy MY.BUFFER into the IOB's buffer and we're done. If it's a WRITE, WRITE.MOVE.BUFFER copies the other way, from the IOB buffer into mine, and then calls the ROM to write out the block. Then GOOD.EXIT clears carry and loads a return code of zero before branching to the end. ERROR.EXIT loads up either WRITE PROTECT or DRIVE ERROR and sets carry before returning to the caller.

FORMAT 3.5 ---

Since we threw away INIT to fit all this inside of DOS, and since the standard INIT wouldn't put enough VTOC or CATALOG space on the disk, we're also going to need a special FORMAT program.

There are two stages in the process of formatting a disk: initializing all the tracks with address information; and writing the VTOC, empty catalog track, and boot program. Initializing a Protocol Converter device is easy, just call the PC and let it do all the work. Then we can use our nice new RWTS to write all the rest of the necessary data. Just be sure that RWTS 3.5 is installed before calling FORMAT 3.5.

Since this catalog track is 31 sectors long there is room for 217 files instead of the normal 105. Other than the length, the structure is exactly the same as a normal DOS catalog. The differences in the VTOC are bytes $34-35, the number of tracks per disk and sectors per track, and the bitmap. The bitmap skips tracks $0 and $11, fills all four bytes per track rather than alternate pairs, and extends all the way to the end of the sector.

The boot program here is just a quick message. I hope to have a real boot loader ready for next month's AAL.

=====================================================================
DOCUMENT :AAL-8605:DOS3.3:BETTER.DIV.7.txt
=====================================================================

```
 1000  *SAVE BETTER DIV 7+
 1010  *---------------------------------
 1020  VERSION    .EQ 1
 1030  RBSC65802  .EQ 1
 1040  HACKNEY    .EQ 2
 1050  TWO.C      .EQ 3
 1060  RBSC6502   .EQ 4
 1070  *---------------------------------
 1080  DIVIDEND   .EQ 0,1
 1090  QUO.REM    .EQ 2,3
 1100  T1         .EQ 4,5
 1110  T2         .EQ 6,7
 1120  *---------------------------------
 1130  CROUT  .EQ $FD8E
 1140  PRBYTE .EQ $FDDA
 1150  COUT   .EQ $FDED
 1160  *---------------------------------
 1170         .OP 65802
 1180         .LIST CON
 1190  *---------------------------------
 1200  TEST
 1210         CLC          ENTER NATIVE MODE
 1220         XCE
 1230    .DO VERSION=HACKNEY
 1240         JSR BUILD.HACKNEY.TABLE
 1250    .FIN
 1260         REP #$20     16-BIT A-REGISTER
 1270         LDA ##$3FF   LARGEST VALUE TO TEST
 1280         STA DIVIDEND
 1290  .1     LDA DIVIDEND
 1300    .DO VERSION=RBSC65802
 1310         JSR DIVIDE.BY.SEVEN.65802
 1320         STA QUO.REM  QUO IN 15...8, REM IN 7...0
 1330    .FIN
 1340    .DO VERSION=HACKNEY
 1350         JSR HACKNEY.DIV7
 1360         STA QUO.REM  QUO IN 15...8, REM IN 7...0
 1370    .FIN
 1380    .DO VERSION=RBSC6502
 1390         JSR DIVIDE.BY.SEVEN.6502
 1400    .FIN
 1410    .DO VERSION=TWO.C
 1420         JSR DIV7.TWOC
 1430    .FIN
 1440         JSR CHECK    TEST RESULT BY MULTIPLYING
 1450         BCC .2       ...CORRECT ANSWER
 1460         JSR PRINT    ...INCORRECT DIVISION
 1470  .2     JSR PAUSE    CHECK FOR KEYPRESS
 1480         BEQ .3       <RET>, ABORT
```

```
1490        REP #$20     16-BIT A-REGISTER
1500        DEC DIVIDEND
1510        BPL .1       ...NEXT ONE
1520 .3     SEC          RETURN TO EMULATION MODE
1530        XCE
1540        RTS
1550 *---------------------------------
1560 *   QUO = VAL * .001001001001001
1570 *---------------------------------
1580 DIVIDE.BY.SEVEN.65802
1590        STA T1       SAVE ORIGINAL VALUE
1600        ASL          MULTIPLY BY 64
1610        ASL
1620        ASL
1630        ASL
1640        ASL
1650        ASL
1660        ADC T1       ADD, EQUIV. TO * .01000001
1670        STA T1       SAVE RESULT
1680        LSR          DIVIDE BY 8, WHICH IS
1690        LSR               EQUIV. TO * .00001000001
1700        LSR
1710        ADC T1       EQUIV TO * .01001001001
1720   .DO 0
1730        STA T1       EXTENDED PRECISION METHOD
1740        XBA          GET EQUIV. TO * .00000000000001
1750        AND ##$00FF
1760        LSR
1770        LSR
1780        LSR
1790        LSR
1800        ADC T1       EQUIV. TO * .01001001001001
1810   .ELSE
1820        CMP ##$8800  FUDGE FACTOR METHOD
1830        ADC ##$0008  ADD $0008 TO ALL VALUES,
1840        CMP ##$8800       AND $0002 MORE TO BIG ONES
1850        ADC ##$0000
1860   .FIN
1870        LSR          DIVIDE BY 2, RESULT IS QUOTIENT
1880        SEP #$20          IN HI BYTE, REM IN NEXT 3 BITS
1890        LSR          ISOLATE REMAINDER IN LO BYTE
1900        LSR
1910        LSR
1920        LSR
1930        LSR
1940        REP #$20
1950        RTS
1960 *---------------------------------
1970 DIVIDE.BY.SEVEN.6502
1980        PHP          SAVE M&X BITS
1990        SEC          SWITCH TO EMULATION MODE
2000        XCE
2010        PHP
2020 *---------------------------------
```

```
2030        LDA DIVIDEND
2040        LSR
2050        LSR
2060        LSR
2070        ADC DIVIDEND
2080        ROR
2090        LSR
2100        LSR
2110        ADC DIVIDEND
2120        ROR
2130        AND #$FC
2140        STA T1
2150        LSR
2160        STA T2
2170        LSR
2180        STA QUO.REM+1     QUO = LO-BYTE/7
2190        ADC T1
2200        ADC T2            QUO*7
2210        EOR #$FF          -QUO*7
2220        SEC
2230        ADC DIVIDEND      REM
2240        LDX DIVIDEND+1    0,1, OR 2
2250        ADC RTBL,X
2260        CMP #7
2270        BCC .1
2280        SBC #7
2290 .1     STA QUO.REM       FINAL REMAINDER
2300        LDA QTBL,X
2310        ADC QUO.REM+1
2320        STA QUO.REM+1     FINAL QUOTIENT
2330 *--------------------------------
2340        PLP          SWITCH TO ORIGINAL MODE
2350        XCE
2360        PLP          X&M BITS
2370        RTS
2380 *--------------------------------
2390 QTBL   .DA #0,#36,#73,#109
2400 RTBL   .DA #-1,#3,#0,#4
2410 *--------------------------------
2420 PRINT
2430        PHP          SAVE M&X BITS
2440        SEC          SWITCH TO EMULATION MODE
2450        XCE
2460        PHP          SAVE ORIGINAL MODE (C-BIT)
2470        LDA DIVIDEND+1
2480        ORA #"0"     PRINT DIVIDEND IN HEX
2490        JSR COUT
2500        LDA DIVIDEND
2510        JSR PRBYTE
2520        LDA #" "     PRINT QUOTIENT IN HEX
2530        JSR COUT
2540        LDA QUO.REM+1
2550        JSR PRBYTE
2560        LDA #" "     PRINT REMAINDER IN HEX
```

```
       JSR COUT
       LDA QUO.REM
       JSR PRBYTE
       JSR CROUT    <RETURN>
       PLP          RESTORE NATIVE/EMULATION BIT
       XCE
       PLP          RESTORE M&X BITS
       RTS
*--------------------------------
CHECK
       LDA QUO.REM
       AND ##$FF00  ISOLATE QUOTIENT
       LSR          DIVIDE BY 64 FOR NOW
       LSR
       LSR
       LSR
       LSR
       LSR
       STA T1
       LSR          MULTIPLY BY SEVEN
       STA T2
       LSR
       ADC T1
       ADC T2
       STA T1       QUO * 7
       LDA QUO.REM  CHECK FOR VALID REMAINDER
       AND ##$00FF  0...7
       CMP ##7
       BCS .1       ...INVALID REMAINDER
       ADC T1       ADD QUO*7
       CMP DIVIDEND ...BETTER BE SAME!
       BNE .1       ...NOT, INVALID QUO & REM
       CLC          SIGNAL VALID ANSWERS
       RTS
.1     SEC          SIGNAL INVALID ANSWERS
       RTS
*--------------------------------
PAUSE
       SEP #$20     8-BIT A-REGISTER
       LDA $C000    CHECK KEYBOARD
       BPL .2       NOTHING TYPED
       STA $C010    CLEAR STROBE
       CMP #$8D     <RETURN>?
       BEQ .2       <RET>, SO DON'T PAUSE
.1     LDA $C000    SOME OTHER KEY, SO PAUSE
       BPL .1       ...TILL ANOTHER KEY TYPED
       STA $C010    CLEAR STROBE
.2     CMP #$8D     .EQ. IF <RET>
       RTS          ...ELSE .NE.
*--------------------------------
*   DIVIDE BY 7 FROM NEW //C ROMS (AT $CB4F-CBB0)
*      USED TO GET NUMBER OF 7-BYTES PACKETS
*      IN A BUFFER, FOR THE PROTOCOL CONVERTER
*--------------------------------
```

```
3110  DIV7.TWOC
3120         PHP          SAVE X&M BITS
3130         SEC          ENTER EMULATION MODE
3140         XCE
3150         PHP          SAVE PREVIOUS MODE
3160  *---ALGORITHM FROM //C-----------
3170         LDX DIVIDEND+1    HI BYTE (0, 1, OR 2)
3180         LDA PDIV7TAB,X   0, $100, OR $200 DIVIDED BY 7
3190         STA QUO.REM+1   QUOTIENT SO FAR
3200         LDA PMOD7TAB,X   0, $100, OR $200 MOD 7
3210         STA QUO.REM     REMAINDER SO FAR
3220  *---PROCESS NEXT 5 BITS----------
3230         LDX #5
3240         LDA DIVIDEND      LOW BYTE
3250         STA T1            WORKING COPY
3260         AND #7            LOW 3 BITS
3270         TAY          SAVE FOR LATER USE
3280  .1     ASL T1       GET NEXT BIT FROM DIVIDEND IN CARRY
3290         BCC .4       IF CLEAR, NO EFFECT ON QUO,MOD
3300         LDA MOD7TAB,X     GET MOD7 FOR 2^N
3310  .2     CLC          UPDATE MOD VALUE
3320         ADC QUO.REM
3330         CMP #7       OVERFLOW?
3340         BCC .3       ...NO
3350         SBC #7       ...YES, CORRECT
3360  .3     STA QUO.REM  REMAINDER SO FAR
3370         LDA DIV7TAB,X     GET QUOTIENT FOR 2^N
3380         ADC QUO.REM+1
3390         STA QUO.REM+1     QUOTIENT SO FAR
3400  .4     DEX               ONE LESS BIT TO DEAL WITH
3410         BMI .5            ...FINISHED
3420         BNE .1            ...FIVE TIMES
3430         TYA               GET BACK FIRST 3 BITS
3440         JMP .2            ADD IN REMAINDER
3450  *---RETURN TO CALLER-------------
3460  .5     PLP          ORIGINAL MODE
3470         XCE
3480         PLP          RESTORE X&M BITS
3490         RTS
3500  *--------------------------------
3510  PDIV7TAB .DA #0,#36,#73
3520  PMOD7TAB .DA #0,#4,#1
3530  MOD7TAB .DA #0,#1,#2,#4,#1,#2
3540  DIV7TAB .DA #0,#1,#2,#4,#9,#18
3550  *--------------------------------
3560  HACKNEY.DIV7
3570         STA T1       SAVE VALUE
3580         AND ##$0007  SAVE LOWER 3 BITS (MOD 8)
3590         STA T2
3600         LDA T1       DIVIDE BY 8
3610         LSR
3620         LSR
3630         LSR
3640         ASL          DOUBLE FOR TABLE INDEX
```

```
       TAX          GET QUO & REM FROM TABLE
       LDA TABLE,X
       ASL          MULTIPLY BOTH BY 8
       ASL
       ASL
       ADC T2       ADD LOWER BITS BACK
       TAX          SAVE RESULT
       AND ##$FF00  KEEP QUOTIENT
       STA T1
       TXA          GET REMAINDER
       ASL          DOUBLE FOR INDEX
       TAX
       LDA TABLE,X  GET QUO & REM FROM TABLE
       CLC          ADD PREVIOUS QUOTIENT
       ADC T1
       RTS
*--------------------------------
BUILD.HACKNEY.TABLE
       PHP          SAVE M&X BITS
       REP #$20     LONG A-REG
       LDA ##TABLE
       STA T1
       SEP #$30     ALL REGS SHORT
       LDX #0       X = REMAINDER
       TXY          Y = QUOTIENT
.1     TXA          STORE CURRENT REMAINDER
       STA (T1)
       INC T1
       TYA          STORE CURRENT QUOTIENT
       STA (T1)
       INC T1
       INX          NEXT REMAINDER
       CPX #7
       BCC .1       ...NO CHANGE TO QUOTIENT
       LDX #0       NEXT QUOTIENT
       INY
  .DO 1
       CPY #10      STOP AFTER QUO=9, REM=6
  .ELSE
       CPY #16      STOP AFTER QUO=15, REM=6
  .FIN
       BCC .1       ...NOT YET
       PLP          RESTORE M&X BITS
       RTS
*--------------------------------
       .BS *+255/256*256-*
TABLE  .EQ *
*--------------------------------
```

```
=============================================================================
DOCUMENT :AAL-8605:DOS3.3:FIND.START.txt
=============================================================================

 1000  *SAVE FIND.START
 1010  *--------------------------------
 1020  *   SEARCH FROM HIMEM TO PP FOR LINE "1000"
 1030  *   SET $CA,CB TO BEGINNING OF THAT LINE
 1040  *--------------------------------
 1050  SRCP   .EQ $00,01
 1060  HIMEM  .EQ $4C,4D
 1070  PP     .EQ $CA,CB
 1080  *--------------------------------
 1090         .OR $300
 1100  *--------------------------------
 1110  DO
 1120         LDX PP       IF NOT FIRST TIME,
 1130         LDA PP+1          START WHERE WE LEFT OFF
 1140         BIT DONE.ONCE.FLAG
 1150         BMI .1       ...NOT FIRST TIME
 1160  *---HAS TO BE A FIRST TIME-------
 1170         SEC          SET FLAG
 1180         ROR DONE.ONCE.FLAG
 1190         LDX HIMEM    START AT TOP OF SOURCE AREA
 1200         LDA HIMEM+1
 1210  *---STORE STARTING POINTER-------
 1220  .1     STX SRCP
 1230         STA SRCP+1
 1240         JSR DEC.SRCP
 1250  *---SEARCH FOR "1000"------------
 1260  .2     JSR DEC.SRCP
 1270         LDA SRCP+1
 1280         CMP /$0800   DON'T SEARCH BEYOND $800
 1290         BCC .3       ...END OF SEARCH
 1300         LDY #0
 1310         LDA (SRCP),Y
 1320         CMP #1000    COMPARE LO-BYTE
 1330         BNE .2       ...NO, KEEP SCANNING
 1340         INY          ...MATCH, CHECK HI-BYTE
 1350         LDA (SRCP),Y
 1360         CMP /1000
 1370         BNE .2       ...NO, KEEP SCANNING
 1380  *---FOUND IT, POINT PP TO IT-----
 1390         JSR DEC.SRCP BACK UP OVER BYTE COUNT
 1400         LDA SRCP
 1410         STA PP
 1420         LDA SRCP+1
 1430         STA PP+1
 1440  .3     RTS
 1450  *--------------------------------
 1460  DEC.SRCP
 1470         LDA SRCP
 1480         BNE .1
```

```
1490        DEC SRCP+1
1500 .1     DEC SRCP
1510        RTS
1520 *---------------------------------
1530 DONE.ONCE.FLAG .HS 00
1540 *---------------------------------
```

```
==============================================================================
DOCUMENT :AAL-8605:DOS3.3:RWTS.3.5.txt
==============================================================================
```

( DTC removed -- lots of garbage characters )

==============================================================================
DOCUMENT :AAL-8605:DOS3.3:S.Format.UDsk.txt
==============================================================================

```
 1000  *SAVE S.FORMAT.UNIDISK
 1010  *--------------------------------
 1020  UNIDISK.SLOT .EQ 5
 1030
 1040  RWTS         .EQ $3D9
 1050
 1060  PC.DISPATCH  .EQ UNIDISK.SLOT*$100+$C000
 1070
 1080  HOME         .EQ $FC58
 1090  COUT         .EQ $FDED
 1100  *--------------------------------
 1110         .OR $803
 1120  *      .TF FORMAT.UNIDISK
 1130
 1140  FORMAT CLC
 1150         LDA UNIDISK.SLOT*$100+$C0FF
 1160         ADC #3
 1170         STA PC.CALL
 1180         JSR PC.DISPATCH   format the disk
 1190  PC.CALL .EQ *-2
 1200         .DA #3
 1210         .DA PC.PARMS
 1220         BCS ERROR
 1230         LDA #2
 1240         STA DRIVE         do drive 2 first
 1250
 1260  DO.CATALOG
 1270         JSR CLEAR.BUFFER
 1280         LDA #$11
 1290         STA TRACK
 1300         STA MY.BUFFER+1   link pointer
 1310         LDY #$1F
 1320  .1     STY SECTOR
 1330         DEY
 1340         BNE .2
 1350         STY MY.BUFFER+1   mark end of catalog
 1360  .2     STY MY.BUFFER+2   link pointer
 1370         JSR CALL.RWTS
 1380         LDY SECTOR
 1390         DEY
 1400         BNE .1            and go back for more
 1410         STY SECTOR
 1420
 1430  DO.VTOC
 1440         JSR CLEAR.BUFFER
 1450         LDX #0
 1460  .1     LDY VTOC.INDEXES,X
 1470         LDA VTOC.VALUES,X
 1480         STA MY.BUFFER,Y   set VTOC header info
```

```
1490        INX
1500        CPX #ENTRY.COUNT
1510        BCC .1
1520        LDA DRIVE          use drive # for volume
1530        STA MY.BUFFER+6
1540        LDA #$FF
1550        INY
1560 .2     INY                skip a track in bitmap
1570        INY
1580        INY
1590        INY
1600 .3     STA MY.BUFFER,Y    mark free
1610        INY
1620        BEQ .4             leave if done
1630        CPY #$7C           track $11?
1640        BEQ .2             yes, skip it
1650        BNE .3             no, go on
1660 .4     JSR CALL.RWTS
1670        DEC DRIVE          now go back and
1680        BNE DO.CATALOG     do drive one
1690
1700 DO.BOOT.SECTOR
1710        INC DRIVE          that was drive one,
1720        JSR CLEAR.BUFFER   so write a boot sector
1730        STA TRACK          A = 0
1740        STA SECTOR
1750        LDY #BOOT.SIZE
1760 .1     LDA BOOT.IMAGE,Y   install the image
1770        STA MY.BUFFER,Y
1780        DEY
1790        BPL .1             fall into CALL.RWTS
1800 *--------------------------------
1810 CALL.RWTS
1820        LDA /IOB
1830        LDY #IOB
1840        JSR RWTS
1850        BCS ERROR
1860        RTS
1870 ERROR  BRK
1880 *--------------------------------
1890 CLEAR.BUFFER
1900        LDY #0
1910        TYA
1920 .1     STA MY.BUFFER,Y
1930        INY
1940        BNE .1
1950        RTS
1960 *--------------------------------
1970 PC.PARMS .DA #1     one parm
1980          .DA #1     unit one
1990 *--------------------------------
2000 IOB    .DA #1
2010 SLOT   .DA #UNIDISK.SLOT*$10
2020 DRIVE  .BS 1
```

```
2030  VOL    .DA #0
2040  TRACK  .BS 1
2050  SECTOR .BS 1
2060  DCT    .DA $B7FB
2070  BUFFER .DA MY.BUFFER
2080         .BS 1
2090         .DA #0
2100  COMAND .DA #2       write
2110  RETURN .BS 1
2120  P.VOL  .BS 1
2130  P.SLOT .BS 1
2140  P.DRIV .BS 1
2150  *--------------------------------
2160  VTOC.INDEXES .HS 00.01.02.03.27.30.31.34.35.36.37
2170  ENTRY.COUNT .EQ *-VTOC.INDEXES
2180  VTOC.VALUES  .HS 04.11.1F.03.7A.11.01.32.20.00.01
2190  *--------------------------------
2200  BOOT.IMAGE
2210         .PH $800
2220  BOOT   .HS 01
2230         JSR HOME
2240         LDY #0
2250  .1     LDA MESSAGE,Y
2260         BEQ .2
2270         JSR COUT          print message
2280         INY
2290         BNE .1
2300  .2     BEQ .2            and hang...
2310
2320  MESSAGE
2330         .HS 8D8D8D
2340         .AS -/Sorry, can't boot DOS here yet./
2350         .HS 8D8700
2360         .EP
2370  BOOT.SIZE .EQ *-BOOT.IMAGE
2380  *--------------------------------
2390  MY.BUFFER
2400         .LIF
```

```
==============================================================================
DOCUMENT :AAL-8605:DOS3.3:S.UNIDISK.RWTS.txt
==============================================================================

 1000  *SAVE S.UNIDISK RWTS
 1010  *--------------------------------
 1020  UNIDISK.SLOT        .EQ 5
 1030
 1040  MY.COMMAND          .EQ $26
 1050  MY.BUFFER.POINTER   .EQ $3C
 1060  IOB.BUFFER.POINTER  .EQ $3E
 1070  IOB.PTR             .EQ $48
 1080
 1090  MY.BUFFER           .EQ $BB00
 1100
 1110  PATCH.POINT         .EQ $BD12
 1120  PATCH.RETURN        .EQ $BD15
 1130
 1140  PC.DISPATCH         .EQ UNIDISK.SLOT*$100+$C000
 1150
 1160  PRBYTE              .EQ $FDDA
 1170  COUT                .EQ $FDED
 1180  *--------------------------------
 1190         .OR $803
 1200         .TF RWTS 3.5
 1210
 1220  INSTALL
 1230         LDX #6            make sure we have a
 1240  .1     LDA ID.TABLE,X    protocol converter
 1250         CMP UNIDISK.SLOT*$100+$C001,X
 1260         BNE NO.PC
 1270         DEX
 1280         DEX
 1290         BPL .1
 1300
 1310         LDA #$4C          patch in the JMP
 1320         STA PATCH.POINT   to our code
 1330         LDA #MY.RWTS
 1340         STA PATCH.POINT+1
 1350         LDA /MY.RWTS
 1360         STA PATCH.POINT+2
 1370         LDA #$60
 1380         STA $A54F         disable INIT
 1390
 1400  MOVE   LDY #IMAGE.SIZE+1 install our code
 1410  .1     LDA IMAGE-1,Y
 1420         STA MY.RWTS-1,Y
 1430         DEY
 1440         BNE .1
 1450
 1460         CLC
 1470         LDA UNIDISK.SLOT*$100+$C0FF
 1480         ADC #3            find protocol
```

```
1490        STA READ.CALL     converter entry
1500        STA WRITE.CALL
1510        BNE DONE          ...always
1520
1530 NO.PC  LDX #0
1540 .1     LDA MESSAGES,X    print an error message
1550        BEQ DONE
1560        JSR COUT
1570        INX
1580        BNE .1
1590 DONE   JMP $3D0
1600 *--------------------------------
1610 MESSAGES
1620        .HS 8D
1630        .AS -/No PC in slot /
1640        .DA #$B0+UNIDISK.SLOT
1650        .HS 878D00
1660 *--------------------------------
1670 ID.TABLE .HS 20.FF.00.FF.03.FF.00
1680 *            ^     ^     ^     ^
1690 *        Protocol Converter ID Bytes
1700 *--------------------------------
1710 IMAGE  .EQ *
1720        .PH $BEAF
1730 MY.RWTS
1740        CMP #UNIDISK.SLOT*$10
1750        BEQ MINE          my call!
1760        TAX               not mine, so do
1770        LDY #$F           patched-over code
1780        JMP PATCH.RETURN  and go back
1790 *--------------------------------
1800 MINE
1810        LDY #$F
1820        CMP (IOB.PTR),Y   check previous slot
1830        BEQ SET.BLOCK     same, so go on
1840        STA (IOB.PTR),Y   set previous slot
1850        LDA #$FF
1860        STA LAST.BLOCK    trash LAST.BLOCK
1870
1880 SET.BLOCK
1890        LDA #0
1900        STA BLOCK+1
1910        LDY #4
1920        LDA (IOB.PTR),Y   get track
1930 .1     ASL
1940        ROL BLOCK+1       *16
1950        DEY
1960        BNE .1
1970        STA BLOCK
1980        LDY #5
1990        LDA (IOB.PTR),Y   get sector
2000        LSR               /2, odd/even into carry
2010        ORA BLOCK
2020        STA BLOCK
```

```
2030
2040  SET.POINTERS
2050         LDA #MY.BUFFER
2060         STA MY.BUFFER.POINTER
2070         LDA /MY.BUFFER
2080         ADC #0       carry sets hi/lo half of buffer
2090         STA MY.BUFFER.POINTER+1
2100         LDY #8
2110         LDA (IOB.PTR),Y   get IOB buffer
2120         STA IOB.BUFFER.POINTER
2130         INY
2140         LDA (IOB.PTR),Y
2150         STA IOB.BUFFER.POINTER+1
2160
2170  SET.DRIVE
2180         LDY #2
2190         LDA (IOB.PTR),Y   get drive
2200         LDY #$10
2210         STA (IOB.PTR),Y   set previous drive
2220         DEY
2230         DEY
2240         STA (IOB.PTR),Y   set previous volume
2250         LSR
2260         BCS SET.COMMAND   .CS. if D1
2270         LDA BLOCK         add 800 to BLOCK if D2
2280         ADC #800
2290         STA BLOCK
2300         LDA BLOCK+1
2310         ADC /800
2320         STA BLOCK+1
2330
2340  SET.COMMAND
2350         LDY #$C
2360         LDA (IOB.PTR),Y   get command
2370         BEQ GOOD.EXIT
2380         CMP #3            exit if not READ or WRITE
2390         BCS GOOD.EXIT
2400         STA MY.COMMAND    save command
2410
2420  CHECK.FOR.RE.READ
2430         LDX #0            zero the flag
2440         LDY #1            check two bytes
2450  .1     LDA BLOCK,Y
2460         CMP LAST.BLOCK,Y  compare
2470         BEQ .2            same, so go on
2480         INX               different, so flag it
2490         STA LAST.BLOCK,Y  and store new value
2500  .2     DEY
2510         BPL .1            now do low bytes
2520         TXA               check the flag
2530         BNE READ          if different, go read
2540
2550  CHECK.FOR.VTOC
2560         LDY #5
```

```
2570        LDA (IOB.PTR),Y   get sector
2580        BNE SKIP.READ     non-zero isn't VTOC
2590        DEY
2600        LDA (IOB.PTR),Y   get track
2610        CMP #$11
2620        BNE SKIP.READ     not $11 isn't VTOC
2630
2640 READ   JSR PC.DISPATCH
2650 READ.CALL .EQ *-2
2660        .DA #1            READ
2670        .DA PARMLIST
2680        BCS ERROR.EXIT
2690
2700 SKIP.READ
2710        LDA MY.COMMAND    check command
2720        CMP #2
2730        BEQ WRITE.MOVE.BUFFER
2740
2750 READ.MOVE.BUFFER
2760        LDY #0
2770 .1     LDA (MY.BUFFER.POINTER),Y
2780        STA (IOB.BUFFER.POINTER),Y
2790        INY
2800        BNE .1
2810        BEQ GOOD.EXIT     ...always
2820
2830 WRITE.MOVE.BUFFER
2840        LDY #0
2850 .1     LDA (IOB.BUFFER.POINTER),Y
2860        STA (MY.BUFFER.POINTER),Y
2870        INY
2880        BNE .1
2890
2900 WRITE  JSR PC.DISPATCH
2910 WRITE.CALL .EQ *-2
2920        .DA #2            WRITE
2930        .DA PARMLIST
2940        BCS ERROR.EXIT
2950
2960 GOOD.EXIT
2970        CLC
2980        LDA #0
2990        BEQ EXIT          ...always
3000
3010 ERROR.EXIT
3020        CMP #$2B     write protect?
3030        BEQ .1
3040        LDA #$40     make everything else DRIVE ERROR
3050        .HS 2C
3060 .1     LDA #$10
3070        SEC
3080
3090 EXIT   LDY #$D
3100        STA (IOB.PTR),Y   save return code
```

```
3110        RTS
3120 *--------------------------------
3130 PARMLIST
3140        .DA #3        3 parameters
3150        .DA #1        unit number
3160        .DA MY.BUFFER buffer address
3170 BLOCK  .BS 3         block number
3180
3190 LAST.BLOCK .HS FFFF
3200 *--------------------------------
3210        .BS $BF97-*
3220        .EP
3230 IMAGE.END .EQ *-1
3240 IMAGE.SIZE .EQ IMAGE.END-IMAGE
3250        .LIF
```

```
==============================================================
DOCUMENT :AAL-8605:ProDOS:BETTER.DIV.7.txt
==============================================================

 1000  *SAVE BETTER.DIV.7
 1010  *--------------------------------
 1020  VERSION    .EQ 1
 1030  RBSC65802  .EQ 1
 1040  HACKNEY    .EQ 2
 1050  TWO.C      .EQ 3
 1060  RBSC6502   .EQ 4
 1070  *--------------------------------
 1080  DIVIDEND   .EQ 0,1
 1090  QUO.REM    .EQ 2,3
 1100  T1         .EQ 4,5
 1110  T2         .EQ 6,7
 1120  *--------------------------------
 1130  CROUT  .EQ $FD8E
 1140  PRBYTE .EQ $FDDA
 1150  COUT   .EQ $FDED
 1160  *--------------------------------
 1170         .OP 65802
 1180         .LIST CON
 1190  *--------------------------------
 1200  TEST
 1210         CLC          ENTER NATIVE MODE
 1220         XCE
 1230    .DO VERSION=HACKNEY
 1240         JSR BUILD.HACKNEY.TABLE
 1250    .FIN
 1260         REP #$20     16-BIT A-REGISTER
 1270         LDA ##$3FF   LARGEST VALUE TO TEST
 1280         STA DIVIDEND
 1290  .1     LDA DIVIDEND
 1300    .DO VERSION=RBSC65802
 1310         JSR DIVIDE.BY.SEVEN.65802
 1320         STA QUO.REM  QUO IN 15...8, REM IN 7...0
 1330    .FIN
 1340    .DO VERSION=HACKNEY
 1350         JSR HACKNEY.DIV7
 1360         STA QUO.REM  QUO IN 15...8, REM IN 7...0
 1370    .FIN
 1380    .DO VERSION=RBSC6502
 1390         JSR DIVIDE.BY.SEVEN.6502
 1400    .FIN
 1410    .DO VERSION=TWO.C
 1420         JSR DIV7.TWOC
 1430    .FIN
 1440         JSR CHECK    TEST RESULT BY MULTIPLYING
 1450         BCC .2       ...CORRECT ANSWER
 1460         JSR PRINT    ...INCORRECT DIVISION
 1470  .2     JSR PAUSE    CHECK FOR KEYPRESS
 1480         BEQ .3       <RET>, ABORT
```

```
1490         REP #$20     16-BIT A-REGISTER
1500         DEC DIVIDEND
1510         BPL .1       ...NEXT ONE
1520  .3     SEC          RETURN TO EMULATION MODE
1530         XCE
1540         RTS
1550  *--------------------------------
1560  *   QUO = VAL * .001001001001001
1570  *--------------------------------
1580  DIVIDE.BY.SEVEN.65802
1590         STA T1       SAVE ORIGINAL VALUE
1600         ASL          MULTIPLY BY 64
1610         ASL
1620         ASL
1630         ASL
1640         ASL
1650         ASL
1660         ADC T1       ADD, EQUIV. TO * .01000001
1670         STA T1       SAVE RESULT
1680         LSR          DIVIDE BY 8, WHICH IS
1690         LSR               EQUIV. TO * .00001000001
1700         LSR
1710         ADC T1       EQUIV TO * .01001001001
1720    .DO 0
1730         STA T1       EXTENDED PRECISION METHOD
1740         XBA          GET EQUIV. TO * .00000000000001
1750         AND ##$00FF
1760         LSR
1770         LSR
1780         LSR
1790         LSR
1800         ADC T1       EQUIV. TO * .01001001001001
1810    .ELSE
1820         CMP ##$8800  FUDGE FACTOR METHOD
1830         ADC ##$0008  ADD $0008 TO ALL VALUES,
1840         CMP ##$8800       AND $0002 MORE TO BIG ONES
1850         ADC ##$0000
1860    .FIN
1870         LSR          DIVIDE BY 2, RESULT IS QUOTIENT
1880         SEP #$20          IN HI BYTE, REM IN NEXT 3 BITS
1890         LSR          ISOLATE REMAINDER IN LO BYTE
1900         LSR
1910         LSR
1920         LSR
1930         LSR
1940         REP #$20
1950         RTS
1960  *--------------------------------
1970  DIVIDE.BY.SEVEN.6502
1980         PHP          SAVE M&X BITS
1990         SEC          SWITCH TO EMULATION MODE
2000         XCE
2010         PHP
2020  *--------------------------------
```

```
2030        LDA DIVIDEND
2040        LSR
2050        LSR
2060        LSR
2070        ADC DIVIDEND
2080        ROR
2090        LSR
2100        LSR
2110        ADC DIVIDEND
2120        ROR
2130        AND #$FC
2140        STA T1
2150        LSR
2160        STA T2
2170        LSR
2180        STA QUO.REM+1     QUO = LO-BYTE/7
2190        ADC T1
2200        ADC T2            QUO*7
2210        EOR #$FF          -QUO*7
2220        SEC
2230        ADC DIVIDEND      REM
2240        LDX DIVIDEND+1    0,1, OR 2
2250        ADC RTBL,X
2260        CMP #7
2270        BCC .1
2280        SBC #7
2290 .1     STA QUO.REM       FINAL REMAINDER
2300        LDA QTBL,X
2310        ADC QUO.REM+1
2320        STA QUO.REM+1     FINAL QUOTIENT
2330 *--------------------------------
2340        PLP          SWITCH TO ORIGINAL MODE
2350        XCE
2360        PLP          X&M BITS
2370        RTS
2380 *--------------------------------
2390 QTBL   .DA #0,#36,#73,#109
2400 RTBL   .DA #-1,#3,#0,#4
2410 *--------------------------------
2420 PRINT
2430        PHP          SAVE M&X BITS
2440        SEC          SWITCH TO EMULATION MODE
2450        XCE
2460        PHP          SAVE ORIGINAL MODE (C-BIT)
2470        LDA DIVIDEND+1
2480        ORA #"0"     PRINT DIVIDEND IN HEX
2490        JSR COUT
2500        LDA DIVIDEND
2510        JSR PRBYTE
2520        LDA #" "     PRINT QUOTIENT IN HEX
2530        JSR COUT
2540        LDA QUO.REM+1
2550        JSR PRBYTE
2560        LDA #" "     PRINT REMAINDER IN HEX
```

```
2570        JSR COUT
2580        LDA QUO.REM
2590        JSR PRBYTE
2600        JSR CROUT     <RETURN>
2610        PLP           RESTORE NATIVE/EMULATION BIT
2620        XCE
2630        PLP           RESTORE M&X BITS
2640        RTS
2650 *--------------------------------
2660 CHECK
2670        LDA QUO.REM
2680        AND ##$FF00   ISOLATE QUOTIENT
2690        LSR           DIVIDE BY 64 FOR NOW
2700        LSR
2710        LSR
2720        LSR
2730        LSR
2740        LSR
2750        STA T1
2760        LSR           MULTIPLY BY SEVEN
2770        STA T2
2780        LSR
2790        ADC T1
2800        ADC T2
2810        STA T1        QUO * 7
2820        LDA QUO.REM   CHECK FOR VALID REMAINDER
2830        AND ##$00FF   0...7
2840        CMP ##7
2850        BCS .1        ...INVALID REMAINDER
2860        ADC T1        ADD QUO*7
2870        CMP DIVIDEND ...BETTER BE SAME!
2880        BNE .1        ...NOT, INVALID QUO & REM
2890        CLC           SIGNAL VALID ANSWERS
2900        RTS
2910 .1     SEC           SIGNAL INVALID ANSWERS
2920        RTS
2930 *--------------------------------
2940 PAUSE
2950        SEP #$20      8-BIT A-REGISTER
2960        LDA $C000     CHECK KEYBOARD
2970        BPL .2        NOTHING TYPED
2980        STA $C010     CLEAR STROBE
2990        CMP #$8D      <RETURN>?
3000        BEQ .2        <RET>, SO DON'T PAUSE
3010 .1     LDA $C000     SOME OTHER KEY, SO PAUSE
3020        BPL .1        ...TILL ANOTHER KEY TYPED
3030        STA $C010     CLEAR STROBE
3040 .2     CMP #$8D      .EQ. IF <RET>
3050        RTS           ...ELSE .NE.
3060 *--------------------------------
3070 *   DIVIDE BY 7 FROM NEW //C ROMS (AT $CB4F-CBB0)
3080 *      USED TO GET NUMBER OF 7-BYTES PACKETS
3090 *      IN A BUFFER, FOR THE PROTOCOL CONVERTER
3100 *--------------------------------
```

```
3110  DIV7.TWOC
3120         PHP          SAVE X&M BITS
3130         SEC          ENTER EMULATION MODE
3140         XCE
3150         PHP          SAVE PREVIOUS MODE
3160  *---ALGORITHM FROM //C-----------
3170         LDX DIVIDEND+1    HI BYTE (0, 1, OR 2)
3180         LDA PDIV7TAB,X   0, $100, OR $200 DIVIDED BY 7
3190         STA QUO.REM+1   QUOTIENT SO FAR
3200         LDA PMOD7TAB,X   0, $100, OR $200 MOD 7
3210         STA QUO.REM     REMAINDER SO FAR
3220  *---PROCESS NEXT 5 BITS----------
3230         LDX #5
3240         LDA DIVIDEND      LOW BYTE
3250         STA T1            WORKING COPY
3260         AND #7            LOW 3 BITS
3270         TAY          SAVE FOR LATER USE
3280  .1     ASL T1       GET NEXT BIT FROM DIVIDEND IN CARRY
3290         BCC .4       IF CLEAR, NO EFFECT ON QUO,MOD
3300         LDA MOD7TAB,X     GET MOD7 FOR 2^N
3310  .2     CLC          UPDATE MOD VALUE
3320         ADC QUO.REM
3330         CMP #7       OVERFLOW?
3340         BCC .3       ...NO
3350         SBC #7       ...YES, CORRECT
3360  .3     STA QUO.REM  REMAINDER SO FAR
3370         LDA DIV7TAB,X     GET QUOTIENT FOR 2^N
3380         ADC QUO.REM+1
3390         STA QUO.REM+1     QUOTIENT SO FAR
3400  .4     DEX               ONE LESS BIT TO DEAL WITH
3410         BMI .5            ...FINISHED
3420         BNE .1            ...FIVE TIMES
3430         TYA               GET BACK FIRST 3 BITS
3440         JMP .2            ADD IN REMAINDER
3450  *---RETURN TO CALLER-------------
3460  .5     PLP          ORIGINAL MODE
3470         XCE
3480         PLP          RESTORE X&M BITS
3490         RTS
3500  *--------------------------------
3510  PDIV7TAB .DA #0,#36,#73
3520  PMOD7TAB .DA #0,#4,#1
3530  MOD7TAB .DA #0,#1,#2,#4,#1,#2
3540  DIV7TAB .DA #0,#1,#2,#4,#9,#18
3550  *--------------------------------
3560  HACKNEY.DIV7
3570         STA T1       SAVE VALUE
3580         AND ##$0007  SAVE LOWER 3 BITS (MOD 8)
3590         STA T2
3600         LDA T1       DIVIDE BY 8
3610         LSR
3620         LSR
3630         LSR
3640         ASL          DOUBLE FOR TABLE INDEX
```

```
        TAX           GET QUO & REM FROM TABLE
        LDA TABLE,X
        ASL           MULTIPLY BOTH BY 8
        ASL
        ASL
        ADC T2        ADD LOWER BITS BACK
        TAX           SAVE RESULT
        AND ##$FF00   KEEP QUOTIENT
        STA T1
        TXA           GET REMAINDER
        ASL           DOUBLE FOR INDEX
        TAX
        LDA TABLE,X   GET QUO & REM FROM TABLE
        CLC           ADD PREVIOUS QUOTIENT
        ADC T1
        RTS
*--------------------------------
BUILD.HACKNEY.TABLE
        PHP           SAVE M&X BITS
        REP #$20      LONG A-REG
        LDA ##TABLE
        STA T1
        SEP #$30      ALL REGS SHORT
        LDX #0        X = REMAINDER
        TXY           Y = QUOTIENT
.1      TXA           STORE CURRENT REMAINDER
        STA (T1)
        INC T1
        TYA           STORE CURRENT QUOTIENT
        STA (T1)
        INC T1
        INX           NEXT REMAINDER
        CPX #7
        BCC .1        ...NO CHANGE TO QUOTIENT
        LDX #0        NEXT QUOTIENT
        INY
  .DO 1
        CPY #10       STOP AFTER QUO=9, REM=6
  .ELSE
        CPY #16       STOP AFTER QUO=15, REM=6
  .FIN
        BCC .1        ...NOT YET
        PLP           RESTORE M&X BITS
        RTS
*--------------------------------
        .BS *+255/256*256-*
TABLE   .EQ *
*--------------------------------
```

```
=========================================================================
DOCUMENT :AAL-8606:Articles:Butterill.Ops.txt
=========================================================================
```

Fast 16x16 Multiply & Divide in 65802............John Butterill
Ottowa, Ontario

Recently I needed a 16-bit multiplication subroutine in my 65802-enhanced Apple II. Naturally, I needed one that was both fast and short. I referred back to the Jan 86 AAL, which contained several examples for the 65802. The one named FASTER caught my fancy because it seemed a good compromise between size and speed. Then I made some changes which I think significantly improve it.

I noted that when you ROR the low half of the product into the multiplier, you get a bit out. This bit remains in the carry. If the low-product and the multiplier share the same location, then you can ROL in the low-product bit and ROL out the multi- plier bit at the same time, instead of loading and LSR-ing the multiplier. By not having to load the multiplier, the Accumu- lator is free to contain the high half of the product without saving and loading it each time around. The result is rather more compact, fitting into 35 bytes (FASTER took 42 bytes).

It is also faster. By my calculations, the best and worst cases take 335 and 383 cycles, respectively. This includes the JSR to call the subroutine and the RTS to get back.

At the expense of two more bytes, I can save nine more cycles: delete line 1240 and add the following:

```
       1304    ROR
       1305    ROR A
```

This avoids the 17th trip through the loop, whose only purpose was to roll-in the final bit of the product.

By the way, some assemblers use the syntax "ROR A" to rotate the contents of the A-register. The S-C Macro Assembler and some others use the syntax "ROR" with a blank operand field for that mode. Then "ROR A" means to rotate the contents of the variable named "A", as in my program. To avoid confusion, you might want to change the variable names, avoiding the name "A".

<<<<listing of multiply subroutine>>>>

A 16-bit by 16-bit division seems inherently messier. First, the divisor must be shifted left until it is at least greater than half the dividend. One can do a fast cycle which shifts the divisor all the way to the left, but for every shift left in this loop, the divisor must be shifted right again in the second (subtracting) loop.

In practice, I feel that the values would not be randomly distributed, but would be biased toward smaller values. I'm more likely to divide by 7 than by 32973, for example. Therefore it is worthwhile putting in the extra code to shift left only as far as is necessary. The scaling portion in my subroutine, lines 1240-1300, shift the divisor until either bit 15 = 1 or the divisor equals/exceeds the dividend.

In the second loop, lines 1310-1400, the shifted divisor is repeatedly compared to the dividend. If it is smaller, it is subtracted and a 1-bit goes into the quotient; otherwise a 0-bit goes in. The loop stops after it has operated with the divisor shifted back to its original position. This is ordinary long division, in binary. The comparison-subtraction is performed from one to 16 times, depending on the values.

As I calculate it, the best case (dividend=divisor) takes 82 cycles. The worst case, which I think would be $FFFF/1, takes 676 cycles. The time is a function of the number of significant bits in the answer.

<<<division subroutine>>>

[ John also wrote a nice demonstration driver for his subroutines, allowing you to enter two hexadecimal values and see the result in hexadecimal. The source code for the demo is included on the monthly/quarterly disk. ]

```
==============================================================================
DOCUMENT :AAL-8606:Articles:Call.Sequences.txt
==============================================================================
```

Using the 65816 Stack Relative Mode........Bob Sander-Cederlof

The 65802 and 65816 have two new address modes that allow you to reach into the stack. The "offset,S" mode lets you access position relative to the stack pointer, and the "(offset,S),Y" mode lets you access data indirectly through an address that is on the stack. The new address modes are available even when the 65802/16 is in the "emulation" mode.

The hardware adds the value of the offset to the current stack pointer to form an effective address. The stack pointer is always pointing at one address below the end of the stack. Thus, an address of "1,S" points to the first item on the stack.

Having these new modes leads to interesting programming possibilities. When you design a subroutine, you have to decide how you are going to pass parameters into and out of the subroutine. Usually we try to use the A, X, and Y registers first. Another method puts the data of the address of the data after the JSR that calls the subroutine. ProDOS MLI calls use this method:

```
       JSR $BF00
       .DA #$C1,PARMS
```

In another method you push data or data addresses on the stack, and then call the subroutine. This is the preferred method in some computers, but not the 6502. The new modes make this mode work nicely in the 65802/16, though.

I coded up two examples to show you might use the new modes. They both are message printing subroutines. The calling method requires telling the subroutine where to find a variable length message. In the first example (lines 1070-1330), I chose to push the address of the message text on the stack before calling the message printing subroutine. In the second example (lines 1340-1640), I used the method of storing the message text immediately after the JSR instruction.

Lines 1070-1110 print out two messages, using the first technique. I use the PEA (Push Effective Address) instruction to put the address of the first byte of the message text on the stack. This instruction pushes first the high byte, then the low byte, of the value of the operand. (I think I would prefer to have called it "PSH #value", because that is the effect. Then the PEI opcode, which pushes two bytes from the direct page, could be "PSH zp". But, nobody asked me.)

Anyway, let's look at the PRINT.IT subroutine. When the subroutine starts looking at the stack, it looks like this:

```
       |  msg addr lo  | 4,S
```

```
        | ------------- |
        |  msg addr hi  | 3,S
        | ------------- |
        |  ret addr lo  | 2,S
        | ------------- |
        |  ret addr hi  | 1,S
        | ------------- |
        |               |<---Stack Pointer
```

The LDA (3,S),Y instruction at line 1240 takes the address at 3,S and 4,S (which is the address of the first byte of the message) and adds the Y-register to it; then the LDA opcode picks up the message byte. After printing all the message and finding the terminating 00 byte, lines 1290-1320 move the return address up two slots higher in the stack (over the top of the message address). At the same time, the original copy of the return address is removed from the stack. Then a simple RTS takes us back to the caller, with a clean stack.

The second example uses the "message buried in the code" method. when PRINT.MSG looks at the stack, only the return address is there. The return address points to the third byte of the JSR instruction, one byte before the message text. Therefore the message printing loop in lines 1500-1550 starts with Y=1. Lines 1560-1620 add the message length to the return address, so that an RTS opcode will return to the caller just past the message.

<<<code here>>>

It might be instructive to look at how these two examples could be code in a plain 6502 environment. First, we must replace the PEA opcodes in lines 1070 and 1090 with the following:

```
       LDA #MESSAGE
       PHA
       LDA /MESSAGE
       PHA
```

Then PRINT.IT would require using temporary memory somewhere or writing self-modifying code. With a pointer in page zero, it could work like this:

<<<6502 version of print.it>>>

PRINT.MSG also can be written in pure 6502 code with either self-modifying code or a pointer in page zero. Here is the self-modifying version:

<<<6502 version of print.msg>>>

==============================================================================
DOCUMENT :AAL-8606:Articles:CorrexAbtBruns.txt
==============================================================================

The Real Story about DOS and BRUN.........Bob Sander-Cederlof

I was wrong. Some of you were kind enough to point it out. John Butterill sent a letter, and others called (sorry, names forgotten). I said, in the January 1986 AAL, that the reason BRUNning programs from inside Applesoft programs often did not work was the fact that DOS used a JMP rather than a JSR to call your program.

The truth is that DOS does call your program with a JMP, but there is still a return address on the stack. The BRUN command processor itself was called with a JSR, in a way. At $A17A there is a JSR $A180. The routine at $A180 jumps to the BRUN processor. So when your program finishes it will return to $A17D, right after the JSR $A180. From there it goes to $9F83.

At $9F83, DOS will finally exit from doing the BRUN command. If MON C is on, the carriage return from the end of the BRUN command will be echoed at this time. This can put you into a loop, however, because the BRUN command re-installed the DOS hooks in the input and output vectors. When the DOS hooks are installed, any character input or output will enter DOS first. Since we are still, in effect, inside DOS, because of the BRUN, we get into a loop. DOS is not re-entrant, as John Butterill put it. The BRUN command processor does a JSR $A851, which re-installs the DOS hooks. If your program tries to do any character I/O through calls to $FDED (COUT) or $FD0C (RDKEY), and you start up your program by BRUNning it from inside an Applesoft program, you will get DOS into a loop. Or, even if your program does not do any I/O, if MONC is on DOS can still get into a loop.

I still think the easiest way to avoid this problem is to avoid using BRUN inside Applesoft programs. Use BLOAD and CALL instead. But sometimes you may want to use BRUN, because you do not know in advance where the CALL address would be. One way to allow I/O inside your own program even though it is to be BRUN from inside an Applesoft program is to disconnect or bypass the hooks. You could output characters by JSR $FDF0, for example. But that would always go to the screen, and you may have a printer or an 80-column card or a modem hooked in, so that isn't a real solution. Another way is to dis-install the DOS hooks, by doing a JSR $9EE0 or the equivalent. The code at $9EE0 does this:

```
               LDX #3
       .1      LDA $AA53,X
               STA $36,X
               DEX
               BPL .1
               RTS
```

This unhooks DOS, but leaves any other I/O devices you have connected hooked in. After doing this step, your program can freely call COUT or RDKEY without DOS even knowing about it. You might also want to store a zero at $AA5E, to turn off MONC. Your program can terminate then by a JMP $3EA, which will restore the DOS hooks.

An alternative that seems to work is to save and restore the location where DOS saves the entering stack pointer. This is the culprit which causes the crippling loop. At $9FB6, just before returning to whoever entered DOS, the stack pointer gets reset to the value it had when DOS was entered. If you enter DOS while you are still in DOS, the first value is replaced with the second. Then the final return point is lost, and it is loop-city. Your program can save and restore $AA59, where the stack pointer is kept:

```
       YOUR.PROGRAM
               LDA $AA59       save DOS stack pointer
               PHA
               LDA #0          turn off MON C
               STA $AA5E

       ...do all your stuff, including I/O

               PLA
               STA $AA59
               RTS
```

This method has the advantage that your program can issue its own DOS commands by printing them, the way you would from Applesoft. For example, the following program will work when BRUN from inside Applesoft.

```
       .OR $1000
       .TF B.SHOW OFF
DEMONSTRATE
       LDA $AA59
       PHA
       LDY #0          issue DOS CATALOG command
.1     LDA MSG,Y
       JSR $FDED
       INY
       CPY #MSGSZ
       BCC .1
       LDA #0
       STA $AA5E    "NOMON C"
       PLA
       STA $AA59
       RTS
MSG    .HS 8D.84
       .AS -/CATALOG/
       .HS 8D
MSGSZ  .EQ *-MSG
```

```
100 PRINT CHR$(4)"MONC"
```

```
110 PRINT CHR$(4)"BRUN B.SHOW OFF"
120 PRINT "FINISHED"
```

However, that program will not work correctly if you just type "BRUN B.SHOW OFF" from the command mode. You will get a syntax error after the catalog displays, because the catalog command is left in the input buffer incorrectly. Oh well!

==============================================================================
DOCUMENT :AAL-8606:Articles:Front.Page.txt
==============================================================================

$1.80

Volume 6 -- Issue 9                                June, 1986

In This Issue...



So Soon?

Another issue of Apple Assembly Line already?  Well, readers sent in articles, Bob went on a writing binge, and we've managed to gain over a week in our efforts to get AAL back on schedule.  You should all actually receive this issue during the month of June!  One side effect of this acceleration is that Bill wasn't ready in time with the code to boot DOS 3.3 from his UniDisk 3.5.  It looks like next month for that program and article.

What, Not Yet?

Osborne/McGraw-Hill reports that their copies of 65816 Assembly Language Programming, by Michael Fischer, arrived today (6/3), so our orders should be shipped within two weeks.  We'll send them on to our customers just as soon as they arrive.  Simon & Schuster has taken over all of Prentice-Hall's titles, so they are now the ones we are bugging about Programming the 65816, by David Eyes.  The latest word from S & S is mid-July.  Sigh.

We understand that there is a 65816 book from Sybex in the stores, but the people who have seen it aren't very impressed, describing it as a 6502 book with some '816 information gleaned from the data sheets but few examples.

More Disk Utilities

We are now carrying the highly-regarded disk utility package Copy II Plus.  This includes disk and file copy programs, catalog and file handling utilities for both DOS and ProDOS, track and sector editing, and much more.  List price for all this is only $39.95, but we'll have it for just $35 + shipping.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228.  Phone (214) 324-2050.

Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.

==============================================================================
DOCUMENT :AAL-8606:Articles:MLI.Error.Hndlr.txt
==============================================================================

Generalized MLI System Error Handling.....Bob Sander-Cederlof

The ProDOS Machine Language Interface (MLI) returns an error code in the A-register if anything goes wrong. There are about 30 error codes, with values from $01 to $5A. BASIC.SYSTEM reduces the number of different error codes to 18, calling many of them simply "I/O ERROR". A nearly complete description of the error codes can be found in several references:

"Apple ProDOS--Advanced Features", pages 68-70.
"Beneath Apple ProDOS", pages 6-59 thru 6-61.
"ProDOS Technical Reference Manual", pages 77-79.

When I am working with a new program which has a lot of MLI calls, it is helpful to have one central error handler to print out the error information. Gary Little gives us such a subroutine on pages 66 and 67 of his "Apple ProDOS -- Advanced Features." Gary's program prints the message "MLI ERROR $xx OCCURRED AT LOCATION $yyyy", where xx is the hexadecimal error code and yyyy is the address of the next byte after the MLI call. You can mentally subtract 6 from the yyyy address to get the actual address of the JSR $BF00 that caused the error.

I assume you already know, if you are following me this far, that MLI calls take the form "JSR $BF00", followed by three data bytes. The first data byte is the opcode, and the other two are the address of the parameter block for the MLI call:

```
       JSR $BF00
       .DA #OPCODE,PARAMETERS
```

It would be nice if the general error handler would give us a little more information. First, I would like for it to print out the actual address of the JSR $BF00, rather than the return address. Second, I would like for it to print out the three bytes which follow the JSR $BF00.

First, I recoded Gary's routine so that it took a lot less space. (Littler than Little's!) I shortened the message and tightened the code. My version prints simply "AT" in place of "OCCURRED AT LOCATION." Then I used a message printing subroutine to print the two text strings, rather than the two separate loops he used. His took 83 bytes, mine only 56.

<<<listing of short version>>>

Next, I started adding the features I mentioned above. The final program takes 92 bytes, which is 9 more than Gary's. It displays the error message "MLI ERROR $xx AT $yyyy (op.addr)."

Lines 1080-1160 pick up the address MLI saved in the System Global Page, and sbtract six from it. The result is stored into the LDA $9999,Y instruction at line 1200. Horrors! Self-modifying code! The loop at lines 1180-1240 copies the three data bytes which follow the JSR $BF00 into the three variables at lines 1390-1410.

Lines 1260-1360 print out the error message. This loop differentiates between ASCII characters (bit 7 = 1) and data offsets (bit 7 = 0). The text to be printed is in lines 1430- 1550. Note that I used the negative ASCII form for the text, and .DA lines for the data bytes which will be printed in hexa- decimal. The expressions in those .DA lines compute an offset from the beginning of the subroutine, which will come out as a value less than $7F. I also used the value 00 to terminate the entire message. The $8D bytes are RETURN characters, to make sure the error message prints on a line by itself.

==============================================================================
DOCUMENT :AAL-8606:Articles:Protocol.Conv.txt
==============================================================================

Using Apple's Protocol Converter..........Bob Sander-Cederlof

The "Protocol Converter" is a firmware-controlled method of turning the //c disk port into a multi-drop peripheral bus able to support up to 127 external I/O devices. The bus connects devices which have enough intelligence: an "Integrated WOZ Machine" (IWM) chip, a 6502-type chip, RAM, and ROM. Data is transferred in a serial bit-stream at roughly 250,000 bits per second. So far, the only device anyone is building to run on the P/C bus is the Unidisk 3.5 from Apple.

As far as I have been able to determine, Apple's only published information about the protocol converter is in the Apple //c Technical Reference Manual, pages 114-142. The listing of the //c firmware in the same Manual also is informative. A prelim- inary document was available to developers, but most of the material is now given in the //c manual. Tom Weishaar ("Uncle DOS") promises a future article on the P/C in his "Open Apple" newsletter. (By the way, the June issue of "Open Apple" used the term "Smartport" as synonymous with "Protocol Converter".)

The Apple //e interface card for the UniDisk 3.5 also supports a "real" Protocol Converter. The Apple Memory Expansion Card, CirTech Flipster, and Applied Engineering RamFactor provide the same software interface with most of the features of the protocol converter for one I/O device (the memory card itself).

Apple briefly mentions the Protocol Converter in the Apple Memory Expansion Card manual (Appendix B, last paragraph), but warns against using it. They say "using the assembly-language protocol is fairly complicated". Nevertheless, a significant amount of the Apple firmware is used to implement the protocol converter features. It appears that someone inside Apple intends that the P/C will be included in the firmware of most future block-oriented devices. From a software stand-point, it could be used regardless of whether the actual hardware used the IWM-based bus, a SCSI bus, or no bus at all.

In order to use the protocol converter firmware, you need first to find it. The first step in finding it is to find which slot it is in. All of the cards with P/C firmware (so far) are also cards which control or emulate disk drives and have firmware supporting the ProDOS device driver protocol. Cards with ProDOS device driver firmware can be identified by four bytes: $Cs01 = $20, $Cs03 = $00, $Cs05 = $03, and $Cs07 = $00. The first three bytes in that list are the same for all disk drive controllers. The zero value at $Cs07 distinguishes it as a disk controller with protocol converter firmware.

The next step is to find the entry point in the firmware for protocol converter calls. The byte at $CsFF is the key. That byte is the offset in the firmware page for ProDOS calls. If $CsFF = $45, for example, ProDOS device driver calls would be "JSR $Cs45". To get the address of the protocol converter entry point, add 3 to the ProDOS entry point. In my example, "JSR $Cs48" would enter the protocol converter firmware. The actual value will probably be different for each kind of card, so you have to use software to find out what it is.

A program to find the slot and build the address of the protocol converter could look like this:

```
pcaddr  .eq $01,$02
find.pc lda #0
        sta pcaddr
        ldx #$C7   slot = 7 to 1 step -1
.1      stx pcaddr+1
        ldy #7
.2      lda (pcaddr),y  $Cs07,05,03,01
        cmp pc.sig,y
        beq .3
        dex
        cpx #$c1
        bcs .1     try next slot
        sec        signal could not find pc
        rts

.3      dey
        dey
        bpl .2
        lda (pcaddr),y   $CsFF
        adc #2     carry was set
        sta pcaddr
        rts        carry clear signals pc found

pc.sig  .HS FF.20.FF.00.FF.03.FF.00
```

Once you have the address of the protocol converter firmware, you call it in a manner similar to ProDOS MLI calls. You must plug the address of the protocol converter entry into a "JSR" instruction, which is followed by a one-byte command code and a two-byte address. The command code is a number from $00 to $09 which specifies which action you want the protocol converter to take. The address is the address of a parameter block, which provides additional information for processing the command, or a place for the information returned by the command. After the protocol converter has finished processing your command, it returns control to the next byte after the pointer to the parameter block. If carry is clear, there was no error. If carry is set, the A-register contains an error code.

Since my FIND.PC program left the address in two page zero locations, we could simply put a JMP opcode ($4C) in front of the address to make it into a JMP instruction. Then our calls to the protocol converter would look like this:

```
callpc  .eq $00         (just before pcaddr)
        jsr find.pc
```

```
                bcs ...         ...no pc found
                lda #$4C        JMP opcode
                sta callpc
                ...     ...other code
                jsr callpc
                .da #cmd,parameters
                ...     ...more code
```

Apple warns programmers NOT to use any page zero locations when calling the protocol converter firmware, saying that some page zero locations are used by that firmware. They do not say what locations they use, but my investigations show that they use bytes in the range from $40 to $4F. What they do is push those on the stack, put in their own data, and at the end restore the original contents from the stack. They use an awful lot of stack, up to 35 bytes. (The RamFactor firmware uses no more than 17 bytes of stack for protocol converter calls, including the two used by your JSR.) If you want be safe rather than possibly sorry, you can copy the PCADDR bytes up into your own program. You could even plug them into every JSR which calls protocol converter. A cleaner way might be like this:

```
                jsr find.pc
                bcs ...         ...no pc found
                lda pcaddr
                sta callp+1
                lda pcaddr+1
                sta callpc+2
                ...
                jsr callpc
                .da #cmd,parameters
                ...
        callpc  jmp *   address filled in
```

Description of Protocol Converter Commands

Apple defines ten commands for the protocol converter firmware. These are not necessarily identical in function for all devices which use the protocol converter. In fact, Apple's memory card uses two of the commands differently than the UniDisk 3.5 does. The protocol converter firmware in the RamFactor functions exactly the same as that in the Apple Memory Expansion Card.

The following chart summarizes the ten commands as implemented in the Apple Memory Expansion Card and RamFactor firmware. A more detailed description of each command follows the chart. I am particularly pointing this at the memory cards rather than the Unidisk 3.5, because I believe these cards will be more popular with hackers like you and me. Furthermore, the Unidisk 3.5 information is available in the //c manual, but Apple has not released this detail for owners of the memory card.

```
   Parameters:   +0  +1  +2   +3   +4   +5   +6   +7   +8
              cmd cnt unit
PC Status   $00   3   0 bufl bufh code
```

```
RAM Status  $00   3   1 bufl bufh code
Read Block  $01   3   1 bufl bufh blkh blkm blkl
Write Block $02   3   1 bufl bufh blkh blkm blkl
Format      $03   1   1
Control     $04   3 0/1 bufl bufh code
Init        $05   1 0/1
Read Bytes  $08   4   1 bufl bufh cnth cntl adrh adrm adrl
Write Bytes $09   4   1 bufl bufh cnth cntl adrh adrm adrl

Error Codes $01 Command not $00-$05,$08, or $09
            $04 Wrong parameter count
            $11 Invalid Unit Number
            $21 Invalid Status or Control code
            $2D Block Number too large
```

PC Status (cmd $00, unit $00, code $00): reads the status of the protocol converter itself into your buffer. The status of a memory card is always 8 bytes, with the first byte = $01 and all the others = $00. Also returns with $08 in the X-register and $00 in the Y-register. ($0008 is the number of bytes stored in your buffer.) This is of value only for compatibi- lity with other devices supporting protocol converter firmware.

RAM Status (cmd $00, unit $01, code $00 or $03): reads the status of the memory card into your buffer. Code $00 stores four bytes: the first is always $F8, and the other three are the number of blocks in the current partition (lo, mid, hi order). (Y,X) will equal ($00,$04) when it is finished, showing that four bytes were stored. Code $03 will store 25 bytes: the first four are the same as code $00 returned; the next 17 are the name of the card in "ProDOS Volume Name" format (length of name in first byte, ASCII characters of name with hi-bit off, padded with blanks); and finally, four zero bytes. The card name is "RAMCARD". (Y,X) will return ($00,$19) when finished, indicating that 25 bytes were stored.

Obviously, the Status commands will operate differently on a real P/C bus, and the actual details will vary according to the device you interrogate.

Read Block (cmd $01): reads the specified block from the memory card. (In RamFactor, the block number is relative, inside the currently selected RamFactor partition.) You can read a block into a buffer in //e Auxiliary Memory by calling the P/C with the RAMWRT soft-switch set to AuxMem.

Write Block (cmd $02): writes the specified block from your buffer into the memory card. (In RamFactor, the block number is relative, inside the current RamFactor partition.) If you are careful and follow all the rules, you can write a block from a buffer in Auxiliary Memory by calling the protocol con- verter with the RAMRD soft-switch set to AuxMem. You have to put the code that sets the RAMRD switch and calls the protocol converter, and its parameter block, in zero-page or stack-page motherboard RAM ($0000-01FF), or in the language card RAM area. Or, you can have both RAMRD and RAMWRT set for AuxMem and be executing a program from within AuxMem. I always have a conceptual battle dealing with this kind of bank switching.

Format (cmd $03): does nothing in a memory card.

Control (cmd $04): does nothing in a memory card. If the code is not $00, you get error code $21. The buffer is never used.

Init (cmd $05): does nothing in a memory card.

Open or Close (cmd $06 or $07): cause error code $01 in a memory card. These commands only apply to character-oriented devices, and memory is a block-oriented device (so says Apple). Maybe someday someone will build a peripheral which is character-oriented and includes P/C firmware.

Read Bytes (cmd $08): reads a specified number of bytes starting at a specified memory-card address into your buffer. The byte count may be as high as $FFFF, but this would obviously wreak havoc inside your Apple. No checks are made inside the protocol firmware for reasonableness of the buffer address or the byte count, so be careful. You would NEVER read into a buffer in the I/O address range ($C000-$CFFF).

The memory-card address may be as high as $7FFFFF. (In RamFactor, the address is relative inside the current partition.) This corresponds to a total of 8 megabytes, which is only half the maximum capacity of a RamFactor card. Apple has arbitrarily limited us to this maximum, because they use the top bit of the card address to specify whether the buffer is in MainMem (bit 23 = 0) or AuxMem (bit 23 = 1). (Bit 23 of the address is bit 7 of the last byte of the parameter block.)

Write Bytes (cmd $09): writes a specified number of bytes from your buffer starting at a specified memory-card address. The details of byte count, buffer location, and memory-card address are the same as for the Read Bytes ($08) command.

The Unidisk 3.5 firmware interprets commands $08 and $09 differently. Unidisk uses this pair to read and write Macintosh disks, which have 524-byte blocks.

All of the RamFactor protocol converter commands operate within the current active partition. In the Apple card there is only one partition (the whole card). RamFactor has nine partitions, and you are always in one of them. If you start with a blank card, the first call to the RamFactor protocol converter will set up the first partition with all but 1024 bytes, make that partition the current active one, and empty all the others.

Bill Morgan's articles on interfacing the Unidisk 3.5 with DOS 3.3 illustrate the use of protocol converter calls with that device. The real power of the protocol converter concept will not be realized until a variety of devices are available which use it. Maybe its real future is bound up in the new 65816-based Apple //.

```
================================================================
DOCUMENT :AAL-8606:Articles:Rindsbergs.CRC.txt
================================================================
```

Practical Application of CRC....................Don Rindsberg

When I read Bob S-C's article on CRC in the February 1986 AAL, I said, "Very interesting, but who needs it". Well, it wasn't long before I ran into a real need myself!

I bought a used IBM PC-Jr and wanted to put my own routines in an auto-start ROM cartridge. After some sleuthing, I found that the power-up routine checks for signature bytes. If they are present, the routine checks the ROM's CRC, which must be $0000 or the machine locks up.

Not knowing the 65802 opcodes that Bob used, and being quite familiar with the 8088 language, I decided to translate the PC-Jr's CRC routine from "8088 dis-assembly language" to "plain vanilla 6502-ese". I simulated the 8088's registers with Apple RAM, and wrote subroutines for some of the 16-bit 8088 instructions.

Now here's what I think is strange about CRC's. If you pass all bytes of a set of data through the CRC generator and then the two CRC bytes themselves, the total CRC result is $0000! The PC-Jr add-on ROMs have the program in all except the last two bytes and the CRC of the program in those last two, so the total CRC for the entire ROM is $0000.

My 6502 code requires you to enter the start in Apple RAM and the length of the ROM data. For example, for a program starting at $2000 in Apple RAM, destined to be blown into a 2716 EPROM (2048 bytes), you would enter an address of $2000 and a length of $0800. These two values go into the first four bytes of the Apple zero page, so you can use a monitor instruction from inside the S-C Assembler like this:

```
       :$00:00 20 00 08
```

My program runs a CRC calculation on all but the last two bytes, and then prints out what the resulting CRC code is. If you store the CRC value in the last two bytes of the ROM image, add two to the length, and re-run my program, the result should be 0000. In a particular example with a 2716, it might look like this:

```
       :$00:00 20 00 08      (set up address & length )
       :$800G                (run CRC calculation     )
       82DF                  (value of CRC computed   )
       :$20FE:82 DF          (store CRC in EPROM image)
       :$02:02               (increase length by two  )
       :$800G                (run CRC calcualtion     )
       0000                  (it worked!              )
```

My routines will not win the speed or elegance contests, but they give me the data!

If you want another check on your coding, run a CRC calculation on the Applesoft $D000 ROM with length $0800.  You should get $D01E if you have an Apple II+ or original //e version.  The enhanced //e gives a CRC of $3BD4 because of some small changes Apple made.

By the way, I use my Apple to generate assembly language code for the IBM PC line.  I created an 8086/8088 cross assembler based on the S-C Assembler for the purpose.  Contact me if you need a tool like this: Don Rindsberg, The Bit Stop, 5958 S. Shenandoah, Mobile, Alabama 36608.  Or call at (205) 342-1653.

```
==============================================================================
DOCUMENT :AAL-8606:Articles:Stack.Relative.txt
==============================================================================
```

Using the 65816 Stack Relative Mode........Bob Sander-Cederlof

The 65802 and 65816 have two new address modes that allow you to reach into the stack. The "offset,S" mode lets you access position relative to the stack pointer, and the "(offset,S),Y" mode lets you access data indirectly through an address that is on the stack. The new address modes are available even when the 65802/16 is in the "emulation" mode.

The hardware adds the value of the offset to the current stack pointer to form an effective address. The stack pointer is always pointing one address below the end of the stack. Thus, an address of "1,S" points to the first item on the stack.

These new modes lead to interesting programming possibilities. When you design a subroutine, you have to decide how you are going to pass parameters into and out of the subroutine. Usually we try to use the A, X, and Y registers first. Another method puts the data or the address of the data after the JSR that calls the subroutine. ProDOS MLI calls use this method:

```
       JSR $BF00
       .DA #$C1,PARMS
```

In another method you push data or data addresses on the stack, and then call the subroutine. This is the preferred method in some computers, but not the 6502. The new modes make this mode work nicely in the 65802/16, though.

I coded up two examples to show how you can use the new modes, both message printing subroutines. The calling method requires telling the subroutine where to find a variable length message. In the first one (lines 1070-1330), I chose to push the address of the text on the stack before calling the printing routine. In the second example (lines 1340-1640), I used the method of storing the message text immediately after the JSR instruction.

Lines 1070-1110 print out two messages, using the first technique. I use the PEA (Push Effective Address) instruction to put the address of the first byte of the message text on the stack. This instruction pushes first the high byte, then the low byte, of the value of the operand. (I think I would prefer to have called it "PSH #value", because that is the effect. Then the PEI opcode, which pushes two bytes from the direct page, could be "PSH zp". But, nobody asked me.)

Anyway, let's look at the PRINT.IT subroutine. When the subroutine starts looking at the stack, it looks like this:

```
          |  msg addr lo  | 4,S
          | ------------- |
```

```
          |  msg addr hi  | 3,S
          | ------------- |
          |  ret addr lo  | 2,S
          | ------------- |
          |  ret addr hi  | 1,S
          | ------------- |
          |               |<---Stack Pointer
```

The LDA (3,S),Y instruction at line 1240 takes the address at 3,S and 4,S (which is the address of the first byte of the message) and adds the Y-register to it; then the LDA opcode picks up the message byte. After printing all the message and finding the terminating 00 byte, lines 1290-1320 move the return address up two slots higher in the stack (over the top of the message address). At the same time, the original copy of the return address is removed from the stack. Then a simple RTS takes us back to the caller, with a clean stack.

The second example uses a "message buried in the code" method. When PRINT.MSG looks at the stack, only the return address is there. The return address points to the third byte of the JSR instruction, one byte before the message text. Therefore the printing loop in lines 1500-1550 starts with Y=1. Lines 1560- 1620 add the message length to the return address, so that an RTS opcode will return to the caller just past the message.

<<<code here>>>

It might be instructive to look at how these two examples could be code in a plain 6502 environment. First, we must replace the PEA opcodes in lines 1070 and 1090 with the following:

```
       LDA #MESSAGE
       PHA
       LDA /MESSAGE
       PHA
```

Then PRINT.IT would require using temporary memory somewhere or writing self-modifying code. With a pointer in page zero, it could work like this:

<<<6502 version of print.it>>>

PRINT.MSG also can be written in pure 6502 code with either self-modifying code or a pointer in page zero. Here is the self-modifying version:

<<<6502 version of print.msg>>>

```
==============================================================================
DOCUMENT :AAL-8606:Articles:Toggling.Values.txt
==============================================================================
```

Toggling Between Two Values.....................Jan Eugenides

In the course of my job as Technical Editor for MicroSPARC, Inc. (the publishers of Nibble and Nibble Mac magazines), I am often called upon to modify programs that we are going to publish to make them compatible with configurations other than the one the author originally wrote for. Recently, I had to change a program to toggle between Drive 1 and Drive 3, rather than Drive ! and Drive 2 as it was originally coded. Here is the original subroutine which toggled the drive number stored in a variable named CD:

```
       TOGGLE.DRIVE
               LDA CD
               CMP #1
               BEQ .1
               LDA #1
               STA CD
               BNE .2
       .1      INC CD
       .2      RTS
       CD      .BS 1
```

This code takes a total of 19 bytes, including the variable CD. My task was to exactly replace this routine with one which would toggle between 1 and 3 rather than 1 and 2. It had to use the same number of bytes, or less. It looks easy enough, but I couldn't come up with a solution. All my routines required one or two more bytes. I finally took the easy way out and patched it with a JMP to a free space near the end of the program, and put my code there. It works, but is there a shorter way?

Bob, you are the best code squeezer around, so I thought I'd give the problem to you. You'll undoubtedly come up with some sneaky code that does the trick in three bytes or less!

An Answer for Jan.........................Bob Sander-Cederlof

I don't know if I am the best code squeezer or not, but I can't squeeze it all the way to three bytes! My best attempt is nine bytes:

```
       TOGGLE.DRIVE
               LDA #1
       CD      .EQ *-1
               EOR #2
               STA CD
               RTS
```

In general, you can toggle back and forth between any two values by using the EOR instruction. The toggle constant is simply the exclusive-or of the two values. For example, to toggle back and forth between the values $A0 and $B2, I would use "EOR #$12".

My subroutine changes 1 to 3 and 3 to 1, as you requested. However, it is not functionally identical to the original code. The original code did not store the variable CD inside an immediate-mode LDA, as I did. If that troubles you, simply change that line to "LDA CD" and add the line "CD .BS 1" at the end. The result takes ten bytes, still well under the limit.

The original code also always had the side-effect of setting carry status, so you might need to add a "SEC" instruction. I doubt it, because the original code would be very weird if it depended on this side-effect.

The original code not only changed 3 to 1, but also changed any other value not already 1 into 1. This is also probably not a necessary feature, because prior code should have made sure that we started with a valid drive number.

I came up with several other approaches to the problem, all of which are shorter than the original subroutine:

```
       TOGGLE.DRIVE
               LSR CD     3 TO 1, OR 1 TO 0
               BNE .1     IT WAS 3 TO 1
               LDA #3     CHANGE 1 TO 3
               STA CD
       .1      RTS

       TOGGLE.DRIVE
               CLC
               LDA CD
               ADC #2     1 TO 3, OR 3 TO 5
               AND #3     5 TO 1
               RTS
```

None of these are particularly tricky or sneaky. In fact, the first and shortest one is the most straightforward. What would be tricky or sneaky is if the original author depended on the hidden side-effects in his subroutine.

```
==================================================================
DOCUMENT :AAL-8606:DOS3.3:Bell.Demo.Src.txt
==================================================================

 1000  * BRUN DEMO
 1010  *SAVE BELL DEMO SOURCE
 1020  *----------------------------
 1030  * DEMO OF BRUN'ING A ML PROG
 1040  * BY RINGING A BELL
 1050  *
 1060  * DOS IS DISCONNECTED
 1070  * TO ALLOW I/O WITHOUT
 1080  * DISRUPTING PROPER RETURN.
 1090  *--------------------------------
 1100  COUT1   .EQ $FDF0    SCREEN OUTPUT
 1110  KEYIN   .EQ $FD1B    KEYBOARD INPUT
 1120  *--------------------------------
 1130          .OR $6A00
 1140  DEMO
 1150          LDX #0       BEFORE ANY I/O,
 1160  .10     LDA $36,X     DISCONNECT DOS
 1170          PHA           BY PUSHING $36.39
 1180          LDA PTRS,X    ONTO STACK,
 1190          STA $36,X     & REPLACING
 1200          INX           WITH COUT1/KEYIN
 1210          CPX #4
 1220          BNE .10
 1230
 1240          JSR $FF3A    RING THE BELL
 1250
 1260          LDX #3       RECONNECT DOS
 1270  .90     PLA           BY PULLING
 1280          STA $36,X     $36.39 FROM
 1290          DEX           THE STACK.
 1300          BPL .90
 1310          RTS
 1320  *--------------------------------
 1330  * REPLACEMENT I/O POINTERS
 1340  *--------------------------------
 1350  PTRS    .DA COUT1,KEYIN
```

==============================================================================
DOCUMENT :AAL-8606:DOS3.3:Butterill.Demo.txt
==============================================================================

( DTC removed -- lots of garbage characters )

```
==============================================================
DOCUMENT :AAL-8606:DOS3.3:Butterill.Div.txt
==============================================================

 1000  *SAVE BUTTERILL'S DIVIDE
 1010  *--------------------------------
 1020  * 16 BIT DIVIDE WITH REMAINDER
 1030  * DIVIDE B BY A
 1040  * LEAVES QUOTIENT IN B,
 1050  *       REMAINDER IN A
 1060  *--------------------------------
 1070  *   TIMING:  A=$0000 -- 39 cycles
 1080  *            B>$7FFF -- 71 or 74 cycles
 1090  *            A=B     -- 82 cycles
 1100  *       A=1,B=$FFFF -- 676 cycles
 1110  *--------------------------------
 1120  A      .EQ 0,1      DIVISOR, REMAINDER
 1130  B      .EQ 2,3      DIVIDEND, QUOTIENT
 1140  *--------------------------------
 1150         .OP 65802
 1160  DIV16
 1170         CLC          ENTER FROM 6502
 1180         XCE          NATIVE MODE
 1190         REP #$20     A-REG 16 BITS
 1200         LDX #0       START SCALE CNTR
 1210         LDA A        GET DIVISOR
 1220         BEQ .90      ...ZERO DIVISOR
 1230         BMI .30      ...DIVISOR > $7FFF
 1240  *---SCALE DIVISOR----------------
 1250  .10    CMP B        ALIGN A TO LEFT
 1260         BCS .20      UNTIL > B
 1270         INX             OR BIT 15 SET
 1280         ASL          & COUNT IN X
 1290         BPL .10
 1300  .20    STA A        SCALED DIVIDEND
 1310  *---START SUBTRACTING------------
 1320  .30    LDA B        GET DIVIDEND
 1330         STZ B        CLEAR QUOTIENT
 1340  .40    CMP A        REPEATED CONDITIONAL
 1350         BCC .50       SUBTRACTION.
 1360         SBC A
 1370  .50    ROL B        ROL IN 1 IF SUBT.
 1380         LSR A               0 IF NO SUBT.
 1390         DEX
 1400         BPL .40
 1410         STA A        REMAINDER
 1420  *---RETURN TO CALLER-------------
 1430  .60    SEC          EXIT TO 6502
 1440         XCE
 1450         RTS
 1460  *---FOR X/0, GIVE 0,0 ANSWER-----
 1470  .90    STA B        DIVISION BY ZERO
 1480         BRA .60
```

```
1490  *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8606:DOS3.3:Butterill.Mult.txt
==========================================================================

```
 1000  *SAVE BUTTERILL'S MULTIPLY
 1010  *--------------------------------
 1020  * 16 BIT MULTIPLY FOR 65802
 1030  * MULTIPLIES A BY B
 1040  * LEAVES ANSWER IN A & B
 1050  *--------------------------------
 1060  A      .EQ 0,1      MULTIPLIER, PRODUCT-LO
 1070  B      .EQ 2,3      MULTIPLICAND, PRODUCT-HI
 1080  *--------------------------------
 1090  *   TIMING:  B=$0000 -- 27 CYCLES
 1100  *            A=$0000 -- 335 CYCLES
 1110  *            A=$FFFF -- 383 CYCLES
 1120  *   (INCLUDING JSR AND RTS)
 1130  *--------------------------------
 1140         .OP 65802
 1150  MULT16
 1160         CLC          ENTER FROM 6502
 1170         XCE
 1180         REP #$20
 1190         LDA B        IF B ZERO,
 1200         BEQ .90      THEN BY-PASS
 1210         DEC B
 1220         LDA ##0000
 1230         LDX #16      FOR 16 BITS
 1240         CLC          FOR 17'TH CYCLE
 1250  .10    ROR          ROLL OUT PRODUCT BIT
 1260         ROR A        ROLL IN 'PLIER BIT
 1270         BCC .20
 1280         ADC B
 1290  .20    DEX
 1300         BPL .10      CYCLES 17 TIMES
 1310         STA B
 1320  .30    SEC          EXIT TO 6502
 1330         XCE
 1340         RTS
 1350  .90    STA A        PROCEDURE FOR B=0
 1360         BRA .30
 1370  *--------------------------------
```

```
=========================================================================
DOCUMENT :AAL-8606:DOS3.3:Div16.Demo.Src.txt
=========================================================================

 1000  * DIV16 DEMO
 1010  *SAVE DIV16 DEMO SOURCE
 1020  *-------------------------------
 1030  * DEMO OF BRUN'ING A ML PROG
 1040  * USING DIV16
 1050  *
 1060  * DOS IS DISCONNECTED
 1070  * TO ALLOW I/O WITHOUT
 1080  * DISRUPTING PROPER RETURN
 1090  *-------------------------------
 1100          .OP 65802
 1110          .OR $6A00
 1120  *-------------------------------
 1130  COUT1   .EQ $FDF0    SCREEN OUTPUT
 1140  KEYIN   .EQ $FD1B    KEYBOARD INPUT
 1150  *-------------------------------
 1160  AL      .EQ 0
 1170  AH      .EQ 1
 1180  BL      .EQ 2
 1190  BH      .EQ 3
 1200  DFLG    .EQ 4        DELIMITER FLAG
 1210  GETLN1  .EQ $FD6F    INPUT LINE TO BUFFER
 1220  PRNTAX  .EQ $F941    OUTPUT A,X AS HEX
 1230  COUT    .EQ $FDED    OUTPUT A AS CHAR
 1240  CROUT   .EQ $FD8E    OUTPUT CR
 1250  *-------------------------------
 1260  DEMO
 1270          LDX #0       BEFORE ANY I/O,
 1280  .10     LDA $36,X     DISCONNECT DOS
 1290          PHA           BY PUSHING $36.39
 1300          LDA PTRS,X    ONTO STACK,
 1310          STA $36,X     & REPLACING
 1320          INX           WITH COUT1/KEYIN
 1330          CPX #4
 1340          BNE .10
 1350
 1360          JSR CROUT
 1370  .20     JSR GETLN1   INPUT LINE TO BUFFER
 1380          JSR HEXVALS  EXTRACT HEX VALUES
 1390          CPY #1       IF NULL LINE,
 1400          BEQ .80         THEN EXIT
 1410          JSR PROG     DIVIDE
 1420          LDA BH
 1430          LDX BL
 1440          JSR PRNTAX   DISP QUOTIENT
 1450          LDA #","
 1460          JSR COUT     DISP ','
 1470          LDA AH
 1480          LDX AL
```

```
1490          JSR PRNTAX   DISP REMAINDER
1500          JSR CROUT
1510          JMP .20
1520
1530  .80     LDX #3       RECONNECT DOS
1540  .90     PLA           BY PULLING
1550          STA $36,X     $36.39 FROM
1560          DEX           THE STACK.
1570          BPL .90
1580          RTS
1590  *--------------------------------
1600  * REPLACEMENT I/O POINTERS
1610  *--------------------------------
1620  PTRS    .DA COUT1,KEYIN
1630
1640  *--------------------------------
1650  * READ TWO HEX 16-BIT WORDS
1660  * FROM INPUT BUFFER. (AFTER WOZ)
1670  *--------------------------------
1680  BUFF    .EQ $200
1690  *--------------------------------
1700  HEXVALS
1710          LDY #0       CLEAR BUFFER INDEX
1720          STY DFLG     CLEAR DELIMITER FLAG
1730  .10     LDA #0       CLEAR A
1740          STA AL
1750          STA AH
1760  .20     LDA BUFF,Y   GET CHAR FROM BUFFER
1770          INY
1780          CMP #$8D     = CR ?
1790          BNE .30
1800          RTS
1810
1820  .30     EOR #$B0     CONVERT ASCII TO HEX
1830          CMP #$0A
1840          BCC .40      IF 0-9
1850          ADC #$88
1860          CMP #$FA
1870          BCS .40      IF A-F
1880          LDA DFLG     ELSE ASSUME
1890          BNE .10       CHAR IS
1900          LDA AL        A DELIMITER
1910          STA BL       MOVE A TO B
1920          LDA AH        IF NOT REPEATED
1930          STA BH        DELIMITER
1940          DEC DFLG     SET DELIMITER FLAG
1950          JMP .10
1960
1970  .40     ASL          SHIFT NIBBLE
1980          ASL           TO LEFT HAND
1990          ASL           SIDE.
2000          ASL
2010          LDX #4       & ROL INTO MEMORY
2020  .50     ASL
```

```
2030          ROL AL
2040          ROL AH
2050          DEX
2060          BNE .50
2070          STX DFLG     CLEAR DELIMITER FLAG
2080          JMP .20
2090  *--------------------------------
2100  * SUBROUTINE
2110  *--------------------------------
2120  PROG    .IN BUTTERILL'S DIVIDE
```

```
==============================================================================
DOCUMENT :AAL-8606:DOS3.3:Mult16.Demo.Src.txt
==============================================================================

 1000  * MULT16 DEMO
 1010  *SAVE MULT16 DEMO SOURCE
 1020  *--------------------------------
 1030  * DEMO OF BRUN'ING A ML PROG
 1040  * USING MULT16
 1050  *
 1060  * DOS IS DISCONNECTED
 1070  * TO ALLOW I/O WITHOUT
 1080  * DISRUPTING PROPER RETURN
 1090  *--------------------------------
 1100          .OP 65802
 1110          .OR $6A00
 1120  *--------------------------------
 1130  COUT1   .EQ $FDF0    SCREEN OUTPUT
 1140  KEYIN   .EQ $FD1B    KEYBOARD INPUT
 1150  *--------------------------------
 1160  AL      .EQ 0
 1170  AH      .EQ 1
 1180  BL      .EQ 2
 1190  BH      .EQ 3
 1200  DFLG    .EQ 4        DELIMITER FLAG
 1210  GETLN1  .EQ $FD6F    INPUT LINE TO BUFFER
 1220  PRNTAX  .EQ $F941    OUTPUT A,X AS HEX
 1230  CROUT   .EQ $FD8E    OUTPUT CR
 1240  *--------------------------------
 1250  DEMO
 1260          LDX #0       BEFORE ANY I/O,
 1270  .10     LDA $36,X     DISCONNECT DOS
 1280          PHA           BY PUSHING $36.39
 1290          LDA PTRS,X    ONTO STACK,
 1300          STA $36,X     & REPLACING
 1310          INX           WITH COUT1/KEYIN
 1320          CPX #4
 1330          BNE .10
 1340
 1350          JSR CROUT
 1360  .20     JSR GETLN1   INPUT LINE TO BUFFER
 1370          JSR HEXVALS  EXTRACT HEX VALUES
 1380          CPY #1       IF NULL LINE,
 1390          BEQ .80         THEN EXIT
 1400          JSR PROG     MULTIPLY
 1410          LDA BH
 1420          LDX BL
 1430          JSR PRNTAX   DISP HI-16
 1440          LDA AH
 1450          LDX AL
 1460          JSR PRNTAX   DISP LO-16
 1470          JSR CROUT
 1480          JMP .20
```

```
1490
1500  .80     LDX #3       RECONNECT DOS
1510  .90     PLA           BY PULLING
1520          STA $36,X     $36.39 FROM
1530          DEX           THE STACK.
1540          BPL .90
1550          RTS
1560  *---------------------------------
1570  * REPLACEMENT I/O POINTERS
1580  *---------------------------------
1590  PTRS    .DA COUT1,KEYIN
1600
1610  *---------------------------------
1620  * READ TWO HEX 16-BIT WORDS
1630  * FROM INPUT BUFFER. (AFTER WOZ)
1640  *---------------------------------
1650  BUFF    .EQ $200
1660  *---------------------------------
1670  HEXVALS
1680          LDY #0       CLEAR BUFFER INDEX
1690          STY DFLG     CLEAR DELIMITER FLAG
1700  .10     LDA #0       CLEAR A
1710          STA AL
1720          STA AH
1730  .20     LDA BUFF,Y   GET CHAR FROM BUFFER
1740          INY
1750          CMP #$8D     = CR ?
1760          BNE .30
1770          RTS
1780
1790  .30     EOR #$B0     CONVERT ASCII TO HEX
1800          CMP #$0A
1810          BCC .40      IF 0-9
1820          ADC #$88
1830          CMP #$FA
1840          BCS .40      IF A-F
1850          LDA DFLG     ELSE ASSUME
1860          BNE .10       CHAR IS
1870          LDA AL        A DELIMITER.
1880          STA BL       MOVE A TO B
1890          LDA AH        IF NOT REPEATED
1900          STA BH        DELIMITER
1910          DEC DFLG     SET DELIMITER FLAG
1920          JMP .10
1930
1940  .40     ASL          SHIFT NIBBLE
1950          ASL           TO LEFT HAND
1960          ASL           SIDE.
1970          ASL
1980          LDX #4       & ROL INTO MEMORY
1990  .50     ASL
2000          ROL AL
2010          ROL AH
2020          DEX
```

```
2030          BNE .50
2040          STX DFLG     CLEAR DELIMITER FLAG
2050          JMP .20
2060  *--------------------------------
2070  * SUBROUTINE
2080  *--------------------------------
2090  PROG    .IN BUTTERILL'S MULTIPLY
```

```
=============================================================================
DOCUMENT :AAL-8606:DOS3.3:ROM.CRC.Calc.txt
=============================================================================
```

```
 1000  *SAVE ROM CRC CALCULATION
 1010  *--------------------------------
 1020  LOCN   .EQ $00,01   ENTER DATA LOCN (L/H)
 1030  SIZE   .EQ $02,03   ENTER ROM SIZE (L/H)
 1040  AL     .EQ $04      SIMULATED 8088 REGISTERS
 1050  AH     .EQ $05
 1060  BL     .EQ $06
 1070  BH     .EQ $07
 1080  CL     .EQ $08
 1090  CH     .EQ $09
 1100  DL     .EQ $0A
 1110  DH     .EQ $0B
 1120  PTR    .EQ $0C,0D   WORK POINTER
 1130  CTR    .EQ $0E,0F   BYTE COUNTER
 1140  *--------------------------------
 1150  PRNTAX .EQ $F941
 1160  *--------------------------------
 1170         .OR $300
 1180  *--------------------------------
 1190  START  LDA LOCN     SETUP POINTER
 1200         STA PTR           TO ROM IMAGE
 1210         LDA LOCN+1
 1220         STA PTR+1
 1230  *--------------------------------
 1240         SEC          GET BYTE COUNT - 2
 1250         LDA SIZE
 1260         SBC #2
 1270         STA CTR
 1280         LDA SIZE+1
 1290         SBC #0
 1300         STA CTR+1
 1310  *--------------------------------
 1320         LDY #$FF     START CRC AT $FFFF
 1330         STY DL
 1340         STY DH
 1350         INY          Y=0
 1360         STY AH       INIT AH REG
 1370  *--------------------------------
 1380  .1     LDA (PTR),Y  GET NEXT BYTE
 1390         JSR FOLD.BYTE.INTO.CRC
 1400         INC PTR      BUMP THE WORK POINTER
 1410         BNE .2
 1420         INC PTR+1
 1430  .2     LDA CTR      DECREMENT THE BYTE COUNT
 1440         BNE .3
 1450         DEC CTR+1
 1460  .3     DEC CTR
 1470         LDA CTR      TEST IF FINISHED
 1480         ORA CTR+1
```

```
1490        BNE .1       ...KEEP GOING
1500        LDX DL       DISPLAY THE RESULT
1510        LDA DH
1520        JMP PRNTAX
1530 *---------------------------------
1540 FOLD.BYTE.INTO.CRC
1550        EOR DH
1560        STA DH
1570        STA AL
1580        JSR ROLAX4   8088 "ROL AX,C"
1590        JSR EORAD    8088 "EOR DX,AX"
1600        JSR ROLAX1   8088 "ROL AX,1"
1610        LDA DH       SWAP BYTES IN REG-D
1620        LDX DL
1630        STX DH
1640        STA DL
1650        JSR EORAD    8088 "EOR DX,AX"
1660        JSR RORAX4   8088 "ROR AX,C"
1670        LDA AL
1680        AND #$E0
1690        STA AL
1700        JSR EORAD    8088 "EOR DX,AX"
1710        JSR RORAX1   8088 "ROR AX,1"
1720        LDA AL
1730        EOR DH
1740        STA DH
1750        RTS
1760 *---------------------------------
1770 *   SIMULATE 8088 "ROL AX,C"
1780 *---------------------------------
1790 ROLAX4 JSR ROLAX1   SHIFT 4 BITS BY SHIFTING
1800        JSR ROLAX1        1 BIT 4 TIMES
1810        JSR ROLAX1
1820 *---------------------------------
1830 *   SIMULATE 8088 "ROL AX,1"
1840 *---------------------------------
1850 ROLAX1 LDA AL       8088 "ROL" SHIFTS END AROUND
1860        ASL          WITHOUT LEAVING A BIT IN CARRY
1870        ROL AH
1880        BCC .1       6502 DOES LEAVE A BIT IN CARRY,
1890        ORA #$01     SO LETS MERGE CARRY IN HERE.
1900 .1     STA AL
1910        RTS
1920 *---------------------------------
1930 *   SIMULATE 8088 "ROR AX,C"
1940 *---------------------------------
1950 RORAX4 JSR RORAX1   SHIFT 4 BITS BY SHIFTING
1960        JSR RORAX1        1 BIT 4 TIMES
1970        JSR RORAX1
1980 *---------------------------------
1990 *   SIMULATE 8088 "ROR AX,1"
2000 *---------------------------------
2010 RORAX1 LDA AH       8088 "ROR" SHIFTS END AROUND
2020        LSR          WITHOUT LEAVING A BIT IN CARRY
```

```
       ROR AL
       BCC .1      6502 DOES LEAVE A BIT IN CARRY,
       ORA #$80    SO LETS MERGE CARRY IN HERE.
.1     STA AH
       RTS
*--------------------------------
*   SIMULATE 8088 "EOR DX,AX"
*--------------------------------
EORAD  LDA AL
       EOR DL
       STA DL
       LDA AH
       EOR DH
       STA DH
       RTS
*--------------------------------
```

==============================================================================
DOCUMENT :AAL-8606:DOS3.3:S.Test6502Call.txt
==============================================================================

```
1000   .LIST OFF
1010  *SAVE S.TEST 6502 CALLING SEQUENCES
1020  *--------------------------------
1030  T1     LDA #MESSAGE.1
1040         PHA
1050         LDA /MESSAGE.1
1060         PHA
1070         JSR PRINT.IT
1080         LDA #MESSAGE.2
1090         PHA
1100         LDA /MESSAGE.2
1110         PHA
1120         JSR PRINT.IT
1130         RTS
1140  *--------------------------------
1150  MESSAGE.1
1160         .HS 8D
1170         .AS -/MESSAGE ONE/
1180         .HS 8D.00
1190  MESSAGE.2
1200         .HS 8D
1210         .AS -/MESSAGE TWO/
1220         .HS 8D.00
1230  *--------------------------------
1240         .LIST ON
1250  RETURN.SAVE  .EQ $00,01
1260  PNTR         .EQ $02,03
1270  PRINT.IT
1280         PLA          POP RETURN ADDRESS
1290         STA RETURN.SAVE+1
1300         PLA
1310         STA RETURN.SAVE
1320         PLA          POP MESSAGE ADDRESS
1330         STA PNTR+1
1340         PLA
1350         STA PNTR
1360         LDY #0       STARTING INDEX
1370  .1     LDA (PNTR),Y  NEXT CHARACTER OF MESSAGE
1380         BEQ .2       ...TERMINATING $00
1390         JSR $FDED    PRINT THE CHAR
1400         INY
1410         BNE .1       ...ALWAYS
1420  .2     LDA RETURN.SAVE
1430         PHA          RELOAD RETURN ADDRESS
1440         LDA RETURN.SAVE+1
1450         PHA
1460         RTS          RETURN TO CALLER
1470     .LIST OFF
1480  *--------------------------------
```

```
1490  *      JSR PRINT.MSG
1500  *      text of message, terminating zero
1510  *--------------------------------
1520  T2
1530         JSR PRINT.MSG
1540         .HS 8D
1550         .AS -/MESSAGE AFTER JSR/
1560         .HS 8D.00
1570         JSR PRINT.MSG
1580         .HS 8D
1590         .AS -/ANOTHER MESSAGE/
1600         .HS 8D.00
1610         RTS
1620  *--------------------------------
1630         .LIST ON
1640  PRINT.MSG
1650         PLA          GET RETURN ADDRESS
1660         STA .1+1     LO-BYTE
1670         PLA
1680         STA .1+2     HI-BYTE
1690         LDY #1
1700  .1     LDA $9999,Y  ADDRESS FILLED IN
1710         BEQ .2       ...TERMINATING $00
1720         JSR $FDED    PRINT THE CHAR
1730         INY
1740         BNE .1       ...ALWAYS
1750  .2     TYA          ADJUST THE RETURN ADDRESS
1760         CLC          BY ADDING THE MESSAGE LENGTH
1770         ADC .1+1
1780         TAY          SAVE LO BYTE FOR A WHILE
1790         LDA #0       THE HIGH BYTE TOO
1800         ADC .1+2
1810         PHA
1820         TYA
1830         PHA
1840         RTS          RETURN TO CALLER
1850  *--------------------------------
```

```
===========================================================================
DOCUMENT :AAL-8606:DOS3.3:S.Test816Call.txt
===========================================================================

 1000  *SAVE S.TEST 65816 CALLING SEQUENCES
 1010  *---------------------------------
 1020         .OP 65816
 1030  *---------------------------------
 1040  *      PEA address of message text
 1050  *      JSR PRINT.IT
 1060  *---------------------------------
 1070  T1     PEA MESSAGE.1
 1080         JSR PRINT.IT
 1090         PEA MESSAGE.2
 1100         JSR PRINT.IT
 1110         RTS
 1120  *---------------------------------
 1130  MESSAGE.1
 1140         .HS 8D
 1150         .AS -/MESSAGE ONE/
 1160         .HS 8D.00
 1170  MESSAGE.2
 1180         .HS 8D
 1190         .AS -/MESSAGE TWO/
 1200         .HS 8D.00
 1210  *---------------------------------
 1220  PRINT.IT
 1230         LDY #0       STARTING INDEX
 1240  .1     LDA (3,S),Y  NEXT CHARACTER OF MESSAGE
 1250         BEQ .2       ...TERMINATING $00
 1260         JSR $FDED    PRINT THE CHAR
 1270         INY
 1280         BNE .1       ...ALWAYS
 1290  .2     PLA          MOVE RETURN ADDRESS
 1300         STA 2,S      OVER THE TOP OF THE
 1310         PLA          MESSAGE ADDRESS, PRUNING
 1320         STA 2,S      THE STACK
 1330         RTS
 1340  *---------------------------------
 1350  *      JSR PRINT.MSG
 1360  *      text of message, terminating zero
 1370  *---------------------------------
 1380  T2
 1390         JSR PRINT.MSG
 1400         .HS 8D
 1410         .AS -/MESSAGE AFTER JSR/
 1420         .HS 8D.00
 1430         JSR PRINT.MSG
 1440         .HS 8D
 1450         .AS -/ANOTHER MESSAGE/
 1460         .HS 8D.00
 1470         RTS
 1480  *---------------------------------
```

```
1490  PRINT.MSG
1500         LDY #1       POINT TO FIRST CHAR
1510  .1     LDA (1,S),Y  GET NEXT CHAR
1520         BEQ .2       ...TERMINATING $00
1530         JSR $FDED    PRINT THE CHAR
1540         INY
1550         BNE .1       ...ALWAYS
1560  .2     TYA          ADJUST THE RETURN ADDRESS
1570         CLC          BY ADDING THE MESSAGE LENGTH
1580         ADC 1,S
1590         STA 1,S
1600         LDA #0       THE HIGH BYTE TOO
1610         ADC 2,S
1620         STA 2,S
1630         RTS          RETURN TO CALLER
1640  *--------------------------------
```

==========================================================================
DOCUMENT :AAL-8606:ProDOS:BUTTERILL.DEMO.txt
==========================================================================

( DTC removed -- lots of garbage characters )

```
==============================================================================
DOCUMENT :AAL-8606:ProDOS:BUTTERILLS.DIV.txt
==============================================================================

 1000  *SAVE BUTTERILLS.DIV
 1010  *--------------------------------
 1020  * 16 BIT DIVIDE WITH REMAINDER
 1030  * DIVIDE B BY A
 1040  * LEAVES QUOTIENT IN B,
 1050  *       REMAINDER IN A
 1060  *--------------------------------
 1070  *   TIMING:  A=$0000 -- 39 cycles
 1080  *            B>$7FFF -- 71 or 74 cycles
 1090  *            A=B     -- 82 cycles
 1100  *       A=1,B=$FFFF -- 676 cycles
 1110  *--------------------------------
 1120  A      .EQ 0,1      DIVISOR, REMAINDER
 1130  B      .EQ 2,3      DIVIDEND, QUOTIENT
 1140  *--------------------------------
 1150         .OP 65802
 1160  DIV16
 1170         CLC          ENTER FROM 6502
 1180         XCE          NATIVE MODE
 1190         REP #$20     A-REG 16 BITS
 1200         LDX #0       START SCALE CNTR
 1210         LDA A        GET DIVISOR
 1220         BEQ .90      ...ZERO DIVISOR
 1230         BMI .30      ...DIVISOR > $7FFF
 1240  *---SCALE DIVISOR----------------
 1250  .10    CMP B        ALIGN A TO LEFT
 1260         BCS .20      UNTIL > B
 1270         INX             OR BIT 15 SET
 1280         ASL          & COUNT IN X
 1290         BPL .10
 1300  .20    STA A        SCALED DIVIDEND
 1310  *---START SUBTRACTING------------
 1320  .30    LDA B        GET DIVIDEND
 1330         STZ B        CLEAR QUOTIENT
 1340  .40    CMP A        REPEATED CONDITIONAL
 1350         BCC .50       SUBTRACTION.
 1360         SBC A
 1370  .50    ROL B        ROL IN 1 IF SUBT.
 1380         LSR A              0 IF NO SUBT.
 1390         DEX
 1400         BPL .40
 1410         STA A        REMAINDER
 1420  *---RETURN TO CALLER-------------
 1430  .60    SEC          EXIT TO 6502
 1440         XCE
 1450         RTS
 1460  *---FOR X/0, GIVE 0,0 ANSWER-----
 1470  .90    STA B        DIVISION BY ZERO
 1480         BRA .60
```

```
1490  *--------------------------------
```

==================================================================
DOCUMENT :AAL-8606:ProDOS:BUTTERILLS.MUL.txt
==================================================================

```
1000  *SAVE BUTTERILLS.MUL
1010  *--------------------------------
1020  * 16 BIT MULTIPLY FOR 65802
1030  * MULTIPLIES A BY B
1040  * LEAVES ANSWER IN A & B
1050  *--------------------------------
1060  A      .EQ 0,1      MULTIPLIER, PRODUCT-LO
1070  B      .EQ 2,3      MULTIPLICAND, PRODUCT-HI
1080  *--------------------------------
1090  *   TIMING:  B=$0000 -- 27 CYCLES
1100  *            A=$0000 -- 335 CYCLES
1110  *            A=$FFFF -- 383 CYCLES
1120  *   (INCLUDING JSR AND RTS)
1130  *--------------------------------
1140         .OP 65802
1150  MULT16
1160         CLC          ENTER FROM 6502
1170         XCE
1180         REP #$20
1190         LDA B        IF B ZERO,
1200         BEQ .90      THEN BY-PASS
1210         DEC B
1220         LDA ##0000
1230         LDX #16      FOR 16 BITS
1240         CLC          FOR 17'TH CYCLE
1250  .10    ROR          ROLL OUT PRODUCT BIT
1260         ROR A        ROLL IN 'PLIER BIT
1270         BCC .20
1280         ADC B
1290  .20    DEX
1300         BPL .10      CYCLES 17 TIMES
1310         STA B
1320  .30    SEC          EXIT TO 6502
1330         XCE
1340         RTS
1350  .90    STA A        PROCEDURE FOR B=0
1360         BRA .30
1370  *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8606:ProDOS:DIV16.DEMO.txt
==========================================================================

 1000  * DIV16 DEMO
 1010  *SAVE DIV16.DEMO
 1020  *--------------------------------
 1030  * DEMO OF BRUN'ING A ML PROG
 1040  * USING DIV16
 1050  *
 1060  * DOS IS DISCONNECTED
 1070  * TO ALLOW I/O WITHOUT
 1080  * DISRUPTING PROPER RETURN
 1090  *--------------------------------
 1100          .OP 65802
 1110          .OR $6A00
 1120  *--------------------------------
 1130  COUT1   .EQ $FDF0    SCREEN OUTPUT
 1140  KEYIN   .EQ $FD1B    KEYBOARD INPUT
 1150  *--------------------------------
 1160  AL      .EQ 0
 1170  AH      .EQ 1
 1180  BL      .EQ 2
 1190  BH      .EQ 3
 1200  DFLG    .EQ 4        DELIMITER FLAG
 1210  GETLN1  .EQ $FD6F    INPUT LINE TO BUFFER
 1220  PRNTAX  .EQ $F941    OUTPUT A,X AS HEX
 1230  COUT    .EQ $FDED    OUTPUT A AS CHAR
 1240  CROUT   .EQ $FD8E    OUTPUT CR
 1250  *--------------------------------
 1260  DEMO
 1270          LDX #0       BEFORE ANY I/O,
 1280  .10     LDA $36,X     DISCONNECT DOS
 1290          PHA           BY PUSHING $36.39
 1300          LDA PTRS,X    ONTO STACK,
 1310          STA $36,X     & REPLACING
 1320          INX           WITH COUT1/KEYIN
 1330          CPX #4
 1340          BNE .10
 1350
 1360          JSR CROUT
 1370  .20     JSR GETLN1   INPUT LINE TO BUFFER
 1380          JSR HEXVALS  EXTRACT HEX VALUES
 1390          CPY #1       IF NULL LINE,
 1400          BEQ .80         THEN EXIT
 1410          JSR PROG     DIVIDE
 1420          LDA BH
 1430          LDX BL
 1440          JSR PRNTAX   DISP QUOTIENT
 1450          LDA #","
 1460          JSR COUT     DISP ','
 1470          LDA AH
 1480          LDX AL
```

```
        JSR PRNTAX   DISP REMAINDER
        JSR CROUT
        JMP .20
.80     LDX #3       RECONNECT DOS
.90     PLA           BY PULLING
        STA $36,X     $36.39 FROM
        DEX           THE STACK.
        BPL .90
        RTS
*--------------------------------
* REPLACEMENT I/O POINTERS
*--------------------------------
PTRS    .DA COUT1,KEYIN
*--------------------------------
* READ TWO HEX 16-BIT WORDS
* FROM INPUT BUFFER. (AFTER WOZ)
*--------------------------------
BUFF    .EQ $200
*--------------------------------
HEXVALS
        LDY #0       CLEAR BUFFER INDEX
        STY DFLG     CLEAR DELIMITER FLAG
.10     LDA #0       CLEAR A
        STA AL
        STA AH
.20     LDA BUFF,Y   GET CHAR FROM BUFFER
        INY
        CMP #$8D     = CR ?
        BNE .30
        RTS
.30     EOR #$B0     CONVERT ASCII TO HEX
        CMP #$0A
        BCC .40      IF 0-9
        ADC #$88
        CMP #$FA
        BCS .40      IF A-F
        LDA DFLG     ELSE ASSUME
        BNE .10       CHAR IS
        LDA AL        A DELIMITER
        STA BL       MOVE A TO B
        LDA AH        IF NOT REPEATED
        STA BH        DELIMITER
        DEC DFLG     SET DELIMITER FLAG
        JMP .10
.40     ASL          SHIFT NIBBLE
        ASL           TO LEFT HAND
        ASL           SIDE.
        ASL
        LDX #4       & ROL INTO MEMORY
.50     ASL
```

```
2030        ROL AL
2040        ROL AH
2050        DEX
2060        BNE .50
2070        STX DFLG     CLEAR DELIMITER FLAG
2080        JMP .20
2090 *--------------------------------
2100 * SUBROUTINE
2110 *--------------------------------
2120 PROG    .IN BUTTERILL'S DIVIDE
```

```
=============================================================================
DOCUMENT :AAL-8606:ProDOS:MLI.ERROR.PLUS.txt
=============================================================================

 1000  *SAVE MLI.ERROR.PLUS
 1010  *--------------------------------
 1020  CMDADR .EQ $BF9C
 1030  *--------------------------------
 1040  PRBYTE .EQ $FDDA
 1050  COUT   .EQ $FDED
 1060  *--------------------------------
 1070  MLI.ERROR.PLUS
 1080         STA ERRCOD   SAVE ERROR NUMBER
 1090         LDY CMDADR+1
 1100         LDA CMDADR   SUBTRACT 6 FROM ADDRESS
 1110         SEC
 1120         SBC #6
 1130         STA CALADR+1      CALL ADDR LO
 1140         BCS .1
 1150         DEY
 1160  .1     STY CALADR+2      CALL ADDR HI
 1170  *--------------------------------
 1180         LDY #2
 1190         LDX #3       COPY OPCODE & PARMS ADDR
 1200  CALADR LDA $9999,X       (ADDRESS FILLED IN)
 1210         INX
 1220         STA PARMADR.H,Y
 1230         DEY
 1240         BPL CALADR   ...UNTIL Y=-1
 1250  *--------------------------------
 1260         BMI .2       ...ALWAYS
 1270  .1     JSR COUT
 1280  .2     INY
 1290         LDA QERR,Y
 1300         BMI .1       ...ASCII CHAR
 1310         BNE .3       ...DATA BYTE
 1320         RTS          ...END
 1330  .3     TAX          USE AS INDEX
 1340         LDA MLI.ERROR.PLUS,X
 1350         JSR PRBYTE
 1360         JMP .2       NEXT CHAR
 1370  *--------------------------------
 1380  ERRCOD     .BS 1
 1390  PARMADR.H  .BS 1
 1400  PARMADR.L  .BS 1
 1410  OPCODE     .BS 1
 1420  *--------------------------------
 1430  QERR   .HS 8D
 1440         .AS -/MLI ERROR $/
 1450         .DA #ERRCOD-MLI.ERROR.PLUS
 1460         .AS -/ AT $/
 1470         .DA #CALADR-MLI.ERROR.PLUS+2
 1480         .DA #CALADR-MLI.ERROR.PLUS+1
```

```
1490        .AS -/ (/
1500        .DA #OPCODE-MLI.ERROR.PLUS
1510        .AS -/./
1520        .DA #PARMADR.H-MLI.ERROR.PLUS
1530        .DA #PARMADR.L-MLI.ERROR.PLUS
1540        .AS -/)/
1550        .HS 8D.00
1560 *--------------------------------
1570        .LIST OFF
```

```
==========================================================================
DOCUMENT :AAL-8606:ProDOS:MLI.ERROR.txt
==========================================================================
```

```
 1000  *SAVE MLI.ERROR
 1010  *--------------------------------
 1020  CMDADR .EQ $BF9C
 1030  *--------------------------------
 1040  PRNTAX .EQ $F941
 1050  CROUT  .EQ $FD8E
 1060  PRBYTE .EQ $FDDA
 1070  COUT   .EQ $FDED
 1080  *--------------------------------
 1090  MLI.ERROR
 1100         PHA          SAVE ERROR CODE
 1110         LDY #QERR
 1120         JSR PRMSG
 1130         PLA
 1140         JSR PRBYTE
 1150         LDY #QAT
 1160         JSR PRMSG
 1170         LDA CMDADR+1
 1180         LDX CMDADR
 1190         JSR PRNTAX
 1200         JMP CROUT
 1210  *--------------------------------
 1220  MSG1   JSR COUT
 1230         INY
 1240  PRMSG  LDA MSGS,Y
 1250         BNE MSG1
 1260         RTS
 1270  *--------------------------------
 1280  MSGS
 1290  QERR   .EQ *-MSGS
 1300         .HS 8D
 1310         .AS -/MLI ERROR $/
 1320         .HS 00
 1330  QAT    .EQ *-MSGS
 1340         .AS -/ AT $/
 1350         .HS 00
 1360  *--------------------------------
 1370         .LIST OFF
```

```
==============================================================================
DOCUMENT :AAL-8606:ProDOS:MULT16.DEMO.txt
==============================================================================

 1000  * MULT16 DEMO
 1010  *SAVE MULT16.DEMO
 1020  *--------------------------------
 1030  * DEMO OF BRUN'ING A ML PROG
 1040  * USING MULT16
 1050  *
 1060  * DOS IS DISCONNECTED
 1070  * TO ALLOW I/O WITHOUT
 1080  * DISRUPTING PROPER RETURN
 1090  *--------------------------------
 1100          .OP 65802
 1110          .OR $6A00
 1120  *--------------------------------
 1130  COUT1   .EQ $FDF0    SCREEN OUTPUT
 1140  KEYIN   .EQ $FD1B    KEYBOARD INPUT
 1150  *--------------------------------
 1160  AL      .EQ 0
 1170  AH      .EQ 1
 1180  BL      .EQ 2
 1190  BH      .EQ 3
 1200  DFLG    .EQ 4        DELIMITER FLAG
 1210  GETLN1  .EQ $FD6F    INPUT LINE TO BUFFER
 1220  PRNTAX  .EQ $F941    OUTPUT A,X AS HEX
 1230  CROUT   .EQ $FD8E    OUTPUT CR
 1240  *--------------------------------
 1250  DEMO
 1260          LDX #0       BEFORE ANY I/O,
 1270  .10     LDA $36,X     DISCONNECT DOS
 1280          PHA           BY PUSHING $36.39
 1290          LDA PTRS,X    ONTO STACK,
 1300          STA $36,X     & REPLACING
 1310          INX           WITH COUT1/KEYIN
 1320          CPX #4
 1330          BNE .10
 1340
 1350          JSR CROUT
 1360  .20     JSR GETLN1   INPUT LINE TO BUFFER
 1370          JSR HEXVALS  EXTRACT HEX VALUES
 1380          CPY #1       IF NULL LINE,
 1390          BEQ .80         THEN EXIT
 1400          JSR PROG     MULTIPLY
 1410          LDA BH
 1420          LDX BL
 1430          JSR PRNTAX   DISP HI-16
 1440          LDA AH
 1450          LDX AL
 1460          JSR PRNTAX   DISP LO-16
 1470          JSR CROUT
 1480          JMP .20
```

```
1490
1500 .80     LDX #3       RECONNECT DOS
1510 .90     PLA           BY PULLING
1520         STA $36,X     $36.39 FROM
1530         DEX           THE STACK.
1540         BPL .90
1550         RTS
1560 *---------------------------------
1570 * REPLACEMENT I/O POINTERS
1580 *---------------------------------
1590 PTRS    .DA COUT1,KEYIN
1600
1610 *---------------------------------
1620 * READ TWO HEX 16-BIT WORDS
1630 * FROM INPUT BUFFER. (AFTER WOZ)
1640 *---------------------------------
1650 BUFF    .EQ $200
1660 *---------------------------------
1670 HEXVALS
1680         LDY #0       CLEAR BUFFER INDEX
1690         STY DFLG     CLEAR DELIMITER FLAG
1700 .10     LDA #0       CLEAR A
1710         STA AL
1720         STA AH
1730 .20     LDA BUFF,Y   GET CHAR FROM BUFFER
1740         INY
1750         CMP #$8D     = CR ?
1760         BNE .30
1770         RTS
1780
1790 .30     EOR #$B0     CONVERT ASCII TO HEX
1800         CMP #$0A
1810         BCC .40      IF 0-9
1820         ADC #$88
1830         CMP #$FA
1840         BCS .40      IF A-F
1850         LDA DFLG     ELSE ASSUME
1860         BNE .10       CHAR IS
1870         LDA AL        A DELIMITER.
1880         STA BL       MOVE A TO B
1890         LDA AH        IF NOT REPEATED
1900         STA BH        DELIMITER
1910         DEC DFLG     SET DELIMITER FLAG
1920         JMP .10
1930
1940 .40     ASL          SHIFT NIBBLE
1950         ASL           TO LEFT HAND
1960         ASL           SIDE.
1970         ASL
1980         LDX #4       & ROL INTO MEMORY
1990 .50     ASL
2000         ROL AL
2010         ROL AH
2020         DEX
```

```
2030          BNE .50
2040          STX DFLG     CLEAR DELIMITER FLAG
2050          JMP .20
2060  *--------------------------------
2070  * SUBROUTINE
2080  *--------------------------------
2090  PROG    .IN BUTTERILL'S MULTIPLY
```

```
=========================================================================
DOCUMENT :AAL-8606:ProDOS:ROM.CRC.CALC.txt
=========================================================================

 1000  *SAVE ROM.CRC.CALC
 1010  *-------------------------------
 1020  LOCN   .EQ $00,01   ENTER DATA LOCN (L/H)
 1030  SIZE   .EQ $02,03   ENTER ROM SIZE (L/H)
 1040  AL     .EQ $04      SIMULATED 8088 REGISTERS
 1050  AH     .EQ $05
 1060  BL     .EQ $06
 1070  BH     .EQ $07
 1080  CL     .EQ $08
 1090  CH     .EQ $09
 1100  DL     .EQ $0A
 1110  DH     .EQ $0B
 1120  PTR    .EQ $0C,0D   WORK POINTER
 1130  CTR    .EQ $0E,0F   BYTE COUNTER
 1140  *-------------------------------
 1150  PRNTAX .EQ $F941
 1160  *-------------------------------
 1170         .OR $300
 1180  *-------------------------------
 1190  START  LDA LOCN     SETUP POINTER
 1200         STA PTR           TO ROM IMAGE
 1210         LDA LOCN+1
 1220         STA PTR+1
 1230  *-------------------------------
 1240         SEC          GET BYTE COUNT - 2
 1250         LDA SIZE
 1260         SBC #2
 1270         STA CTR
 1280         LDA SIZE+1
 1290         SBC #0
 1300         STA CTR+1
 1310  *-------------------------------
 1320         LDY #$FF     START CRC AT $FFFF
 1330         STY DL
 1340         STY DH
 1350         INY          Y=0
 1360         STY AH       INIT AH REG
 1370  *-------------------------------
 1380  .1     LDA (PTR),Y  GET NEXT BYTE
 1390         JSR FOLD.BYTE.INTO.CRC
 1400         INC PTR      BUMP THE WORK POINTER
 1410         BNE .2
 1420         INC PTR+1
 1430  .2     LDA CTR      DECREMENT THE BYTE COUNT
 1440         BNE .3
 1450         DEC CTR+1
 1460  .3     DEC CTR
 1470         LDA CTR      TEST IF FINISHED
 1480         ORA CTR+1
```

```
       BNE .1       ...KEEP GOING
       LDX DL       DISPLAY THE RESULT
       LDA DH
       JMP PRNTAX
*--------------------------------
FOLD.BYTE.INTO.CRC
       EOR DH
       STA DH
       STA AL
       JSR ROLAX4   8088 "ROL AX,C"
       JSR EORAD    8088 "EOR DX,AX"
       JSR ROLAX1   8088 "ROL AX,1"
       LDA DH       SWAP BYTES IN REG-D
       LDX DL
       STX DH
       STA DL
       JSR EORAD    8088 "EOR DX,AX"
       JSR RORAX4   8088 "ROR AX,C"
       LDA AL
       AND #$E0
       STA AL
       JSR EORAD    8088 "EOR DX,AX"
       JSR RORAX1   8088 "ROR AX,1"
       LDA AL
       EOR DH
       STA DH
       RTS
*--------------------------------
*   SIMULATE 8088 "ROL AX,C"
*--------------------------------
ROLAX4 JSR ROLAX1   SHIFT 4 BITS BY SHIFTING
       JSR ROLAX1        1 BIT 4 TIMES
       JSR ROLAX1
*--------------------------------
*   SIMULATE 8088 "ROL AX,1"
*--------------------------------
ROLAX1 LDA AL       8088 "ROL" SHIFTS END AROUND
       ASL          WITHOUT LEAVING A BIT IN CARRY
       ROL AH
       BCC .1       6502 DOES LEAVE A BIT IN CARRY,
       ORA #$01     SO LETS MERGE CARRY IN HERE.
.1     STA AL
       RTS
*--------------------------------
*   SIMULATE 8088 "ROR AX,C"
*--------------------------------
RORAX4 JSR RORAX1   SHIFT 4 BITS BY SHIFTING
       JSR RORAX1        1 BIT 4 TIMES
       JSR RORAX1
*--------------------------------
*   SIMULATE 8088 "ROR AX,1"
*--------------------------------
RORAX1 LDA AH       8088 "ROR" SHIFTS END AROUND
       LSR          WITHOUT LEAVING A BIT IN CARRY
```

```
2030        ROR AL
2040        BCC .1       6502 DOES LEAVE A BIT IN CARRY,
2050        ORA #$80     SO LETS MERGE CARRY IN HERE.
2060 .1     STA AH
2070        RTS
2080 *--------------------------------
2090 *   SIMULATE 8088 "EOR DX,AX"
2100 *--------------------------------
2110 EORAD  LDA AL
2120        EOR DL
2130        STA DL
2140        LDA AH
2150        EOR DH
2160        STA DH
2170        RTS
2180 *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8606:ProDOS:S.02.CALL.SEQ.txt
==========================================================================

 1000   .LIST OFF
 1010  *SAVE S.02.CALL.SEQ
 1020  *--------------------------------
 1030  T1     LDA #MESSAGE.1
 1040         PHA
 1050         LDA /MESSAGE.1
 1060         PHA
 1070         JSR PRINT.IT
 1080         LDA #MESSAGE.2
 1090         PHA
 1100         LDA /MESSAGE.2
 1110         PHA
 1120         JSR PRINT.IT
 1130         RTS
 1140  *--------------------------------
 1150  MESSAGE.1
 1160         .HS 8D
 1170         .AS -/MESSAGE ONE/
 1180         .HS 8D.00
 1190  MESSAGE.2
 1200         .HS 8D
 1210         .AS -/MESSAGE TWO/
 1220         .HS 8D.00
 1230  *--------------------------------
 1240         .LIST ON
 1250  RETURN.SAVE  .EQ $00,01
 1260  PNTR         .EQ $02,03
 1270  PRINT.IT
 1280         PLA          POP RETURN ADDRESS
 1290         STA RETURN.SAVE+1
 1300         PLA
 1310         STA RETURN.SAVE
 1320         PLA          POP MESSAGE ADDRESS
 1330         STA PNTR+1
 1340         PLA
 1350         STA PNTR
 1360         LDY #0       STARTING INDEX
 1370  .1     LDA (PNTR),Y  NEXT CHARACTER OF MESSAGE
 1380         BEQ .2       ...TERMINATING $00
 1390         JSR $FDED    PRINT THE CHAR
 1400         INY
 1410         BNE .1       ...ALWAYS
 1420  .2     LDA RETURN.SAVE
 1430         PHA          RELOAD RETURN ADDRESS
 1440         LDA RETURN.SAVE+1
 1450         PHA
 1460         RTS          RETURN TO CALLER
 1470     .LIST OFF
 1480  *--------------------------------
```

```
1490  *      JSR PRINT.MSG
1500  *      text of message, terminating zero
1510  *--------------------------------
1520  T2
1530         JSR PRINT.MSG
1540         .HS 8D
1550         .AS -/MESSAGE AFTER JSR/
1560         .HS 8D.00
1570         JSR PRINT.MSG
1580         .HS 8D
1590         .AS -/ANOTHER MESSAGE/
1600         .HS 8D.00
1610         RTS
1620  *--------------------------------
1630         .LIST ON
1640  PRINT.MSG
1650         PLA          GET RETURN ADDRESS
1660         STA .1+1     LO-BYTE
1670         PLA
1680         STA .1+2     HI-BYTE
1690         LDY #1
1700  .1     LDA $9999,Y  ADDRESS FILLED IN
1710         BEQ .2       ...TERMINATING $00
1720         JSR $FDED    PRINT THE CHAR
1730         INY
1740         BNE .1       ...ALWAYS
1750  .2     TYA          ADJUST THE RETURN ADDRESS
1760         CLC          BY ADDING THE MESSAGE LENGTH
1770         ADC .1+1
1780         TAY          SAVE LO BYTE FOR A WHILE
1790         LDA #0       THE HIGH BYTE TOO
1800         ADC .1+2
1810         PHA
1820         TYA
1830         PHA
1840         RTS          RETURN TO CALLER
1850  *--------------------------------
```

```
==========================================================================
DOCUMENT :AAL-8606:ProDOS:S.816.CALL.SEQ.txt
==========================================================================

 1000  *SAVE S.816.CALL.SEQ
 1010  *--------------------------------
 1020         .OP 65816
 1030  *--------------------------------
 1040  *      PEA address of message text
 1050  *      JSR PRINT.IT
 1060  *--------------------------------
 1070  T1     PEA MESSAGE.1
 1080         JSR PRINT.IT
 1090         PEA MESSAGE.2
 1100         JSR PRINT.IT
 1110         RTS
 1120  *--------------------------------
 1130  MESSAGE.1
 1140         .HS 8D
 1150         .AS -/MESSAGE ONE/
 1160         .HS 8D.00
 1170  MESSAGE.2
 1180         .HS 8D
 1190         .AS -/MESSAGE TWO/
 1200         .HS 8D.00
 1210  *--------------------------------
 1220  PRINT.IT
 1230         LDY #0       STARTING INDEX
 1240  .1     LDA (3,S),Y  NEXT CHARACTER OF MESSAGE
 1250         BEQ .2       ...TERMINATING $00
 1260         JSR $FDED    PRINT THE CHAR
 1270         INY
 1280         BNE .1       ...ALWAYS
 1290  .2     PLA          MOVE RETURN ADDRESS
 1300         STA 2,S      OVER THE TOP OF THE
 1310         PLA          MESSAGE ADDRESS, PRUNING
 1320         STA 2,S      THE STACK
 1330         RTS
 1340  *--------------------------------
 1350  *      JSR PRINT.MSG
 1360  *      text of message, terminating zero
 1370  *--------------------------------
 1380  T2
 1390         JSR PRINT.MSG
 1400         .HS 8D
 1410         .AS -/MESSAGE AFTER JSR/
 1420         .HS 8D.00
 1430         JSR PRINT.MSG
 1440         .HS 8D
 1450         .AS -/ANOTHER MESSAGE/
 1460         .HS 8D.00
 1470         RTS
 1480  *--------------------------------
```

```
1490  PRINT.MSG
1500         LDY #1       POINT TO FIRST CHAR
1510  .1     LDA (1,S),Y  GET NEXT CHAR
1520         BEQ .2       ...TERMINATING $00
1530         JSR $FDED    PRINT THE CHAR
1540         INY
1550         BNE .1       ...ALWAYS
1560  .2     TYA          ADJUST THE RETURN ADDRESS
1570         CLC          BY ADDING THE MESSAGE LENGTH
1580         ADC 1,S
1590         STA 1,S
1600         LDA #0       THE HIGH BYTE TOO
1610         ADC 2,S
1620         STA 2,S
1630         RTS          RETURN TO CALLER
1640  *--------------------------------
```

# F I N I S