APPLE III

The Personal Computer

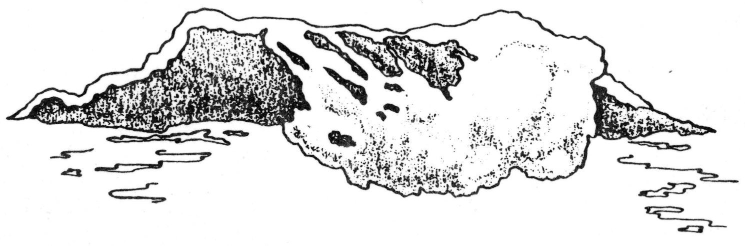
APPLE III

The Personal Computer

APPLE ///

SOFTWARE RESOURCE GUIDE

Catch



The ///rd Wave

APPLE /// SOFTWARE RESOURCE GUIDE

As the Apple /// has gained increasingly greater and greater respect and popularity in the business community, it is exciting to note some of the excellent software programs that have come onto the market. The following list of Apple /// software products has been compiled from a variety of sources and is listed by (1) quick-reference category, (2) descriptive category and (3) by vendor. This Apple /// Software Directory has been arranged into the following major divisional categories:

Business and Professional Software Categories

- Accounting
- Appointment/Calendar
- Data Base and Record Processing
- Education
- Financial Modeling and Analysis
- Graphics
- Information Services
- Languages
- Mail List Processing
- Real Estate
- Reference Sources
- Special Purpose
- Telecommunications
- Utilities
- Word and Text Processing

PRICES listed are suggested retail price.

Apple Computer can accept no liability as to the accuracy of information included in this resource guide.

QUICK-REFERENCE BY CATEGORY

ACCOUNTING

Accounting Series 8	(Peachtree)
Business Accounting	(Suburban Computer)
Bus.Bookkeeping Sys	(Dakin 5)
Great Plains	(Great Plains)
EASy	(Denver Software)
Insurance Agent	(Suburban Computer)
Legal Office	(Suburban Computer)
Medical Clinic	(Monument Computer Service)
Gusher	(High Technology)
State of the Art	(State of the Art)

APPOINTMENT/CALENDAR

Medical Clinic	(Monument Computer Service)
Vigil	(Quark)

DATA BASE AND RECORD PROCESSING

BibMod	(Blue Lakes)
DataBase ///	(Creative Software)
DataFax	(Link)
Data Manager ///	(MicroLab)
Data Reporter	(Synergistic Software)
FYI	(Living Videotext)
PFS	(Software Publishing)
PFS Report	(Software Publishing)
Quick File ///	(Apple)
ROSE	(Denver Software)
TDM	(Pascal Systems)
VersaForm	(Applied Software)
1st Class Mail	(Continental)

EDUCATION

Guidance Information System	(TSC)

FINANCIAL MODELING AND ANALYSIS

Spreadsheet and Analysis -

VisiCalc ///	(VisiCorp)
Advanced VisiCalc	(VisiCorp)
Consultant's Edge (aid)	(Ferox)
Desktop/Plan	(VisiCorp)
DocuCalc (aid)	(Micro Decision)
Micro DSS/A	(Addison-Wesley)
Micro-Foresight	(United Information Services)
Multiplan	(Microsoft)
Personal Money Manager	(Educational Computing Systems)
ProCalc (aid)	(Professional Calculations)
ProForma (aid)	(Management Control Concept)
Senior Analyst	(Apple)

Job Planning/Costing -

VisiSchedule	(VisiCorp)
P.A.C.E.	(High Technology)

Statistics -

InvenMod	(Blue Lakes)
StatMod	(Blue Lakes)

Scientific -

InvenMod	(Blue Lakes)
ProCalc	(Professional Calculations)
StatMod	(Blue Lakes)

GRAPHICS

Business Graphics	(Apple)
ChartMaster	(Decision Resources)
Desktop/Plan	(VisiCorp)
Fig Factory	(Sun Software)
Grafpak ///	(Smartware)
PFS Graph	(Software Publishing)
PlotMod	(Blue Lakes)
Screen Director	(Business & Professional)
SuperPlot	(Microware Associates)

INFORMATION SERVICES

Dow Jones News/Retrieval	(Dow Jones Information Services)
The Source	(Source Telecomputing)
DIALOG Information Services	(DIALOG Information Services)
CompuServe	(CompuServe Information Services)

LANGUAGES AND DEVELOPMENT SYSTEMS

Languages -

Fortran	(Microsoft)
Pascal	(Apple)
M Basic	(Microsoft)
COBOL ///	(Apple)
Business Basic	(Apple)
ALD System ///	(Foxware)
Transforth ///	(Insoft)
PL-1	(under CP/M)

Development System -

ALD, Assembly Language Development System	(Insoft)
Basic Extension	(Foxware)
COBOL	(Apple)
Easyform	(Business Software Services)
Jeppson Disassembler	(Softalk)
Pascal	(Apple)
Program Writer/Report	(Vital Information)

MAIL LIST PROCESSING

Mail List Manager	(Apple)
Mail List Interface	(Quark)
MailMod	(Blue Lakes)
1st Class Mail	(Continental Software)
PMS	(First Byte, Inc.)

REAL ESTATE

Restanal	(Management Control Concepts)
Property Management Sys	(Realty Software Co)
Property Manager	(Datamost)
Real Est Inv Prog	(Datamost)

REFERENCE SOURCES

Bus Handbook and Software Directory	(Vanloves)
1983 Apple][-/// Software Directory	(Vanloves)
Book of Apple Computer Software '83	(Addison-Wesley)
Software Location Service	(Sofsearch)

SPECIAL PURPOSE

Discourse	(Quark)	print spooler
VersaForm	(Applied Software Technology)	business forms
Visischedule	(VisiCorp)	project management
Critical Path Scheduling System	(Great Divide Software)	project management
Sales Order Entry	(Plain and Simple Software)	sales records
PARIS	(Parker Computer Systems)	accounting and insurance for physicians
MiniMax	(Healthwest)	financ. impact of health care
Gusher	(High Technology)	oil investments
PACE	(High Technology)	job costing
Market Technician	(Datamost)	stock market analysis
Personal Ledger	(Datamost)	tax preparation
OPAL ///	(Le Roux International)	IBM protocols emulator
Elect. Bulletin Board	(Alpine Computing)	
Universal Life Analysis		insurance

TELECOMMUNICATIONS

Access ///	(Apple)
Appeltel Disk	(Logica/BVT)
BiSync	(Viking)
Bisync	(IE Systems)
DataLink	(Link)
Dow Jones	(Apple)
Log On	(Ferox)
Micro-Courier ///	(Apple) 1st quarter 83
Micro/Terminal	(Microcom)
TypeFace	(Quark)
SWI Apple	(Core Technology)

UTILITIES

Pascal Utilities -

Easyform - screen generator	(Business Software Services)
Epson Screen Printer	(Alpine Computing)
Scat ///	(Expanding Space)
LinkIndex - B-tree indexer	(Link)
LinkSampler - tutorial	(Link)
LinkVideo - screen generator	(Link)
Pascal Library	(Apple)
QuestMod - screen generator	(Blue Lakes)
Record Processing Services	(Apple)
Script ///	(Apple)
TPG - source code generator	(Pascal Systems)
Pascal Interface	(Applied Software)

Other Utilities -

Blockaccess	(PowerTools)
Catalyst	(Quark)
Character Set Editor	(PowerTools)
Jeppson Disassembler	(Softalk)
Merge ///	(Forsythe Computers)
Sostran	(Alpine Computing)
Character Set Editor	(Alpine Computing)

WORD AND TEXT PROCESSING AND SPELLING CHECKERS

Word Processors -

Apple Writer	(Apple)
DocuWriter	(Creative Discount)
Type-Righter	(Imagineering)
Ink Well	(Foxware Systems)
Powertext	(Beaman-Porter)
Word ///	(Westico)
Word Juggler	(Quark)
Word Weaver ///	(Synergistic Software)
Write-On!	(Datamost)

Spelling Checker -

Apple Speller	(Apple)	1st Quarter 83
LexiCheck	(Quark)	
Legal Dictionary	(Quark)	

Accounting

BPI - General purpose accounting package is the first to combine accounting information with general purpose data base features using RPS. Available 1st quarter, 1983.

Business Bookkeeping System - a cash basis accounting system for small businesses featuring 3 ledgers; customer, vendor and employee, also general ledger and chart of accounts. Dakin 5 Corp, PO Box 21187, Denver, CO 80221 $395.

EASy - Comprehensive, integrated general ledger, receivables and payables system for small to medium-size business. Prints financial reports, invoices, checks and statements. Entirely menu driven. Supports multiple companies. Denver Software Company, 14100 E. Jewell Ave, Aurora, CO 80012. $725.

GL-Plus - General ledger accounting system. The "Plus" is a built-in accounts receivable and accounts payable capacity. Features include: 1) Simple Operation, 2) User specified charts of accounts, 3) Instant update of all accounts as each transaction is entered, 4) Large variety of reports, 5) Reports always as current as the last entry, 6) Built-in Accounts Receivable and Accounts Payable, 7) Can be used with a two disk drive system or with ProFile. Great Divide Software, 8060 W. Woodard Drive, Lakewood, CO 80227. 303-337-0383. $495.

Hardisk Accounting Series - Menu driven, double entry accounting system. Combines general ledger, accounts payable, accounts receivable, inventory, point of sale, sales order entry, purchase order entry, payroll, fixed asset management and mailing labels into an integrated accounting system. All modules are interactive and include complete audit trails. Designed for ProFile. Great Plains Software, 123 15th Street North, Fargo, ND 58102. 701-293-8483.

Job Control System versions 3-100 and 3-400 - Job costing system which maintains files for up to 1000 open jobs depending on the version. Reports available are: work orders, jobs by due date, job cost summaries, employee-hour reports, and post-audit trials. Comes with tutorial disks. Pascal. High Technology Software, PO Box 14665, Oklahoma City, OK 73113. Version 3-100, $450; version 3-400, $750.

Micro GL /// - Double entry general ledger and budgeting package. Uses ProFile to store up to 9999 transactions. Maintains up to 999 accounts. Provides variance reporting for month-to-data and year-to-date. Prints audit trails and balance sheets. Well documented and menu-driven. Micro Business Solutions, 622 Plymouth Ln, Foster City, CA 94404. $300.

MBSI - combines general purpose accounting with sales analysis and order processing. Requires Softcard ///. Micro Business Software.

Peachtree Software - General purpose accounting system available for CP/M. Includes accounts payable, accounts receivable, general ledger, inventory control and payroll. Requires SoftCard ///. Peachtree, 8th floor, 3445 Peachtree Rd. NE, Atlanta, GA 30326. $600 each.

State of The Art - Offers a comprehensive choice of stand alone or fully integrated accounting packages. The programs are easy to use and offer upward growth paths, i.e. Apple][data may be transferred to the Apple /// version, and a mapping program to install to a ProFile. Outstanding documentation and tutorials. State of The Art. 3183-A Airway Ave, Costa Mesa, CA 92626 800-422-2151 or 714-850-0111. Dealer Demo packages available. G/L module $595. Budget & financial reporting $495. A/R $595. A/P $595 Inventory control $595. Available December 1982: Sales invoicing $495, Professional time and billing $795, Word processing $395. Available 1st quarter 1983: Payroll $595, Personal Management $495.

Appointment/Calendar

Medical Clinic - Monument Computer Service, PO Box 603, Joshua Tree, CA 92252. 619-365-6668.

Vigil - Tracks more than 200 daily or weekly appointments. Includes adjustable alarm and advance warnings. Quark Engineering, 1433 Williams, Ste. 1102, Denver, CO 80218. 303-399 1096. $95.

Data Base and Records Processing

Data Base /// - has user definable layouts, automatic page numbering, calculations, totaling, averaging, and counting of data. Creative Software, 6081 Barbados Avenue, Cypress, CA 99063. 714-893-4695. $175.

DataFax - organizes information into folders, each of which contains as many individual pages of information as are required. Link Systems, 1640 19th Street, Santa Monica, CA 90404. 213-453-1851. $199.

The Data Machine - A data base management and reporting system designed for file maintenance and reporting from large data bases. Features maximum file size of 32,767 records, one key field for updating, up to twenty levels of possible subtotaling, batch loading to allow mass update from a sequential input. Pascal Systems, 830 Menlo Avenue, Menlo Park, CA 94025. 415-321-0761. $750.

Data Manager III -- Data Base Management System including report generator. Allows 16 MB files, with up to 200 fields per record, 32000 records per file. MicroLab, 2310 Skokie Valley Road, Highland Park, IL 60035 (312) 433-7550. $500

FYI - General purpose data base that combines graphics, word processing and numerical information into one data set. Designed to run on ProFile. Living Videotext, Inc., P.O. Box 3429, Saratoga, CA 95070 (408) 741-1104 $350.

PFS /// - Personal Filing System allows the user to set up a data base with variable field sizes. Includes report generator. Software Publishing, 2021 Landings Dr, Mountain View, CA 94043. $250.

RPS /// - Record Processing Services provides file management services for programs handling large quantities of data. The multikey file access method allows files to be readily interchanged amoung programs built on RPS. Data are not locked up in any one program. Pascal. Apple Computer. $150. Available 4th quarter 1982.

QuickFile /// - Generalized record processing system with memory resident files Incorporates both tabular and record presentation, complete sorting and selection, calculated columns and group totals, plus flexible report generation. Produces label and columnar reports. Output format is compatible with Apple Writer ///. Apple Computer. $100.

Education

Guidance Information System - Makes available a large body of occupational, schooling and financial information to help people make life decisions. By typing a simple set of instruction, people can immediately zero-in on information that fits their particular needs and interest. Users can compare the results of choices and decisions they have made with the results of other choices they have made. Over 4000 installations nationwide. Requires Apple /// with Profile. TSC/Houghton-Mifflin, Box 683, Hanover, NH 03755. 603-448 3838. Annual License. Quantity discounts available for multiple installations.

Financial Modeling and Analysis

Desktop Plan /// - combines financial analysis with model consolidation. Flexible report formats include graphics capability. VisiCorp, 2895 Zanker Rd, San Jose, CA 95134. $300.

VisiCalc /// - Electronic spreadsheet. VisiCorp through Apple Computer. $250.

Advanced VisiCalc - Extended version of the original VisiCalc includes file compatibility with word processing and graphics, variable column widths, protected fields and keystroke memory. VisiCorp. $400.

Micro DSS/F - Decision Support System / Finance - A very powerful financial modeling program. "Big 8" accounting firms are using this package to replace IFPS which runs only on mainframes. By unplugging timesharing IFPS, some accounting firms are able to cost justify the Apple /// and DSS/F in less than one month. Graphics is built-in. Excellent tool for either existing time sharing users or corporate planners. Ferox Micro Systems, 1701 North, Ft. Myer Drive, Suite 611, Arlington, Virginia 22209. 800-336-5496 or 703-841-0880. $1500. Distributed through Addison-Wesley. Quantity discounts are available.

DSS/A - Decision Support System / Analysis - Business Statistics and Sorting Utility Package. Ferox Micro Systems. $495. Quantity discounts are available.

Consultant's Edge - Program which writes user customized menus for DSS/F models. Ferox Micro Systems. $250. Quantity discounts are available.

Micro-Foresight - A comprehensive financial planning and forecasting system which can interchange models between Foresight on a mainframe and the Apple using integral communications software. Micro-Foresight commands are identical to mainframe Foresight. Features backward modeling, determine how input variables must change to meet defined objectives, and sensitivity analysis. Comprehensive report generation with all or partial content. Supports consolidations and graphical output. Variable sized columns, numeric editing, comments and footnotes. Requires CP/M. United Information Services, 6626 Convoy Court, San Diego, CA 92111. 714-560-7070.

Multiplan - Interactive electronic worksheet. Automatic passing of information from sheet to sheet. English prompts with extensive help features. Named cells and formulas. Ability to "open" windows in multiple worksheets. Requires CP/M. Microsoft, 10700 Northup Way, Bellevue, WA 98004.

PMM - Personal Money Manager - a full featured interactive budget manager. Educational Computing Systems, Oak Ridge, TN 37830. $60.

ProCalc - The program allows comparative calculations for different assumptions to be presented side by side. All formulas and data entries are immediately available for review. A built-in checking mode ensures the integrity of calculation procedures. ProCalc enables the VisiCalc /// user to improve and simplify technical applications. Professional Calculations, 4895 Futura Street, Eugene, OR 97404.

ProForma - Financial Planning Program. Management Control Concept, 124 St. Mary's St., Boston, MA 02215 (617) 536-8840

Senior Analyst /// - modeling package allows a financial model of virtually unlimited size due to its model consolidation capability. Apple Computer. $225.

Graphics

Business Graphics /// - powerful plotting program accepts data from any SOS compatible application (VisiCalc, Desktop Plan, Senior Analyst). Creates a variety of graphs including bar charts, pie charts and scatter diagrams. Includes statistical analysis. Displays graphs on a variety of printers and plotters. Business and Professional Software through Apple Computer. $250.

Chart-Master - Produces bar charts, line charts, scatter diagrams and pie charts, as well as text pages and signs on the screen, paper or transparencies. Data can be entered manually or automatically from VisiCalc or other programs. Charts can be edited, stored and retrieved. Options include producing up to nine charts per page, footnote and framing capabilities, left and right y-axes, a variety of hatching and line types, exploded pie segments, linear regression and curve-fitting. Chart-Master is interactive, menu-driven and requires very little training. Decision Resources, PO Box 309, Westport, CT 06880. 203-222-1974. Available 1st quarter 1983. $375.

Graph Power - Menu driven business graphics package. Functions in conjunction with DSS/F or independently. Supports the HP-7470 but no printers. Line plots, bar charts, pie charts and reports on paper or transparancies. Chart formats and data can be saved on disk and transmitted to other computers via Ferox's LogOn communications software package. Ferox Micro Systems, 1701 North Ft. Myer Drive, Suite 611, Arlington, Virginia 22209. 800-336 5496 or 703-841-0800. $295. Quantity discounts are available.

GraphEdit - Powerful graphics editor allows user to set up presentation quality graphic displays including charts, graphs and user defined images. Ferox Micro Systems. Available 4th quarter, 1982.

FontEdit - Allows user to create specialized type fonts including foreign languages, engineering symbols, etc. Ferox Micro Systems. Available 4th quarter, 1982.

Fig Factory - Graphics utility. Sun Software, P.O. Box 189, Tustin, CA 92680 (714) 559 1390 $75

Screen Director - allows user to combine Business Graphics screens into a set of slides for presentations and seminars. Slides may be displayed on the screen and controlled using a remote control switch. Business and Professional Software, 143 Binney Street, Cambridge, MA 02142. 617 491-3377. $150.

Superplot - Menu-driven plotting program that draws pie, line, or scatter charts, projects trends and will print or store charts for report or slide-show style presentations. Allows the use of DIF files. Microware Associates. $150.

Information Services

Dow Jones News/Retrieval - Dow Jones provides extensive news and business information. The data bases include: News, Current Quotes, Historical Quotes, Corporate Earnings Estimator, complete company reports including company profiles, detailed corporate financial information on over 3,200 companies, weekly economic survey, weekly economic update, Wall Street Journal highlights, Wall Street Week, sports, weather and movie reviews. Dow Jones Information Services, PO Box 300, Princeton, New Jersey 08540. 800-257 5114.

The Source - Offers electronic mail, mailgrams, travel agency, airline schedules, on-line shopping, wire service news, stock information, financial newsletters, consumer information, employment agency, an information search service for locating documents, as well as computing services. Source Telecomputing, McLean, VA.

DIALOG Information Services - The DIALOG Service provides access to more than 55 million references to journal and newspaper articles, conference papers and reports in over 130 data bases covering all areas of science, technology, business, medicine, social science, current affairs and humanities. DIALOG Information Services, 3460 Hillview Avenue, Palo Alto, CA 94304. 800-227-1927. A typical search cost ranges from $5 to $15.

Prestel - Services similar to The Source. Developed by the British Post Office. Logica, New York.

General Videotex - Offers electronic mail, a totally electronic bill-paying service and retrieval of encyclopedia information. Cambridge, MA.

CompuServe - Offers electronic mail, on-line shopping, wire service news, stock information, stock tracking, financial newsletters, consumer information, as well as computing services. CompuServe Information Services, Columbus, Ohio.

Languages

ALD System /// - An assembly language development system. Generates Apple][DOS 3.3 disk files - NOT SOS. Uses larger memory and speed to decrease development time for Apple][programs. Insoft, 259 Barnett Rd. #3, Medford, OR 97501. $125.

BASIC Extension - Disk, array and utility routines developed in assembler code and available as invokable modules to the BASIC programmer. Foxware Products, 165 West Mead Avenue, Salt Lake City, Utah 84010. 801-364-0394. $95.

Business BASIC - implementation of the popular Business Basic language found on many minicomputers. Includes easy to use graphics and program structures. Apple Computer. $125.

COBOL /// - GSA validated high intermediate COBOL with built in screen generator and interactive debugging. Apple Computer. Available 4th quarter, 1982.

MBASIC - Microsoft BASIC. Apple Computer. Included with Softcard ///.

Pascal /// - General Purpose High level language is the first to implement IEEE floating point standard. Apple Computer. $250.

Transforth /// - FORTH for the Apple ///. Includes six different text and graphic modes, turtle graphics commands, easily formatted text windows, four different character sets, and software control of the microprocessor speed. Insoft. $125.

Reference Sources

Vanloves Business Handbook and Software Directory for Microcomputers - The "Business" book of software and peripherals. $20.

Vanloves 1983 Apple][- /// Software Directory - Covers 36 user categories from agriculture to word procesing, handling more than 3000 programs. $25.

The Addison-Wesley Book of Apple Computer Software '83 - Guidebook to more than 500 programs for the Apple, including education, business, utilities, games and entertainment. Offers critical program analysis. $20.

Sofsearch - Software location service. Sofsearch will tailor each search of their 17,000 files according to function, computer type and industry. The cost of one search is $45, or $145 for a one year subscription. The subscription covers the cost of five searches during a year plus a quarterly updating of each search. Sofsearch, PO Box 5276 San Antonio, TX 78201. 512-340-8735.

Mail List Processing

Mail List Manager - Comprehensive mailing list processing system. Apple Computer. $150.

1st Class Mail - Keeps track of mailing lists or other general data bases. Allows data to merge with Apple Writer /// or Wordstar. Available October 1982. Continental Software, 11223 South Hindry Avenue, Los Angeles, CA 90045. 213-417 8031. $125.

Electronic Bulletin Board - uses a telephone modem with Apple /// to enable customers to have dealer access in a mail function. Alpine Computing, 851 N. Main, Logan, UT 84321 (801) 753-8410.

PMS (Prospect Management System) - Comprehensive lead and prospect management system to allow customer to automate and manage seminar attendance, direct mail list management, and telephone prospecting details. Interfaces with commercially available names and address thru National Business Lists. Ideal for any business that does extensive telephone prospecting, direct mail, and seminar selling (i.e. insurance agencies, brokers, and computer retail stores). Requires ProFile, and Hayes Modem (if telephone prospecting auto-dial feature is used). Available January 1983. First Byte, Inc., 3711 Long Beach Blvd., Suite 100, Long Beach, CA 90807 (213) 595-7006.

Universal Life Analysis - allows insurance brokers to analyze needs of client based on a variety of parameters. Requires Softcard ///.

Stat Mod - High level statistical analysis for the Apple /// includes graphics and general purpose analysis routines. Blue Lakes Software, 3240 University Ave, Madison, WI 53705.

VersaForm - Business form processing, Pascal based program. Features ease of use, unique capability of handling columnar data, works well with the ProFile or diskette, data entry checking and auto-filling fields, built-in calculator, calculated fields, and much more. Applied Software Technology, 14125 Capri Drive, Los Gatos, CA 95030, 408-370-2662. $495.

VisiSchedule - Critical Path Method project management system. Produces summaries, time charts and milestone reports. Easy to evaluate the impact of changes. Compatible with VisiCalc ///. VisiCorp, 2895 Zanker Rd, San Jose, CA 95134. $300.

Critical Path Scheduling System - System which aids in the scheduling of complex projects such as those in construction, aerospace, and many other industries. It is a powerful tool for analyzing and scheduling all the tasks required to complete any multi-level project on time and at the lowest cost. Its features include: Simple operation, large project capacity (over 2000 tasks), easy update of project status, large variety of reports, graphical project presentation, provision for manpower planning, can be used with one or two disk drive system or with ProFile. Many desired features such as automatic start and finish dates option, display of "float" time, flexible reports generation, sort and test option, etc. Great Divide Software, 8060 Woodard Drive, Lakewood, CO 80227. $495.

Sales Order Entry /// - Provides a complete, easy, and accurate method to enter and maintain sales order records. Manufactures can keep track of who ordered products, when it was ordered, if it was shipped, if the item was billed, how much it cost, and much more. Wholesalers can use it to help improve delivery, see which products are selling better, and help to calculate commission payments for sales people. Sales representatives can keep tab on who ordered what from where. Determination can be made as to what has been shipped, what commissions have been paid and what commissions are due. Written in plain English, without any assumptions of the user's knowledge of computers. Program is unprotected and help is available. Plain and Simple Software, 9003 Lexington NE, Albuquerque, NM 87112. 505-293-2448.

Physicians Accounts Receivable and Insurance System - PARIS is capable of performing all aspects of a physicians billing as well as accounts receivable. It provides summary reports of a patient's diagnosis and history. It utilizes standard approved "CPT" Medical Codes or the physicians own unique codes for diagnosis as well as billing information for services rendered. PARIS includes: patient information setup, patient charges, payments and adjustments, end of day processing, patient history reports, monthly statements, insurance claim form, ad hoc reporting, aging analysis, month end summary report, fiscal year-end processing, patient listings, and balance due report. Designed for ProFile. Parker Computer Systems, 4701 Fletcher, Fort Worth, Texas 76107. 817 429 6482. $2,495. Manuals $20.

MiniMax - Utilizes Apple /// and Visicalc which enables health care financial managers with no prior computer experience to immediately analyze potential dollar impact of Medicare regulations that maximizes Medicare reimbursements. HealthWest, 1 HealthWest Center, 20500 Nordhoff, Chatsworth, CA 91311 (213) 700-2000

Gusher - Oil and Gas accounting and investments package. High Technology Software, PO Box 14665, Oklahoma City, OK 73113.

P.A.C.E. - Prompt Accurate Cost Estimator. Job costing and control. High Technology.

Personal Ledger - Individualized bookkeeping with tax preparation as a goal. Datamost, 9748 Cozy Croft Ave., Chatsworth, CA 91311. $130

The Market Technician - Analyze and track the stock market with access to the Dow Jones index. Datamost. $130.

Property Management System - Realty Software Co., 1116 F 8th Street, Manhattan Beach, CA 90266, $375.00

Property Manager - track cost, taxes, collections, etc. for large commercial properties such as apartment buildings or shopping centers. Datamost. 9748 Cozy Croft Ave, Chatsworth, CA 91311. $295.

Real Estate Investment Program - analyze and project property values and investments. Datamost, 9748 Cozy Croft Ave., Chatsworth, CA 91311. $130

Restanal (Restaurant Analysis) - forecasts the expense of real property and calculates the benefit of ownership, including cash flow, appreciation, reduction of mortgage principal and tax effect. Particularly applicable for purchase decisions. Management Control Concept, 124 St. Mary's St, Boston, MA 02215. Available January 1983. $800.

Apple Access /// - Digital VT-100 and VT-52 terminal emulation. Includes recording to disk and file transmission. Apple Computer. $150.

Appeltel Disk - allows access to Prestel World Videotex Service, which includes information on world currency and commodities, import and export information and hotel and airline schedules and reservations. Logica/BVT, 666 Third Ave, New York, NY 10017. (212) 599-0828. $85.

DataLink - Comprehensive communications system. Features multiple terminal emulation, up and down loading of files, auto dial, auto logon, keystroke memory, and more. Link Systems, 1640 19th Street, Santa Monica, CA 90404. 213-453-1851. $100. Available 4th quarter 1982.

SWI Apple /// Communication System - allows the Apple /// to act like a Burroughs TD830 intelligent terminal. Communicates with the Burroughs computer using the standard "poll-select" and "point to point" protocols. Most of the Burroughs functions are provided including the status line and multiple pages. Core Technology Corp, 134 W. University, Suite 203, Rochester, Michigan 48063. 313-651-6421. $600.

Micro-Terminal - turns the Apple /// into an intelligent terminal, includes macros, file transfer, protocol selection, built-in editor, etc. Microcom, 1400A Providence Highway, Norwood, MA 02062. 617-762-9310. $100.

Log On - Asynchronous communications software to link with remote microcomputers or mainframes. Can communicate from Apple][to Apple ///. Can download from mainframe or commercial data base. Does not emulate a video terminal. Ferox Micro Systems, 1701 North Fort Myer Drive, Suite 611, Arlington, Virginia 22209. 800-336-5496 or 703-841-0800. $150. Quantity discounts are available. (Available 1st Quarter, 1983)

Bisync Emulation - IBM binary synchronous communications from the Apple ///. Emulates 3271 with 3277, 3274 with 3278, 3275 and 3276 displays; 3284 and 3286 printers; 2780, 3780, 2770, 3741 and 3781 batch remote job entry. Supports leased line and dial-up up to 9600 bps. Requires Softcard /// and synchronous modem. Viking Associates, 2726 S. Moline Court, Aurora, CO 80014. 303-337 2608. $1195 for interface board and software. $990 for software only.

Bisync Emulation - IE Systems lets a CP/M based Apple /// communicate with remote computers using bisynchronous protocols. When combined with the bisynchronous software products from Micro-Integration Inc., an Apple can emulate an RJE (remote job entry) terminal using either 3780, 3741, 2780 and 2770 protocols or a 3271/77, 3274/78, 3275, and 3276 bisynchronous device. Serial I/O board with bisynchronous software product $1195. Dealer discounts available. IE Systems, Inc., 98 Main Street , POB 359, Newmarket, NH 03857 (603) 659-5891

Utilities

Jeppson Disassembler -- Translates Apple /// machine language into readable form. Softalk Magazine, PO Box 60, North Hollywood, CA 91603

Epson Screen Printer - allows printing of the Apple /// text and graphics screen with various options. Alpine Computing, 851 N. Main, Logan, UT 84321. (801) 753-8410. $60. A driver with copying rights is also available for $130.

Discourse - a spooler designed for use with the Apple /// and ProFile. Printer output is spooled to disk freeing the computer for another task while simultaneously printing. Quark Engineering, 1433 Williams, Suite 1102, Denver, CO 80218. (303) 399-1096. $125.

Pascal Utility Library - General utility procedures for the Pascal programmer. Apple Computer. $75.

Data Saver - PFS Data Saver allows the saving of files from ProFile which are too large to fit on one diskette. Files can also be transferred from the multiple diskettes to ProFile. Software Publishing, 2021 Landings Dr, Mountain View, CA 94043. $30.

PSORT - Pascal text sort/merge utility. Shattock and Associates through Apple Computer.

Blockaccess -- Read, writes, edits data directly on disk one block at a time. Requires Pascal. Documentation on disk. $40 includes source code. PowerTools, 1206 Karen Ave, Austin, TX 78757.

Backup /// - Hard Disk utility. Marks backed up files with a dirty bit, only copying unbacked files. Originally available only with ProFile.

Character Set Editor - Uses keyboard with joystick to create new character sets. Requires Pascal. Documentation on disk. $40 includes source code. PowerTools.

Scat /// - Utility to provide catalog and file reports written in Business BASIC. Expanding Space. $29.95

TDM - Data Base Manager with report generator. Resident p-System interpreter makes it portable. Pascal Systems, 830 Menlo Ave. Ste 109, Menlo Park, CA 94025. $750.00

TPG - Pascal source code generator. Menu driven, 16 files open simultaneously, 17 screens. p.System code is transportable. Pascal Systems.

Turnkey Emulator - Automatically boots any DOS 3.3 compatible program. Eliminates the need for a two stage emulation boot by combining SOS and DOS boot information on the same disk. Available in Integer and Applesoft versions.

Catalyst - eliminates the need to swap disks when changing applications by permitting the user to boot from Profile. Switching from one application to another requires one keystroke. When combined with the Discourse spooler, Catalyst provides a performance improvement in printing and booting disks. Quark Engineering. $149.

Program Writer/Report -- Program Generator. Vital Information, 7899 Mastin Drive, Overland Park, KS 66204 $200.00

Sostran - utilizes Visicalc to make Apple][files interchangeable with Apple /// files. Alpine Computing.

Character Set Editor - a character set editor which creates characters for a 14 x 14 pixel. Alpine Computing.

/// Edit - Business BASIC development tool for editing, testing and debugging. Expanding Space. $29.95

/// Number - Renumber facility with many options available. Expanding Space. $29.95

Word and Text Processing

Apple Writer /// - Turns your system into a sophisticated word processor. Includes full text-editing capabilities such as find-and-replace, move text, change case, and underline. Also includes powerful Word Processing Language (WPL) as an added tool. Apple Computer. $225.

Word Juggler - Powerful word processing package combines general purpose word processing with spelling dictionary, mail list and typesetting features. Incorporates a generalized print spooler (Discourse). Quark Engineering, 1433 Williams, Suite 1102, Denver, CO 80218. Word Juggler $295. Mail List Manager Interface $35.Legal Dictionary $85.

TypeFace - An accessory for Word Juggler to interface with computerized typesetting equipment. Documents prepared using Word Juggler can be sent indirectly to a typesetting machine. Quark Engineering. $175.

Lexicheck - High performance spelling checker for use with Word Juggler. Eliminate virtually all typographical errors and common misspellings with a single keystroke. Scans up to 14000 words per minute. Built in dictionary can be expanded indefinitely to include technical words. Quark Engineering. $195.

Ink Well - Easy to use, "beginner's" level word processing system. Written in Business BASIC. Foxware Systems, 165 West Mead Ave., Salt Lake City, UT (801) 364-0394.

Word /// - A simple word processor written in Business BASIC. Allows a programmer to modify the source code to add customized functions. Word /// displays text as it is printed, including underlining, boldface, and inverse. System Decisions Group, 149 Rowayton Ave., Rowayton, CT 06853 Distributed by Westico 25 Van Zant Street, Norwalk, CT 06855 (203) 853-6880 $90.

WordStar /// - Popular word processing system for CP/M. Requires Softcard ///. Available 4th quarter 1982.

Write-On - Word processing program for the Apple ///. Datamost, 9748 Cozy Croft Ave., Chatsworth, CA 91311. (213) 709-1202. $130.

Script /// - Text formatter for files created by the Pascal Editor. Apple Computer. $125.

Merge /// - A file utilities package for the Apple /// that allows data files created on Apple Writer /// or Personal Filing System (PFS) /// to be merged with any Apple Writer /// document. The package includes on e Apple /// formatted diskette with Apple Writer /// files and word processing language (WPL) programs. A sample PFS template and data is included. Does for AW3 what templates do for VisiCalc to transport data files to and from PFS. Forsythe Computers, 7748 Forsythe Blvd, St. Louis, MO 63105. $55.

Epson Screen Printer	60	Alpine Computing 851 N. Main Logan, UT 84321 (801) 753-8410
VersaForm	495	Applied Software Technology 14124 Capri Drive Los Gatos, CA 95030 (408) 370-2662
Pascal Interface	245	
Apple Speller	N/A	Apple Computer, Inc. 20525 Mariani Ave. Cupertino, CA 95014 (408) 996-1010
Apple Writer ///	225	
Access ///	150	
Business Graphics	175	
Backup ///	N/A	
COBOL ///	495	
Dow Jones News/Quotes	130	
Pascal ///	250	
Pascal Library	75	
Quick File ///	100	
Rec Processing Servs	180	
Senior Analyst	225	
Script ///	125	
Mail List Manager	150	
Powertext	N/A	Beaman-Porter Inc. David Guest Pleasant Ridge Road Harrison, NY 10528. (914) 967-3504
StatMod	250	Blue Lakes Software 3240 University Ave. Madison, WI 53705 (608) 233-6502
QuestMod	250	
PlotMod	250	
MailMod	250	
BibMod	250	
InvenMod (Nov.)	395	
Screen Director	250	Business & Professional Software, Inc. 143 Binney St. Cambridge, MA 02142
PIK-Printer/Plotter Interface Kit		

Easyform	200	Business Software Services 17 Pease Street Wilbraham, MA 01095
1st Class Mail	150	Continental Software 11223 South Hindry Ave. Los Angeles, CA 90045 (213) 417-8031
SWI Apple ///	600	Core Technology Corporation 134 W. University, Suite 203 Rochester, MI 48063 (313) 651-6421
DocuWriter	250	Creative Discount Software Suite 2156 256 S. Robertson Beverley Hills, CA 90211 (800) 824-7888
Data Base ///	175	Creative Software 6081 Barbados Ave Cypress, CA 99063 (714) 893-4695
Business Bookkeeping System	395	Dakin 5 Corp. PO Box 21187 Denver, CO 80221
Visicalc Formatting Aids	45	Data Security Concepts PO Box 31044 Des Peres, MO 63131 (314) 965-5044
Personal Ledger	130	Datamost 9748 Cozy Croft Ave. Chatsworth, CA 91311 (213) 709 1202
The Market Technician	130	
Write On!	130	
Real Estate Inv.Progm.	130	
Property Manager	295	

Chartmaster	375	Decision Resources PO Box 309 Westport, CT 06880 (203) 222-1974
ROSE EASy	350 565	Denver Software Co. Suite 15 14100 E. Jewell Ave. Aurora, CO 80012
PMM - Personal Money Manager	60	Educational Computing Systems Oak Ridge, TN 37830
FontEdit Graph Edit Log On Graph Power Consultants Edge Dec.Sup.Sys/Finance Dec.Sup.Sys/Analysis	N/A N/A 150 295 250 1500 495	Ferox Micro Systems 1701 N. Ft. Meyer Dr. Suite 611 Arlington, VA 22209 (800) 336-5496 or (703) 841-0880
Merge ///	55	Forsythe Computers 7748 Forsyth Blvd. St. Louis, MO 63105 (314) 721-4300
Inkwell Basic Extension	N/A 95	Foxware Products 165 West Mead Ave. Salt Lake City, UT (801) 364-0394
G/L Plus Critical Path Scheduling System	495 495	Great Divide Software 8060 Woodard Dr Lakewood, CO 90227 (303) 337-0383
Great Plains General Ledger Accounts Rec/Pay Payroll Inventory	 N/A N/A N/A N/A	Great Plains Software 123 15th Street N. Fargo, ND 58102 (701) 293-8483

MiniMax	N/A	HealthWest 1 HealthWest Center 20500 Nordhoff Chatsworth, CA 91311 (213) 700-2000
P.A.C.E Gusher Job Control System 3-100 3-400	395 995 450 750	High Technology Software Products, Inc. P.O. Box 14665 2201 NE 63rd St. Oklahoma City, OK 73113 (405) 478-2105
Type-Righter	195	Imagineering, Inc. Suite #10 405 S. Farwell Eau Claire, WI 54601
ALD System /// Transforth	N/A	Insoft 259 Barnett Road, #3 Medford, OR 97501 (503) 244-4181
OPAL ///	600	Le Roux International, Inc. 3090 Acushnet Avenue New Bedford, MA 02745
LinkVideo LinkSampler LinkIndex DataFax	55 60 195 249	Link Systems 1640 19th St. Santa Monica, CA 90404 (213) 453-1851
FYI	350	Living Videotext, Inc PO Box 3429 Saratoga, CA 95070 (408) 741-1104
Appeltel Disk	85	Logica/BVT 666 Third Ave New York, NY 10017 (212) 599-0828

ProForma	1200	Management Control Concept
Restanal (Jan 1983)	800	124 St Mary's St.
		Boston, MA 02215
		(617) 536-8840
Micro G/L	300	Micro Business Solutions
		622 Plymouth Lane
		Foster City, CA 94404
		(415) 573-5556
Data Manager III	500	MicroLab
		2310 Skokie Valley Road
		Highland Park, Il 60035
		(312) 433-7550
Micro-Terminal	150	Microcom Inc.
Micro-Courier	(Oct)	1400-A Providence Highway
		Norwood, MA 02062
		(617) 762-9310.
Multiplan	275	MicroSoft
		10700 Northrup Way
		Bellevue, WA 98004
		(206) 828-8080
Superplot	150	Microware Associates
Medical Clinic	1495	Monument Computer Service
		Village Data Center
		P.O. Box 603
		Joshua Tree, CA 92252
PARIS	1250	Parker Computer Systems
Manuals	20	4701 Fletcher
		Fort Worth, TX 76107
		(817) 429-6482

TDM	750	Pascal Systems
TPG	895	830 Menlo Ave. Suite 109
		Menlo Park, CA 94025
		(415) 321-0761
Accounting Series 8	600	Peachtree Software
		8th Floor
		3445 Peachtree Rd. NE
		Atlanta, GA 30326
		(404) 266-0673
Sales Order Entry	500	Plain and Simple Software
		9003 Lexington NE
		Albuquerque, NM 87112
		(505) 293-2448
Character Set Editor	40	PowerTools
Blockaccess	40	1206 Karen Ave.
		Austin, TX 78757
ProCalc	130	Professional Calculations, Inc.
		4895 Futura St.
		Eugene, OR 97404
Word Juggler	295	Quark Engineering
LexiCheck	195	1433 Williams, Suite 1102
Legal Dictionary	85	Denver, CO 80218
Mail List Interface	35	(303) 399-1096
Discourse	125	
TypeFace	175	
Catalyst	149	
Grafpak ///	35	Smartware
		2281 Cobble Stone Court
		Dayton, OH 45431
		(513) 426-3579

Jeppson Disassembler		Softalk Magazine
Program	3	Box 60
Boot Disk	8	North Hollywood, CA 91603
		(213) 980-5074
PFS	175	Software Publishing Corporation
PFS Report	125	2021 Landings Drive
PFS Graph	125	Mountain View, CA 94043
		(415) 962-8910
VisiBridge/RTP	79	Solutions, Inc.
		PO Box 989
		Montpelier, VT 05602
		(802) 229-0368
Fig Factory	75	Sun Software
		PO Box 189
		Tustin, CA 92680
		(714) 559-1390
Word Weaver ///		Synergistic Software
		830 N. Riverside Dr. Suite 201
		Renton, WA 98055
		(206) 226-3216
Business Accounting	600	Suburban Computer Systems
Insurance Agent		SMAIL TCS915
Legal Office		
Bisync Emulation	990	Viking Associates
		2726 S. Moline Ct
		Aurora, CO 80014
		(303) 632-7004
VisiCalc	295	VisiCorp
Advanced VisiCalc	395	2895 Zanker Rd.
Desktop/Plan	300	San Jose, CA 95134
VisiSchedule	300	(408) 946 9000

Program Writer/Report	200	Vital Information 7899 Mastin Dr. Overland Park, KS 66204
Word ///	195	Westico 25 Van Zant St. Norwalk, CT 06855 (203) 853-6880

All of the preceding software is compatible with all Apple /// peripherals including the ProFile hard disk system.

No Apple II EMULATION and limited CP/M software has been included in this list.

PRICES listed are suggested retail price.

Apple Computer accepts no liability as to the accuracy of this information.

This list has been compiled from a variety of sources. Updates, additions and corrections are welcomed and may be sent to:

Apple Computer Inc.
Apple /// Software
20525 Mariani Ave. MS 18-E
Cupertino, California 950140

APPLE ///

PERIPHERALS/ACCESSORIES

Catch

The ///rd Wave

The following is a partial listing of several Apple /// accessories and peripheral devices. Additionally, with very few exceptions, all standard printers and plotters are compatible with the Apple /// computer.

Peripheral Cards

Burtronix Protocard /// - Uses proven circuitry to interface a parallel interface chip (6522) to the Apple /// hardware bus, allowing the user to pput custom circuits right on the board and connect them to the 6522. No knowledge of the Apple /// hardware bus is necessary. Room is provided on the boar for either a 26-pin ribbon connector (supplied) or a 25-pin D-type connector (also supplied) for external connections. A software driver on disk i provided to link the Protocard /// to Business BASIC, Pascal, or any other software that uses the SOS Drivers. Elcom Systems Peripherals, 429 Harrison Street, Suite A., Corona, CA 91729 (714) 734-8220. $195.

Time Card /// - Multi-function time utility for the Apple /// computer. Contains the year, month, hour, minute and second. A countdown timer with a range of one millisecond to 999 hours, 59 minutes, 59 seconds, 999 milliseconds. Selectable 12 or 24 hour time formats. Diagnostic error reporting. Fully compatible with the Apple SOS operating system. Vista Computer Company, 1317 E. Edinger, Santa Ana, CA 92705 (714) 953-0523. Price $195.

PKASO ID12-Color Interface - offers versatile ways to add intelligent text and graphics printing capabilities to the Apple ///. This includes full snapshot dump of any text or graphics screen image, 16 level gray scale printing, user created or software defined printing characters, and Super-Res graphics using the full dot resolution of the printer. Other features are included for use with color printers. Compatible with most popular languages such as BASIC, Pascal, and CP/M. An Apple /// package is available making full use of the Apple ///'s expanded graphics and changeable character fonts. Complete with cable, instructional diskette, and comprehensive manual. Interactive Structures, Inc., 146 Montgomery Avenue, Bala Cynwyd, PA 19004 (215) 667-1713. $165

The Grappler and Grappler Plus - Parallel printer and graphics interface. Orange Micro Inc., 3150 East La Palma Suite G, Anaheim, CA 92806 (714) 630-3620

Apple /// UPIC - Lets users attach a variety of parallel-mode printers to Apple /// Computer. Included with the card is a diskette containing an operating system driver, which lets you custom-configure the card to wark with the printer you are using. Can also be used with equipment other than printers, functioning as a general purpose parallel input/output interface. Apple Computer. $225.

Apple Softcard /// - Allows your Apple /// to take advantage of the wide variety of CP/M Z-80 based application software. Consisting of a plug-in Z-80 microprocessor circuit board, four manuals, two diskettes (including the CP/M operating system, MicroSoft BASIC and PIP utilities) the SoftCard /// system dramatically increases the amount of software available for the Apple ///. Apple Computer. $450.

Apple /// OEM Prototyping Card - Modular-printed circuit cards on which you can build custom interfaces for the Apple ///, accommodating most integrated circuits and components. Has built-in facilities for attaching a vaariety of edge connectors and switches to circuits. Apple Computer. $45.

Thunderclock Card - Clock/Calendar card for the Apple ///. Apple Computer. $149.

Hard Disk and Floppy Disk Systems

Hobbyist - Hard disk systems designed to work with Apple ///. Santa Clara Systems, Inc., 560 Division Street, Campbell, CA 95008.

Corvus Hard Disk and Networking Systems for the Apple /// - Also available in conjunction with other computer systems. Corvus.

Davong Systems, 1061 Terra Bella Avenue, Mountain View, CA 94043 (415) 965-7130. 5Mb $1995 10MB $2495 15Mb $2995

Disk /// - Floppy disk drive for the Apple ///. Up to three external Disk ///'s may be attached. Apple Computer. $435

Micro-Sci Apple /// Disk Drives - A3 Disk - 143K $399 A73 Disk Drive - 286 K $649 A143 Disk Drive - Double Sided 572K $799 Micro-Sci, Santa Ana, CA 92705. (714) 662-2801

Nestar Networking Systems - Networks available for Apple II and Apple /// Computers. Nestar.

AMS 5000 - Hard Disk System. 5, 10, or 20 megabyte.
Proguard - 8" Floppy Disk controller 2.2 megabytes. Sorrento Valley Associates, 11722 Sorrento Valley Road, San Diego, CA 92121 (714) 452-0101

Profile - 5 1/4", Winchester technology hard disk for the Apple /// styled to be placed between the Apple /// and Monitor. It increases the Apple ///'s on-line storage capacity to 5 million bytes, allowing you to safely store in one location information that would fill 35 floppy diskettes. Apple Computer. $2195.

Telecommunication Peripheral Devices

IE Systems lets a CP/M based Apple /// communicate with remote computers using bisynchronous protocols. When combined with the bisynchronous software products from Micro-Integration Inc., an Apple can emulate an RJE (remote job entry) terminal using either 3780, 3741, 2780 and 2770 protocols or a 3271/77, 3274/78, 3275, and 3276 bisynchronous device. Serial I/O board with bisynchronous software product $1195. Dealer discounts available. IE Systems, Inc., 98 Main Street, POB 359, Newmarket, NH 03857 (603) 659-5891

Bisync Emulation - IBM binary synchronous communications from the Apple ///. Emulates 3271 with 3277, 3274 with 3278, 3275 and 3276 displays; 3284 and 3286 printers; 2780, 3780, 2770, 3741 and 3781 batch remote job entry. Supports leased line and dial-up up to 9600 bps. Requires Softcard /// and synchronous modem. Viking Associates, 2726 South Moline Court, Aurora, CO 80014. 303-337-2608. $1195 for interface board and software. $990 for software only.

SWI Apple /// Communication System - allows the Apple /// to act like a Burroughs TD830 intelligent terminal. Communicates with the Burroughs computer using the standard "poll-select" and "point to point" protocols. Most of the Burroughs functions are provided including the status line and multiple pages. Core Technology Corp, 134 W. University, Suite 203, Rochester, Michigan 48063. 313-651-6421. $600.

Accessories and Miscellaneous Items

Apple /// Carrying Case - the Apple /// case features a foam padded cover and base, a removable locking cover that allows for cables to exit the case with the cover closed and locked, non-metallic hold down strap, rubber no-slip bumpers, sturdy ABS plastic end-cap construction. Fiberbilt, 601 west 26th Street, NY, NY 10001 (212) 675-5820. $100

Apple /// Vinyl Carrying Case - Vinyl carrying case for the Apple ///. Includes sturdy shoulder strap for ease in carrying. Apple Computer. $20.

Key III (facilitates a "detachable keyboard") - adds cable extension underneath the Apple /// , allowing the keyboard to be placed in any convenient location within a 3 foot radius. Accessory III 225 Rio Vista, Unit 54, Anaheim, CA 92806. 714-630-1583. Available from Dec. 1, 1982-Jan. 1, 1983 for $99, after Jan. 1 for $130

Using VisiCalc...a "hands on" Approach - an audio cassette training program containing two 120 minute cassette tapes, one data disk and instructions. McMullen & McMullen, Inc., P.O. Box 230, Jefferson Valley, NY 10535 (914) 245-2734. $59.95

Cursor /// - Powerful X-Y control system for the Apple /// computer, with two precision firing switches. Apple Computer. $59.95.

Silentype Printer - Quiet, compact thermal graphics printer that plugs directly into the back of the Apple ///. Apple Computer. $350.

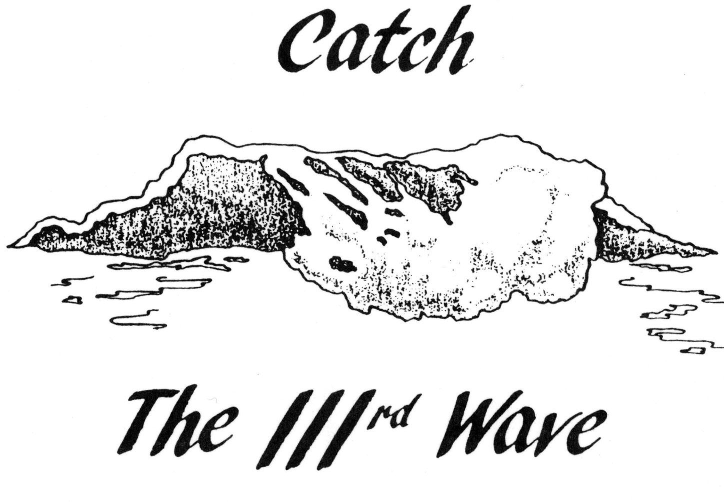
256K Upgrade - Plug-in board that allows expansion of 128K Apple /// to a 256K system without losing expansion slot space. The 192K of addressable memory made available by the upgrade is especially useful for programmers as well as for professionals interested in creating large financial models. Apple Computer. $945.

PRICES listed are suggested retail price.

Apple Computer can accept no liability as to the accuracy of any of the above information.
This list has been compiled from a variety of sources. Updates, additions and corrections are welcomed and may be sent to:

Apple Computer Inc.
Apple /// Software
20525 Mariani Ave. MS 18-E
Cupertino, California 95014

APPLE /// ARTICLES



The Apple III

Turning a lemon into a peach

by Steve Ditlea

When the Apple III was first introduced in 1980, Apple Computer management proclaimed it *the* personal business computer. Cynics, however, insisted that the only business served by this 6502A microprocessor-based cousin to the Apple II was Apple; the company seemed to be rushing its new product out the door to bolster the confidence of investors as they were offered the first public sale of Apple stock. In the following months, the Apple III was plagued with problems ranging from chips that popped out of their sockets at the slightest provocation to a lack of software. How could you have a business system without business-applications programs? Of course, the Apple II had been introduced with nothing more than a few home-brew cassette programs, but those were different times . . . .

So last fall, with much fanfare, the Apple III was relaunched, this time boasting more memory (including the first Winchester-type hard-disk memory offered by a major microcomputer manufacturer), more programs, and greater reliability. After announcing all this, Apple product manager Will Hood insisted that "with improved testing, the Apple III has become more reliable than the Apple II." It was with some skepticism that I listened to the claims made for this seemingly jinxed machine. I reserved judgment until I tried it myself.

Steve Ditlea is a regular contributor to Popular Computing.

Having tested the Apple III and some of its growing array of software, I can report that it is a unique, somewhat eccentric machine that could well become the business microcomputer workhorse Apple originally envisioned. It is not just an Apple II with more memory. Conceptually the Apple III is a different computer system, one that is more flexible and more adaptable to changing configurations and languages. Yet it suffers from what computer programmers call "creeping elegance," a phenomenon that occurs when a product becomes overdesigned and frills get in the way of basic strengths.

The standard factory-shipped Apple III is supplied with 128K bytes of user memory (a 256K-byte model will be reaching dealers by the time this appears), almost three times as much as that of the standard Apple II Plus, but alas, a lot of that memory is eaten up by the Apple III's operating system, SOS (Sophisticated Operating System; pronounced "sauce"). Coordinating applications-program instructions and the computer's hardware and I/O (input/

Reprinted by Permission

output) devices, this software has been designed to load from disk into user memory many of the functions that usually reside within a permanent memory chip in a computer with less internal storage.

For Apple II veterans who suffered through the emergence of Applesoft over integer BASIC as the most common high-level programming language or through the upgrading of Apple's disk operating system, the idea of adapting to changes by simply updating program disks, rather than installing new chips and boards, must seem attractive. On the other hand, each program disk must load in so much operations information that less than half of the Apple III's 128K bytes of user memory remain for an applications program and data.

Typically, a program disk contains the SOS "kernel" program, as well as separate "device drivers" with instructions for controlling the Apple III's keyboard, video display, in-board disk drive, external hard-disk drives, two printers, and a communications interface, along with the interpreter for a given high-level programming language. Total memory required for these system instruction files: more than 64K bytes. Though both SOS-formatted disks and the leaner (15K-byte) DOS 3.3-formatted Apple II disks can store a maximum of 140K bytes of information, the improved Apple III may actually provide only half the useful applications program and data capacity on a single disk. (The Apple III's microprocessor operates 40 percent faster than the Apple II's processor, but the extra operating instructions in SOS slow down the actual speed of execution of an applications program.)

With all this added bulk, what are the advantages of the Apple III's operating system? In addition to its flexibility in adapting to different system configurations, it has a built-in hierarchy of file designations that permit the efficient calling up of files from storage in various parts of the computer system. SOS also offers user- or device-generated interrupts of the processor's routines so new data can automatically be loaded or sent out to a monitor or printer—a function that must be written in by programmers on the Apple II. Apple software authors can also say good-bye to PEEKs and POKEs for examining memory locations; with SOS, programmers can make particular requests for allocating or relocating memory locations. Because all Apple III programming languages use SOS, data files from different applications programs can often be interchanged. SOS also permits simple reprogramming of the Apple III keyboard to include special-function keys or call up special letter fonts in English or other languages.

The flexibility of SOS and the ability to easily adapt to system expansion and upgrading should help the business user avoid expensive customization of applications programs. A number of off-the-shelf programs for the Apple III make good use of SOS file-handling abilities; others circumvent SOS for greater speed in executing instructions. With the emulation disk that comes with it, the Apple III can also run Apple II business software. Plugging an optional CP/M card into one of the Apple III's expansion slots allows you to use standard CP/M-based business programs, as well.

To dispel any doubts about the reliability of the Apple III, every factory-shipped machine comes with a Confidence Program disk.

At a Glance

Name: Apple III

Uses
Personal computing, business, professional firms

Manufacturer
Apple Computer Inc.
20525 Mariani Ave.
Cupertino, CA 95014
(408) 996-1010

Standard Features
6502A microprocessor; 74-key full-travel keyboard; video outputs for color and monochrome display; up to 80 characters per line, 24 lines, uppercase and lowercase, 128 programmable characters, 16 colors, 500 by 192 graphics dot resolution; built-in speaker and output for external speaker; 5¼-inch floppy-disk drive; 128K bytes of RAM; 4K bytes of preprogrammed ROM including self-test diagnostics; interfaces for RS-232C devices, two joystick controllers, Apple Silentype printer, up to four expansion boards internally; software includes disk operating system and BASIC interpreter

Base List Price
$3495

Typical System Price
Apple III with Silentype printer, monochrome display, $4165

Popular Options and Accessories
Hard disk, external floppy-disk drives, letter-quality printer

Reliable Hardware

To dispel any doubts about the reliability of the Apple III, every factory-shipped machine comes with a Confidence Program disk. Slip it in the built-in disk drive, turn on the computer's on/off switch, select Machine Status on the program menu, and the system runs through a self-test that checks off various system functions on screen. It then reports in a digitized voice over the Apple III's internal speaker: "I'm okay. Machine status normal."

Apple III hardware is actually more than okay—it is sturdy and well planned, if slightly ungainly, and it can hold its own against the IBM Personal Computer or any other microcomputer system aimed at the business user. The main unit is a one-piece central processor/keyboard combination with a horizontal 5¼-inch disk drive built into the right side of the unit. Sporting the Apple color scheme of beige, brown, and gray, the Apple III looks like a squat cash register. It is only slightly wider and higher than the Apple II, yet weighs almost three times as much because of metal in the chassis, shielding, and a built-in fan, all designed to add to the machine's reliability in a business environment.

The nondetachable keyboard will look unfamiliar to Apple II users: it includes a separate numeric keypad for easier data entry, as well as an honest-to-goodness tab key (useful in word-processing applications), up-arrow and down-arrow keys (for easier cursor control), and two extra control keys, "open" and "closed" (giving programmers more flexibility in designating "help" or other special command keys). Because this is a business machine, it also has uppercase and lowercase characters controlled by the Shift key. Text can be displayed on the monitor screen in any of three modes: 80-column, 24-line, monochromatic; 40-column, 24-line, 16-color foreground and background; or 40-column, 24-line, monochromatic. There are also three graphics modes: 280 by 192, 16-color; 140 by 192, 16-color; 560 by 192, monochromatic; plus Apple II emulation modes.

The Apple III comes with interfaces for external floppy-disk drives (up to three drives can be daisy-chained with the computer's built-in drive), two analog/digital ports, an RS-232C serial interface for communications, a 15-pin color video port, as well as standard black-and-white video and audio plugs. With its 128K-byte main memory upgraded to 256K bytes by a simple dealer-service procedure, the Apple III becomes the first personal computer to use 64K-byte memory chips to put a full 256K bytes of storage on a single (non-user-accessible) board. Fully configured, the Apple III boasts 64K bytes more memory than the comparable IBM Personal Computer and also has room for extra component boards in its four expansion slots.

To take full advantage of the Apple III, an interface card for Apple's Profile, a 5-megabyte Winchester hard-disk drive, should be in one of those slots. Only slightly bigger than a shoe box and about as noisy as an electric heater (occasional joyful beeps punctuate its communications with the Apple III), the Profile efficiently stores and recalls the equivalent of 35 floppy disks, or 1200 pages of text. As the Profile demonstration-program disk proves (with a little help from a high-resolution Dick Cavett), the hard disk has extremely fast data retrieval and graphics-storage capability. The Profile works like a dream—except for its lack of simple backup ability: preserving important records requires time-consuming downloading of information onto multiple disks, and Apple's answer to the need for backup on the Profile is to buy another $3500 Profile.

As has come to be expected from Apple, documentation for the Apple III is ample, well written, and graphically pleasing. The principal manuals are the *Owner's Guide*, with its overview of the Apple III, the *Standard Device Drivers Manual*, which details keyboard, disk, and accessory functions, and an individual *Profile Owner's Manual* (which erroneously shows the interface card going into expansion slot 1 instead of 4 in the Apple III, causing a bit of unnecessary aggravation in trying to get the system up and running with off-the-shelf software).

Software Worth Waiting For

In addition to Apple Computer Inc.'s release of a useful set of programs for Apple III business users, independent software developers have come up with enhanced versions of business progams proven successful on the Apple II. The Apple III's added internal memory, large-scale storage in hard disk, improved keyboard, and installed base of more than 25,000 users should make it an attractive computer for business-software programmers. Here's a sampling of some of the new software packages:

Financial Planning

Visicalc III (Apple Computer Inc., $250) may be one of the best reasons for buying an Apple III. Though it has the same "electronic spreadsheet" format of up to 63 columns and 254 rows as the original Visicalc (written for the Apple II by Software Arts and published by Personal Software, now Visicorp), this enhanced version has increased data-handling capabilities with a full 64K-byte user-memory space instead of the 19K bytes available on an Apple II Plus–hence, larger financial models can be constructed and processed. Other useful additions include built-in logarithmic, trigonometric, and logical functions that can be called up with simple three-letter commands. Also included is a second disk called the Visicalc Sampler, which provides useful preformatted Visicalc templates for personal or department budgeting, construction estimates, time value of money, and depreciation. Visicalc's well-written manual has been reset and spiral bound to conform to the design of other Apple Computer documentation.

Tax Preparation Models (Professional Software Technology Inc., 180 Franklin St., Cambridge, MA 02139) are the first in a series of applications templates that will add to the usefulness of Visicalc III. The Personal Tax Model ($99) includes every line of the federal income tax form 1040, as well as W-2 forms and Schedules C, D, and SE. Calculations weighing different tax strategies can be made in seconds; final figures can be printed directly onto a 1040 form. The Professional Tax Model ($149) also includes models for Corporate 1120 and Partnership 1065 federal tax forms.

The Apple III is a unique, somewhat eccentric machine that could well become the business microcomputer workhorse Apple originally envisioned.

Desktop/Plan-III (Visicorp Inc., 2895 Zanker Rd., San Jose, CA 95134, $300) is an expanded version of the best-selling menu-driven budget-and-analysis program on the Apple II. Taking advantage of the Apple III's greater memory capacity, Desktop/Plan-III has the capability of specifying up to 300 columns in a model instead of a maximum of 18 on the Apple II, and up to 8000 elements in a model compared to 2700 elements on the Apple II. This version of the program allows automatic consolidation of submodels into larger master models, faster program loading and data-file reading, more flexible calculation rules, and additional graphics ability.

Superplot (Microware Associates, 220 East 50th St., New York, NY 10022, $150) is an original business-graphics program written especially for the Apple III. With simple menu selections, it allows the user to enter data, including Visicalc DIF files, select a pie, line, scatter, or one of three different kinds of bar charts, edit any aspect of the chart, create projections of trends, and then print out or store a series of charts for reports or slide-show-style presentations.

Accounting and Job Costing

Micro GL III (Micro Business Solutions Inc., 622 Plymouth Lane, Foster City, CA 94404, $300) is another program written specifically for the Apple III. Using the Profile hard disk to store up to 9999 transactions (or 1200 transactions on a floppy disk) for accounting needs, Micro GL III offers small business a comprehensive general-ledger and budgeting package. Using double-entry bookkeeping methods, a user can post checks with automatic crediting of checking account, maintain up to 999 accounts, provide variance reporting for budgets on month-to-date and year-to-date bases, generate transaction history reports for audit and tax purposes, and chart accounts, balance sheets, and cash-flow reports. Fully menu-driven and well-documented, Micro GL III should distinguish itself even after other accounting programs for the Apple III reach the market.

Job Control System (High Technology Software, 2201 NE 63rd, Oklahoma City, OK, 73113 $750) is a fast, efficient, job-costing and control program written in Pascal. It can maintain files for 1000 open jobs on the Profile hard disk, which is more than enough for any small business to keep track of work in progress and improved productivity. Among the reports generated by Job Control System are work orders, jobs by due date, job-cost summaries, employee-hour reports, and post-audit trails. This program, complete with tutorial disks to practice data manipulation before entering the real thing, can put sophisticated job-management tools at the disposal of those businesses that need them.

Data Management

PFS: and PFS:REPORT (Software Publishing, 2021 Landings Dr., Mountain View, CA 94043, $145 and $125) are adaptations of top-selling electronic file-card software for the Apple II. The Apple III PFS software series offers larger forms for more data, with up to 32,000 forms per file on the Profile hard disk instead of the 1000-form maximum of Apple II floppy disks. This version of PFS:REPORT formats a maximum of 255 characters per line and up to 20 columns per report compared to the Apple II's 160 characters per line and 9 columns maximum.

As a solution to the problem of making backup copies of information on the hard disk, the PFS Profile Data Saver (Software Publishing, $30) allows files with more data than can fit on a single floppy disk to be downloaded onto several floppy disks; if data on the hard disk is lost, this software will reload files from the backup floppy disks. The Data Saver can also load preformatted information, like the PFS/Profile demonstration, which loads three disks containing the equivalent of the *Vanloves Apple Software Directory* (more than 1100 programs). Although there are no plans to make the *Vanloves* catalog publicly available in this fc-m, it does open the door for a new medium for publishing information like Census Bureau data, economic statistics, and commercial mailing lists.

Word Processing

Apple Writer III (Apple Computer Inc., $255), Paul Lutus's all-new version of his now-classic word processor for the Apple II, includes a wealth of features that make it an ideal word-processing choice for the Apple III. Apple Writer III retains Apple Writer's easy-to-edit single-text mode, but it has greater flexibility, including split-screen display for editing separate sections of a document, a user-defined glossary of simple commands to insert frequently employed words or phrases, five different character fonts, and an elementary programming language that enables full automation of personalized mailings or drafts of boilerplate contracts. Use of the Apple Writer III Utilities disk included in this package allows files to be transferred from Apple Writer II to Apple Writer III and vice versa. The program also accesses files from Mail List Manager, Apple's elementary address file sort-and-merge program (up to 6 lines per file, 960 labels per disk).

Word Juggler (Quark Engineering, 1433 Williams, Denver, CO 80218, $295) is the first word-processing program to make use of the full complement of keys on the Apple III. Two special plastic templates are placed on the keyboard to label the top line of numeric keys for formatting commands and the separate numeric keypad for editing functions. This word processor features single-keystroke commands, smooth cursor movement, well-documented error messages, and single-page call-up for printing a specific page of text. Limited to 1210-line documents in user memory (longer documents are composed of files drawn from disk memory during printing), Word Juggler is geared to office letter writing, including customizing of documents with merging of internally generated data files. Word Juggler is also the only word processor for the Apple III that offers its own spelling check.

Word III (Westico, 25 Van Zandt St., Norwalk, CT 06855, $195) is a simple word processor written in Business BASIC, which allows a programmer to modify the source code to add customized functions. Word III displays text as it will be printed, including underlining and boldface, and offers the option of viewing text in black characters on a white background. The user must gain a working knowledge of the Apple III's operating system before being able to put Word III to use, as the program relies heavily on SOS error routines.

Write-On III (Datamost, 19273 Kenya St., Northridge, CA 91326, $249.95) is a faster version of a reliable Apple II word processor, but it fails to make much use of the Apple III's unique features or expanded keyboard beyond the operating Shift key and up and down cursor controls. Only two pages have been added to the Apple II's manual to explain features of Write-On III, although certain Apple II features no longer apply. This program is valuable for the data-file routines that permit form-letter automation; new Write-On III documentation would help immeasurably.

Languages and Utilities

Apple Business BASIC (Apple Computer Inc., $125) is an extended version of Applesoft, with such features as 18-digit accuracy, Print Using and Image statements for easier formatting of reports and documents, and file commands like Catalog and Unlock that are part of the operating system on the Apple II. With more than 70K bytes of user-available workspace, Business BASIC allows lengthy programs to be stored in memory.

Apple III Pascal (Apple Computer Inc., $250) provides an efficient compiled language and development environment and allows you to write wider-ranging business programs (up to 64K bytes are available for your programs and data). Apple III Pascal also includes an editor as well as the only assembler software Apple currently offers for the III.

ALD System III (Insoft, 10175 Barbur Blvd., Portland, OR 97219, $125) is an assembly-language development system written by prolific software author Paul Lutus. Though ALD is designed to take full advantage of the Apple III's cursor-control keys and greater memory capacity, it is not designed to generate SOS disk files. Instead, using DOS 3.3, it is an effective development medium for stand-alone Apple III software or programs designed for the Apple II. Lutus now takes advantage of the greater memory capacity of the III with his assembler to speed up development of new programs for the Apple II.

Transforth III (Insoft, $125) is Paul Lutus's version of the versatile FORTH language, which can be employed for writing business-applications programs running in machine language. Transforth is the only edition of FORTH currently available on the Apple III, and it includes such useful features as six different text and graphics modes, powerful turtle-graphics commands, easily formatted text windows, four different character sets, software control of the speed of the Apple III's microprocessor, and comprehensive error-trapping routines.

Languages include Apple Business BASIC, Apple III Pascal, and a version of FORTH that has turtle-graphics commands.

Apple Access III (Apple Computer Inc., $150) is a communications program that turns the Apple III into an intelligent terminal, using its built-in RS-232C interface and a modem to exchange data with a variety of computers and timesharing systems. This program allows the Apple III to emulate Digital Equipment Corporation's popular VT-100 and VT-52 terminals, operate at transfer rates of 110, 300, 1200, 2400, 4800, or 9600 bits per second, and load entire data files from remote systems to process off-line in order to cut down on timesharing charges.

Summary

In areas such as keyboard, display, and memory capacity, the Apple III excels. Some new features, however, leave room for improvement. One glaring example is the lack of a convenient backup method for the hard-disk storage system. All in all, the Apple III is a powerful but far from revolutionary business machine. □

The Apple III and Its New Profile

An in-depth look at the "new" Apple III microcomputer and its Profile hard disk.

Robin Moore
Warner Hill Rd. RFD #5
Derry, NH 03038

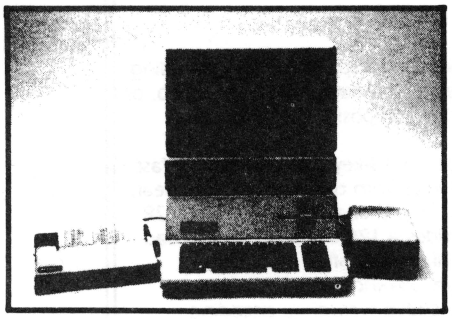


Photo 1: *A view of the Apple III, the Profile hard-disk drive, and the Monitor III showing a sample of Visicalc III on the screen.*

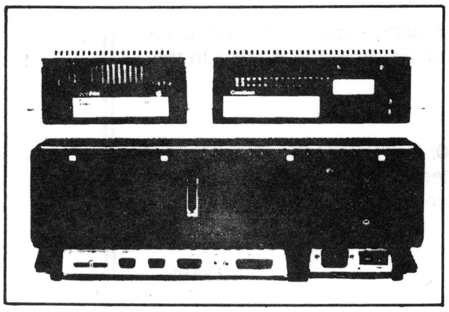


Photo 2: *A rear view of the Apple III and Profile showing the Silentype and game paddle ports A and B, along with the video, audio, RS-232C, and floppy-disk connectors. The peripheral card visible is the Profile interface card.*

In 1980 when the Apple III was first released, there were problems. Deliveries were delayed, and when the machines finally arrived, they often didn't work. The integrated circuits tended to wander out of their sockets. Little software except Visicalc was available, and the much-promoted real-time clock/calendar didn't work well. The Apple III was, on the whole, unreliable. It was a bad start.

Now, in 1982, the problems are gone. The sockets have been changed and the software bugs fixed. The Apple III has been rereleased with revised software, Pascal, and a brand-new peripheral—the Profile, a 5-megabyte hard-disk drive. The new Apple III is an impressive machine and certainly a contender for the title of Best Personal Computer in the less than $10,000 class.

System Overview

Let's take a closer look. The Apple III is a single unit that includes the central processing unit, keyboard, memory, floppy-disk drive, and video output (see photos 1 and 2). It has been designed to meet the needs of the professional or small-business user. Instead of offering an initial low-cost unit requiring a number of additions, Apple Computer Inc. has included the most common system expansions as standard in the Apple III. These include an enhanced keyboard, a 24-row by 80-column display, an integral disk drive, 128K bytes of memory, a programmable 128-character set, improved high-resolution graphics, and an Apple II emulation program (see the At a Glance box for additional features and details).

In addition, several peripherals are available for the Apple III. The most impressive of these is the Profile, Apple's new 5-megabyte hard-disk drive. (The Profile will be described in detail later in this article.) Other options from Apple Computer in-

About the Author
Robin Moore is manager of microprocessor development for A. B. Dick Co. and maintains a strong interest in FORTH, graphics, and computer music. He is also librarian for the Southern New Hampshire Apple Core.

At a Glance

Name
The Apple III Computer

Manufacturer
Apple Computer Inc.
20525 Mariani Ave.
Cupertino, CA 95014
(408) 996-1010

Components
System Unit

Size:	width 17.5 inches (44.45 cm), depth 18.2 inches (46.23 cm), height 4.8 inches (12.19 cm)
Weight:	26 pounds (11.8 kg)
Power Required:	107–132 volts AC, 60 Hz, 100 watts maximum
Processor:	6502B (2 MHz) with bank switching and enchanced indirect addressing, double stack and zero pages
Memory:	128K bytes of dynamic RAM (expandable to 256K bytes), 4K bytes of self-test and boot-loader ROM
Standard:	keyboard for text and data entry; programmable RS-232C serial communications/printer interface; power-up self-check and disk bootstrap; both color-graphics and black-and-white/gray-scale graphics video outputs; two game-paddle/joystick connectors; three audio generators—fixed beep, 1-bit programmable, and 6-bit A-D converter; one 140K-byte 5¼-inch floppy-disk drive
Video Display:	Three Text Modes 24 by 80, black and white, normal and inverse 24 by 40, black and white, normal and inverse 24 by 40, 16 color characters on 16 color backgrounds All text modes have software-definable 128-character sets Four Graphics Modes 280 by 192, 16-color foreground and background with limitations 280 by 192, black and white 140 by 192, 16 colors with no limitations 560 by 192, black and white
Video Outputs:	Both black-and-white/gray-scale and color-graphics outputs providing NTSC monochrome composite video, NTSC color composite video, or 4-bit coded RGB color with a separate composite synchronization signal
Keyboard:	74 keys for text and data entry; includes 13-key numeric pad for fast numeric entries, four cursor control keys with two-speed auto-repeat, three special-function keys, and text keys that allow entry of all 128 ASCII characters; SOS software provides a 128-character type-ahead keyboard buffer; all keys automatically repeat after ½ second
Disk Drives:	System supports up to four 140K-byte 5¼-inch floppy-disk drives using Apple-format 6/8 GCR (group-coded recording) encoding

Operating System
Apple III SOS 1.1 (Sophisticated Operating System); single task, interrupt-driven, configurable operating system with hierarchical file structure, multiple file protection levels, and device-independent byte-oriented I/O

Special Features
An Apple II emulation mode that allows use of almost all existing Apple II software; utilities that allow transfer of DOS text files, Visicalc files, and Pascal files from the Apple II to the Apple III

Software Available for the Apple III
Visicalc III $250; Applewriter III $225; Apple III Pascal $250; Business BASIC $125; Apple Access III (communications software) $150; Apple III Business Graphics $175; Pascal Utility Library $75; Script III $125; Mail List Manager $150; all from Apple Computer Inc.

Hardware Prices (Apple Computer Inc)

Apple III 128K-byte system	$3495
Apple III 256K-byte system	$4295
Additional disk drives (three maximum)	$495
Profile 5-megabyte Winchester hard disk-drive and interface card	$3,499
Universal parallel interface card	$225
Apple Monitor III (monochrome/green screen)	$320
Game controllers	$29.95

Apple III (list prices)	
128K-byte system unit with integral 140K-byte 5¼-inch floppy-disk drive, Apple SOS operating system software, both color-graphics and black-and-white/gray-scale video outputs, RS-232C serial interface, game control port, and Silentype printer interface	$3495
additional floppy-disk drive (three maximum)	$495
Apple Business BASIC software	$125
total	$4115
IBM Personal Computer (suggested retail prices)	
48K-byte system unit, disk-adapter card, one 160K-byte floppy-disk drive, DOS software, Disk BASIC	$2235
16K bytes of added memory and game adapter card	$145
additional floppy-disk drive (one maximum)	$570
serial RS-232C interface card	$150
additional 64K-byte memory card	$540
color-graphics video adapter card	$300
Microsoft extended BASIC software	$40
total	$3980

Table 1: *Price comparison of comparable versions of the Apple III and the IBM Personal Computer. Both systems include 128K bytes of memory, two floppy-disk drives, color-graphics video output, serial RS-232C interfaces for Qume (or equivalent) letter-quality printers, and game-paddle adapters. The system chosen is one that might be purchased by people who wish to combine business and personal applications. Note that in this configuration the IBM has used up all its expansion slots, while the Apple III still has all four of its slots left for further expansion.*

clude the Silentype thermal printer, additional floppy-disk drives, the monochrome green-screen Monitor III, a universal parallel I/O (input/output) interface card, and game controllers.

Many of the existing Apple II interface cards will work in an Apple III while in the Apple II emulation mode. However, use of Apple II cards in an Apple III will probably make it exceed FCC (Federal Communications Commission) radio-frequency radiation limits and may cause interference on nearby television sets or radios. In addition, Apple II cards are not compatible with Apple III software unless special *device-driver* routines are written, and Apple provides virtually no information on how to write them.

Apple Computer currently provides a variety of software packages for the Apple III in addition to Business BASIC and Apple Pascal. There are also various hardware and software products available for the Apple III from other vendors and the number of these will increase as the Apple III user community grows.

The only software built into the Apple III is a 4K-byte ROM (read-only memory) that holds power-up self-test and disk bootstrap routines. All other software is loaded from disk. Although this means that languages use up some of the available RAM (random-access read/write memory), it also allows easy software upgrades and fixes that would be more difficult if the software were permanently in ROM.

System Pricing

The approach to Apple III pricing is almost directly opposed to the pricing strategy used for the Apple II and the IBM Personal Computer. Because Apple chose to include a large number of standard features, the Apple III has a relatively high initial cost ($3495); however, it can expand to 256K bytes of memory, four floppy-disk drives, and a letter-quality Qume (or equivalent) printer without using *any* of the expansion slots. A fully usable system can be configured by adding just a video monitor and an inexpensive serial printer.

Table 1 shows a price comparison of the Apple III and the IBM Personal Computer. Both systems are configured with 128K bytes of memory, two floppy-disk drives, a serial RS-232C printer interface, color-graphics video outputs, and game controllers. The IBM system costs slightly less but uses all of its expansion slots, while the Apple III still has its four slots available for future growth.

The Apple III User

A look at the documentation and software supplied with the system will quickly reveal that the Apple III is targeted for professional and small-business users. Clear tutorials and example programs on disk demonstrate most system functions and features. There is even a two-disk program to lead you through the keyboard and display functions step by step.

The Apple III is not designed for the home hobbyist. Much of the technical information included with the Apple II is absent in the Apple III package. There is no discussion of bus structure, I/O addressing, memory usage, or screen-memory mapping. There are no listings published for any of the system software, either in the Apple III ROMs or on disk. Apple does not even tell you about the monitor program included in the ROMs (which is accessible by holding the Control and Open-Apple keys while pressing Reset).

All this technical information is unimportant to business users. They are more interested in *using* the Apple

Circle 214 on inquiry card. ⟶

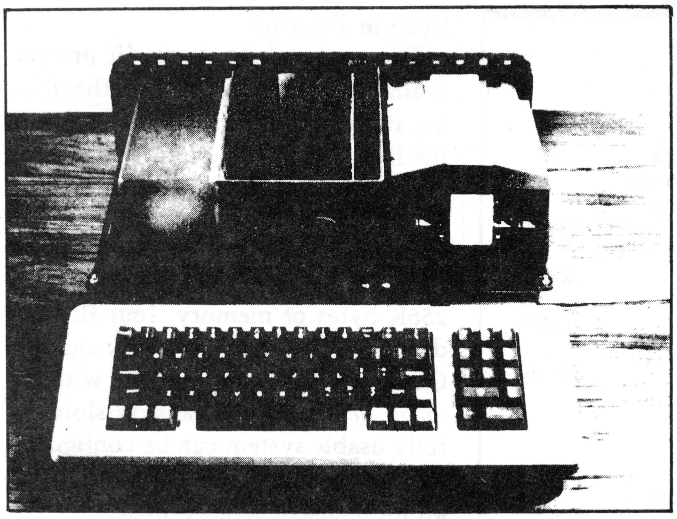


Photo 3: *The Apple III with its main cover removed. The power supply is housed in the enclosure visible to the left, I/O card slots are in the center, and the disk drive is on the right. The entire Apple III is built around a single thin-wall aluminum casting that provides both support and shielding.*

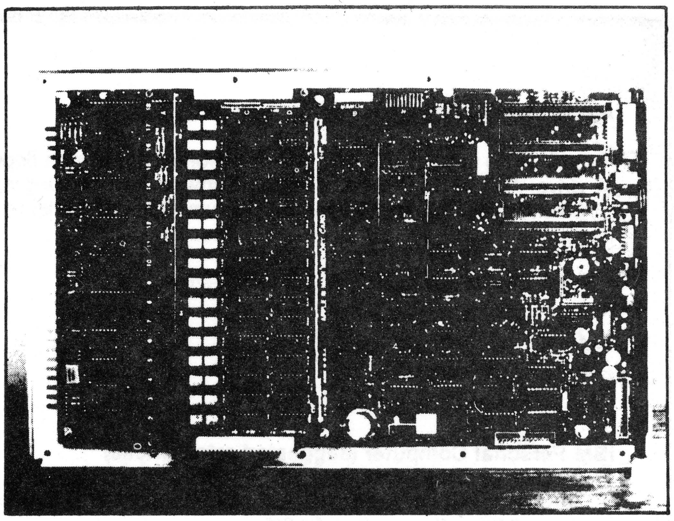


Photo 4: *The Apple III main PC board. The piggy-back-mounted board to the left of center is the removable main memory board. Using this board, the Apple II can be expanded to its full memory capacity without using up any of its I/O expansion slots.*

III than in dissecting it, and will, in most cases, use commerical software. The Apple III is admirably designed to serve their needs. For hobbyists there are better choices, namely, the Apple II.

Inside the Enclosure

The Apple III is a fine example of a quality product designed for high-volume production. The entire unit is built around a single thin-wall aluminum casting that provides support and shielding as well as heat dissipation so that no cooling fan is required. The expansion card guides are molded into the casting, and fully enclosed boxes are built in for both the main printed-circuit (PC) card and the switching power supply (see photo 3).

All of the circuitry, except memory, is on one main PC board (see photo 4). The system memory board mounts piggy-back style onto the main board and avoids taking an expansion slot. In fact, the Apple III can be expanded to its full 256K-byte memory capacity in the same fashion, leaving all slots free.

The Apple III central processing unit is based on a 6502B microprocessor with custom external circuitry that provides a number of enhancements to the normal 6502 instruction set. These enhancements include expanded addressing range, alternate stack and zero pages, and improved indirect addressing that is supported by a separate pointer page.

Although the technical information provided by Apple is somewhat vague, apparently the 6502B is run at 2 MHz during the video blanking intervals and at 1 MHz while the beam is writing information onto your monitor screen. This provides an average speed of about 1.4 MHz, but the screen can be turned off temporarily during program execution to allow the processor to run at its full 2-MHz speed, if desired.

The Apple III can be configured to 256K bytes without using a single expansion slot.

While a normal 6502B can address a maximum of 64K bytes of memory, the Apple III uses bank switching to expand this range to a theoretical maximum of 512K bytes.

Up to fifteen 32K-byte blocks of memory can be switched to occupy the range of addresses between 2000 and 9FFF hexadecimal. This switching is handled automatically by the operating system and is totally "transparent"; that is, the switching executes in the background without affecting any task you may be performing in the visible foreground. It should be noted that, to date, Apple Computer has not announced any Apple III memory expansion beyond 256K bytes. Perhaps this will be a future option.

The main PC board also includes the disk controller, serial interface, video generation circuitry, and the expansion card slots. The expansion-bus connections in the Apple III are essentially the same as those in the Apple II, although DMA (direct memory access) is handled somwhat differently. The Apple III *Owners Manual* provides no information about the expansion bus. Hopefully, this type of information will be available in the future. There are few competing systems that do not make this sort of information available to the public.

The Keyboard

Experienced typists should find the Apple III keyboard easy to use (see photo 5). Unlike the Apple II, this keyboard has a typewriter layout so that touch-typists should feel comfor-

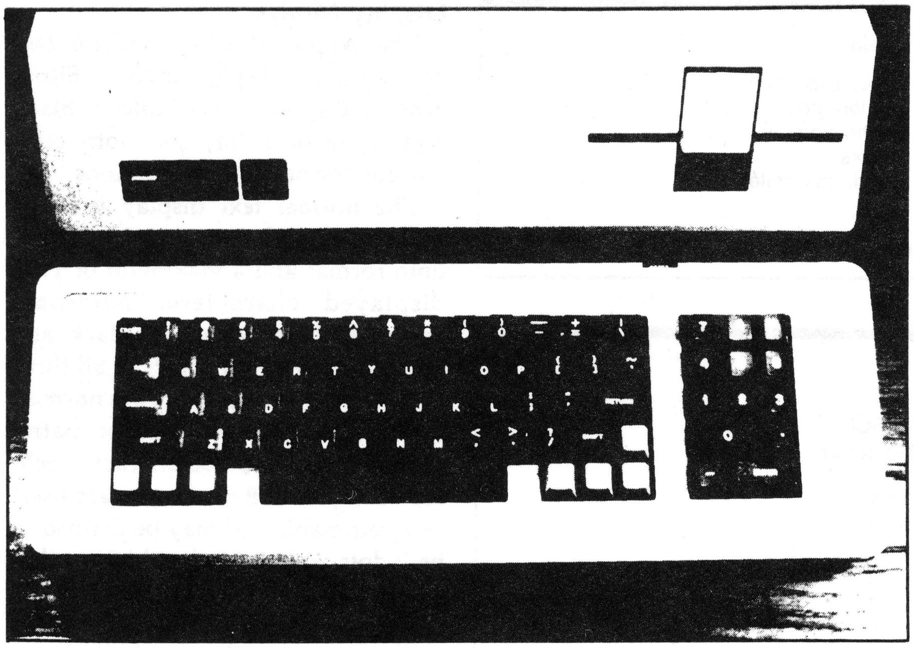


Photo 5: *The Apple III keyboard. Although it looks separate, it is actually part of the Apple III main enclosure.*

table with the key placement. The layout of the numeric keypad on the right, which resembles that of a calculator, allows easy entry of numeric data. The Apple III can also generate all 128 ASCII (American Standard Code for Information Interchange) codes without extra hardware.

In addition to the normal Shift, Control, and Caps-Lock modifier keys, the Apple III includes special Open-Apple and Close-Apple keys that you can define for special functions. All keys automatically repeat when pressed for more than ½ second, and the four cursor-movement keys each provide a 2-speed repeat—pressing gently repeats at 11 Hz, while pressing firmly repeats at 33 Hz.

Apple's SOS 1.1 operating system provides a 128-character type-ahead buffer so that keystrokes won't be lost if you continue to type while the system is busy. This buffer may be emptied, or flushed, if the program running needs to wait for a particular keystroke.

One of the biggest complaints about the original Apple II concerned the close proximity of the Reset key to the rest of the keyboard. In the Apple III the Reset key has been positioned at the rear edge of the keyboard enclosure, thus avoiding the accidental resets encountered in early Apple IIs. Simultaneously pressing Control and Reset simulates a power-up and reboots the system from the main disk drive.

In addition to the normal keyboard functions, a number of special control features are built into the Apple III keyboard. Pressing the Control key and one of the keys on the numeric pad will allow you to turn the video on and off, flush the type-ahead buffer, suspend screen output so that the processor can run at maximum speed, display control characters, or turn off the screen until the program requests an input.

In general, I found the keyboard versatile and pleasant to use. (Although the keyboard is actually part of the main enclosure, it is styled to appear as a separate unit. A convenient recess at the top can support a book or a pencil.) My only problem was that the very light touch required to avoid automatic key repeat sometimes caused me to produce extra characters. You have to break the habit of letting your hands rest on the keyboard while thinking about what to type next.

Mode	Format	Colors
0	24 by 40	black and white
1	24 by 40	16 foreground and 16 background colors
2	24 by 80	black and white

Table 2: *Apple III text display modes, screen formats, and color capabilities.*

Color	Color Value	ASCII Character	Gray Level
black	0	0	black
magenta	1	1	
dark blue	2	2	
lavender	3	3	
dark green	4	4	dark gray
gray	5	5	
medium blue	6	6	
light blue	7	7	
brown	8	8	medium gray
orange	9	9	
gray 2	10	:	
pink	11	;	
green	12	<	light gray
yellow	13	=	
aqua	14	>	
white	15	?	white

Table 3: *Table of graphics colors or gray levels produced by the GRAFIX driver routine. After opening the routine as an output device, colors may be selected by printing a CHR$(9) followed by an ASCII character. The color values shown are extracted from the lower four bits of the ASCII code transmitted. Higher-level graphics functions are provided by the BGRAF invocable module.*

Display Modes

The Apple III offers several text and graphics display modes. Either type of display is available in black and white or color, and both offer various formats and resolutions.

The normal text display is black and white, with a 24-row by 80-column format and a maximum of 1920 displayed characters. Alternate modes include 24 by 40 black and white and 24 by 40 color. In all three text modes the characters are normally displayed as a 5- by 7-dot matrix within a 7- by 8-dot character cell. However, all 128 characters are user-programmable and may be defined to be 7 dots wide by 8 dots high so that adjacent characters will touch in all directions if desired. (See table 2 for available text display modes.)

In the 40-column color-text mode, you can display 16 colors of characters on 16 colors of background. In combination with the user-definable character set, you can produce some surprisingly good color-graphics displays. For example, Apple's well-known "running-horse" demonstration program (shown in photo 6) is produced in color-text mode. The color values shown in table 3, although specified for graphics, can also be used for color text.

With four graphics modes, the Apple III's capabilities are significant-

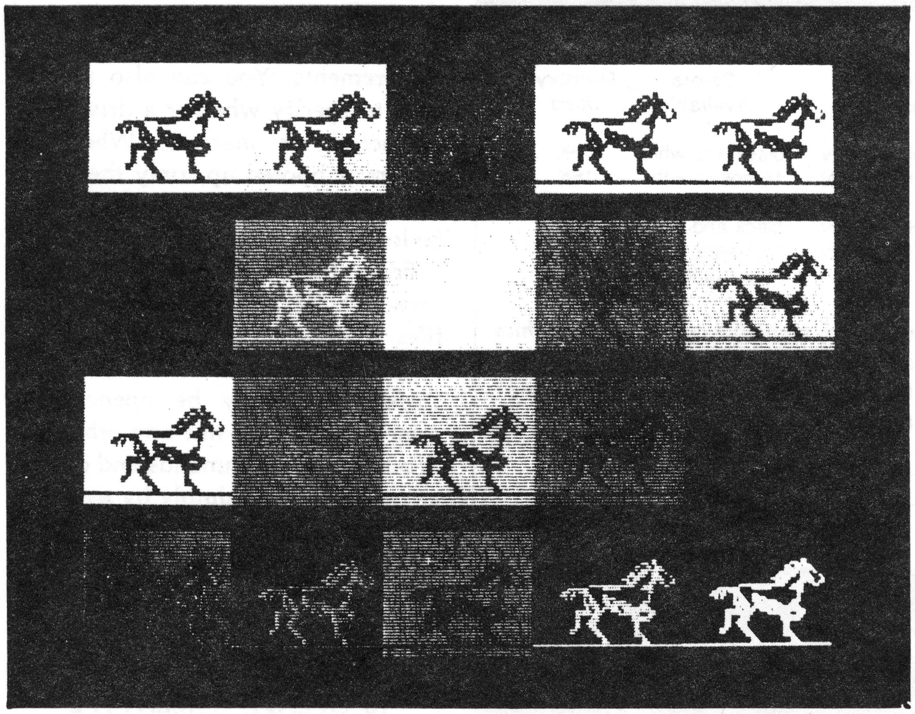


Photo 6: *The well-known "running horse" demonstration. This display was generated using the 24-row by 80-column color-text display mode using the Apple III's programmable character set to produce the special shapes required.*

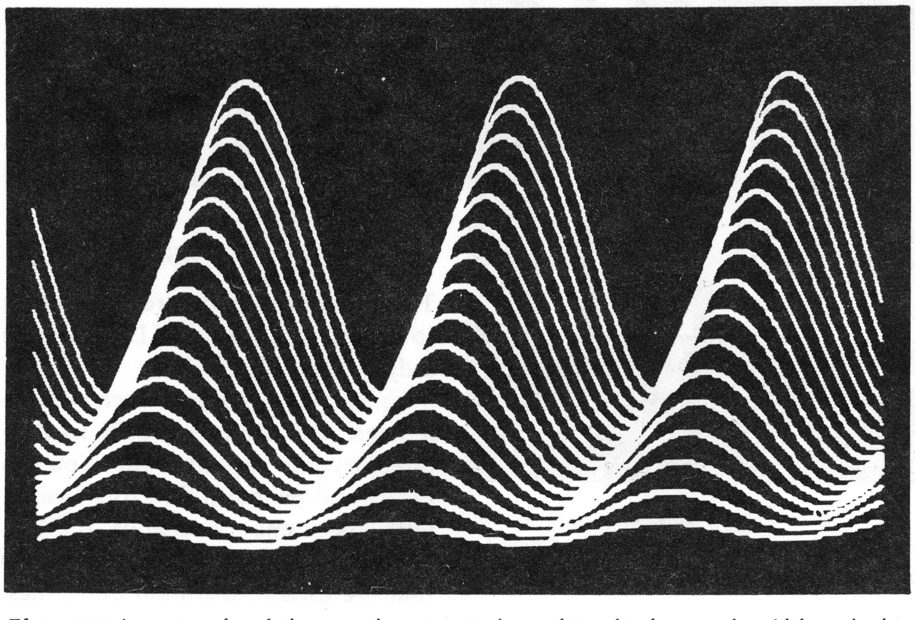


Photo 7: *An example of the 560- by 192-pixel graphics display mode. Although this mode doesn't offer color, it is ideal for displays that require fine detail.*

ly better than those of the Apple II (table 4 shows the available modes). The highest resolution offered is 560 by 192 pixels, black and white. This mode is useful for scientific or technical displays that require maximum resolution, as shown in photo 7. There is also a 280 by 192 black-and-white display mode.

The highest-resolution color display available is 280 by 192 pixels. Using this mode you can display up to 16 colors with some limitations. In each 7-dot-wide section of a given vertical coordinate, only two colors can be displayed. Bits that are turned on will display the specified foreground color, while bits that are turned off display the background color for that section. This is usually noticed only when lines of different colors cross. The limited color mode is useful for many applications where 16 colors are required but where maximum resolution is needed (an example is shown in photo 8).

The most colorful graphics mode is the 140- by 192-pixel 16-color mode. With no limitations on color placement, it is capable of producing very impressive displays (see photo 9). One of the more interesting techniques in this mode mixes various colors of dots to produce a variety of in-between shades of color. Using this technique, it is possible to produce several hundred colors on an Apple III.

Although the resolution is effectively reduced in the shaded areas, this method is typically used for filling in areas of pictures rather than for outlines, which are normally drawn in solid color. A talented artist with a digitizing tablet and the appropriate software can produce results like those shown in photo 10.

Apple SOS

Apple's SOS (Sophisticated Operating System) 1.1 is one of the more powerful operating systems available for an 8-bit microcomputer and offers features usually found only on larger machines. SOS supports multiple nested directories, handles interrupt-driven and DMA I/O, and manages the Apple III memory and hardware environment.

A unique feature of SOS is that there is no user interface. All communications with SOS are handled by the resident language (BASIC or Pascal for now) in a fashion compatible with the language syntax. For example, with Business BASIC you display a disk directory by typing CATALOG (or CAT), but in Pascal you would press F to enter the filter and then press E to get an extended directory. Rumor has it that Apple is working on a separate SOS user-interface package. This would allow access to SOS without requiring that a language be loaded into the system.

All Apple III I/O is handled by SOS through device drivers. Each

Circle 297 on inquiry card. ⟶

Graphics Mode		Graphics	Colors	Memory
Main Screen	Alternate Screen	Resolution	Available	Used
0	4	280h by 192v	black and white	8K
1	5	280h by 192v	16 colors with limitations	16K
2	6	560h by 192v	black and white	16K
3	7	140h by 192v	16 colors, no limitations	16K

Table 4: *The Apple III graphics modes, resolution, available colors, and graphics screen memory requirements. Each main mode allows two separate screen buffers so that one screen may be updated while the other screen is displayed. When the black-and-white gray-scale video output is used, the 16 colors are output as 16 gray levels from black to white.*

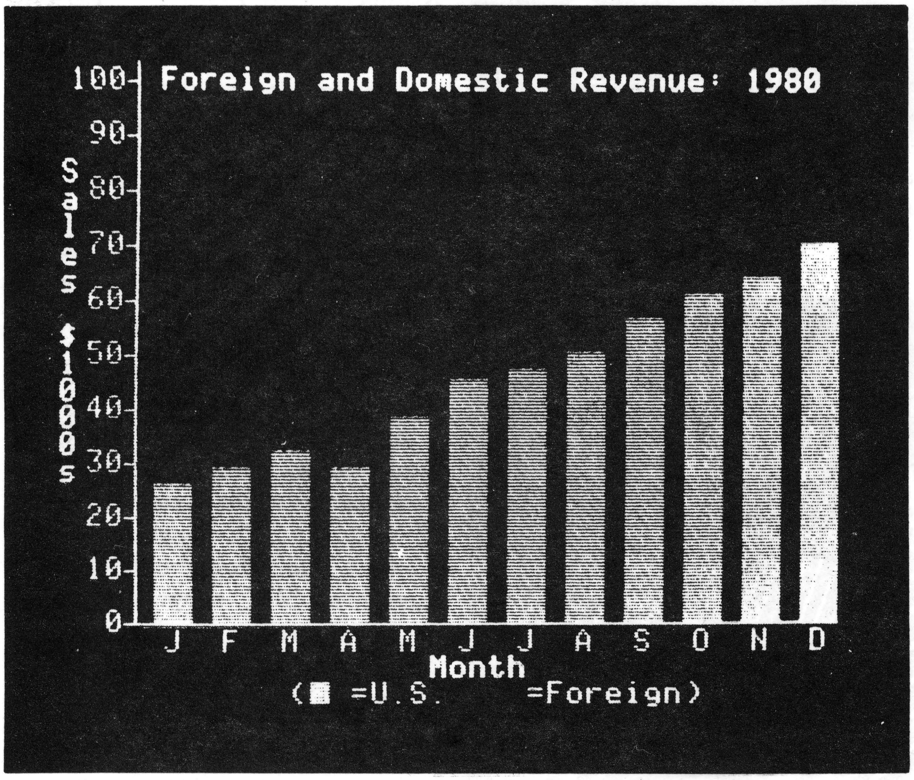


Photo 8: *An example of the Apple III's 280- by 192-pixel limited 16-color mode. While there are some limitations on the combinations of colors that can be displayed next to each other, this mode offers the highest color resolution and is useful in many applications.*

device driver is a group of routines designed to communicate with a particular hardware device and provide a uniform interface to SOS. For example, in a minimal Apple III system, you need the device driver .CONSOLE to handle the keyboard and text display, as well as .FMTD1 to handle the system floppy disk. Some of the other drivers included with the system are .AUDIO, .RS232, .PRINTER, and .GRAFIX. Even though the RS-232C interface and the graphics display hardware are included in the Apple III, they are considered optional I/O devices for programming purposes.

The System Configuration Program (SCP) provides a variety of tools that allow you to modify and reconfigure the system device drivers. Once the device drivers are specified, the SCP can regenerate a version of the system that meets your particular requirements. You can also use the SCP to specify whether a driver will be active or inactive. When the system is booted up, only the active drivers in the SOS.DRIVERS file will be loaded and require memory space.

From the programmer's point of view, device drivers are treated as files and can be used from either BASIC or Pascal. With Business BASIC they may be opened, accessed, and closed like any other file. (You can pass commands and data to an opened driver simply by using the PRINT# statement.) For example, the following Business BASIC lines would list the current program on the Silentype printer if the .SILENTYPE driver were installed:

```
10 OPEN#1, ".SILENTYPE"
20 OUTPUT#1
30 LIST
40 CLOSE#1
```

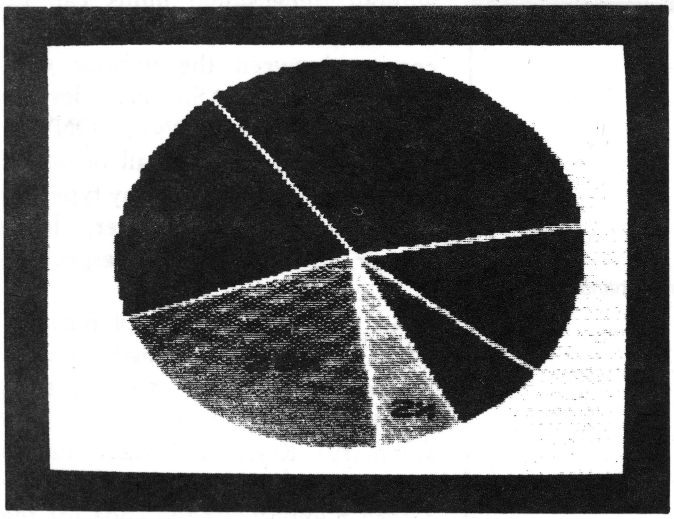
SOS allows the disk drives to be accessed either by their *device name* (e.g.,.D1) or by the *volume name* of the disk currently in the drive (e.g., MYDISK). Suppose that line 10 from the previous example were changed to read:

```
10 OPEN#1, "MYDISK/LISTFILE"
```

This would cause the program listing to be sent to a file called LISTFILE on a disk called MYDISK.

Unlike most systems which provide a single disk directory, SOS treats a directory like any other file. You can create and maintain directories easily with the same commands (LOCK, UNLOCK, RENAME, DELETE, etc.) that are used to maintain other files. You can assign any type of file to a directory, and any given directory may be a file assigned to another, higher-level directory.

The key to dealing with these nested levels of directories is the SOS *pathname*. Using device and file names separated by slashes, you can tell SOS what path to follow through various levels of directories. For example, the pathname /MYDISK/RECORDS/CHECKS/JAN.81/ would search the system for a disk volume



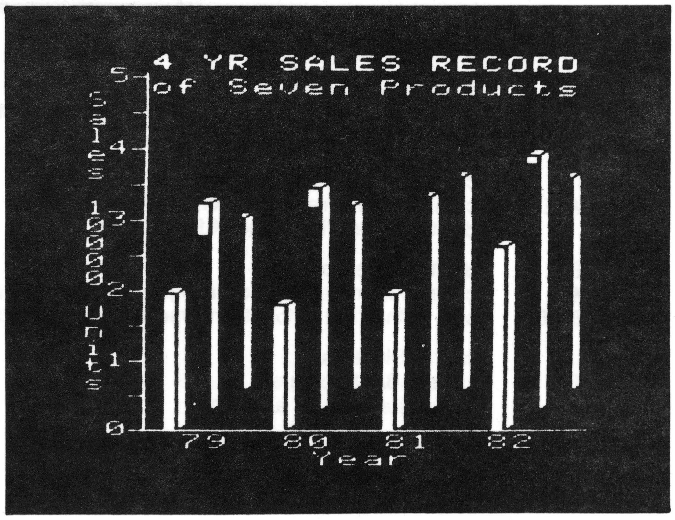


Photo 9: *Two examples of the 140- by 192-pixel full 16-color mode.*

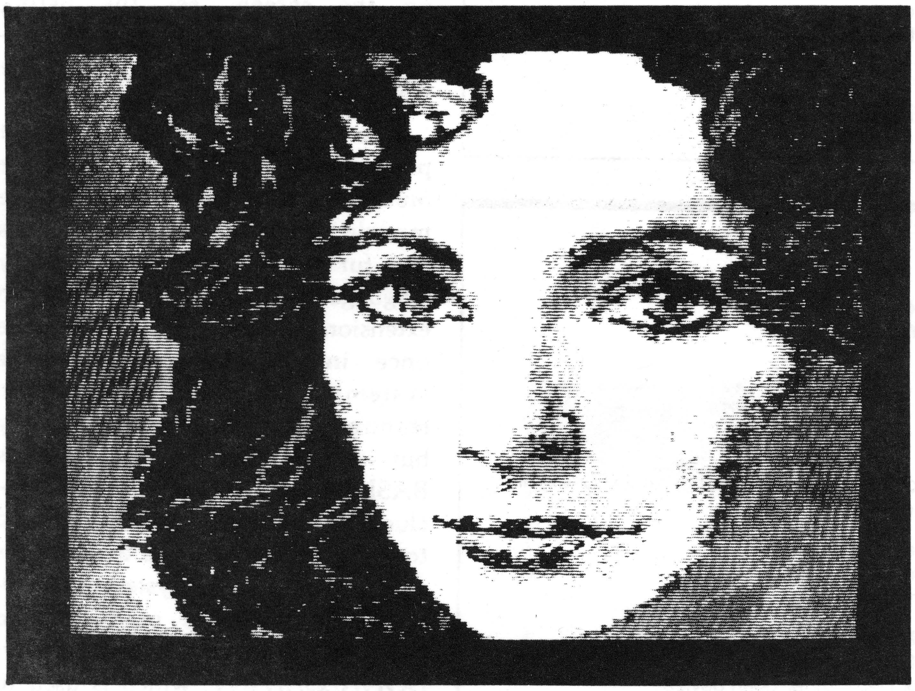


Photo 10: *A talented artist with a digitizing tablet and the appropriate software can produce results like this by using blended colors in the Apple III's 140- by 192-pixel color mode.*

named MYDISK, locate the directory RECORDS (which itself would contain the subdirectory CHECKS), and then locate the file JAN.81. The pathname specifies the sequence of directories to follow when accessing a given file. As a convenience, SOS provides a pathname prefix facility. By using PREFIX$ in the previous example, we could have set the pathname prefix to /MYDISK/RECORDS/ and then simply referred to CHECKS/JAN.81.

File types supported by SOS include DATA, which holds raw binary data; PASTXT (a Pascal text file); PASCODE (a machine language or Pascal program file); BASIC program files; ASCII files of unformatted text; PASDTA (Pascal data files); CAT or directory files; FONT files for the programmable character generator; and FOTO files, which store graphics screen images.

Business BASIC

Although it is fairly conventional, Apple's Business BASIC provides a combination of advanced and unique features that makes it an easier language to use than Applesoft BASIC. With Business BASIC you should be able to write shorter programs with fewer errors. (See tables 5a–5e for a summary of the language.)

Business BASIC supports both TEXT and DATA files. The commands PRINT# and INPUT# are used to access text files while READ# and WRITE# allow you to store or read any type of data in a DATA file. All files may be sequential or random access (with the record size defined when the file is created). You can also use the word CREATE to make new files and directories. Directory entries may be examined by reading sequential text records from a directory file.

The language also provides formatted I/O. To output data to either the screen or a file, you can specify the format with an IMAGE statement or within the PRINT USING statement. The Apple III's output formats are very flexible. Numbers may be printed in fixed-point, floating-point, scientific, or engineering formats. You can also align the right or left edges of the output to a particular column or center the output if you wish.

Four main data types are available in Business BASIC. You can use integers ranging from −32,768 to +32,767; real numbers with 6-digit precision, long-integers with 64-bit binary precision, or strings that can vary from 0 to 255 characters. Arrays

Command	Description
CATALOG	lists a disk directory
CHAIN	executes a program from disk leaving variables intact
CLEAR	clears program variables
CONT	continues interrupted program
CREATE	creates a new file or directory on disk
DEL	deletes a specified range of BASIC lines
DELETE	deletes a file from disk
HOME	clears current text window and places cursor in upper left-hand corner
INVERSE	sets further text output to inverse video characters
LIST	lists BASIC lines
LOAD	loads a BASIC program
LOCK	protects a file from alterations
NEW	clears a program and variables from memory
NORMAL	sets further text output to noninverse video
NOTRACE	turns off trace option
UNLOCK	removes protection from a disk file
RENAME	changes name of file on disk
RUN	loads and runs programs from disk or runs current program
SAVE	saves current program on disk
TEXT	sets screen to text mode with full-screen window
TRACE	turns on trace option

Table 5a: *A summary of Business BASIC commands.*

Statement	Description
CLOSE	closes all open files
CLOSE#	closes a particular file
DATA	standard DATA statements
DEFD FN	user-defined function
DIM	dimensions arrays
END	ends program
FOR. . .NEXT	standard FOR loop
GET	reads a single character from the keyboard or an EXEC text file
GOSUB	executes a subroutine
GOTO	continues execution at a specified line
IF. . .GOTO. . .ELSE	modified IF statement
IF. . .THEN. . .ELSE	standard IF statement
IMAGE	defines a PRINT USING format
INPUT	reads data from the keyboard
INPUT#	reads text from a disk file or other open device
INVOKE	loads an external file module of assembly-language routines
ON EOF#	sets up end-of-file error trap
OFF EOF#	turns off end-of-file error trap
ON ERR	sets up general error trapping
OFF ERR	turns off general error trapping
ON KBD	sets up keyboard interrupt handling
OFF KBD	turns off keyboard interrupt handling
ON GOSUB	standard computed GOSUB statement
ON GOTO	standard computed GOTO statement
OPEN# . . . AS	opens a file as INPUT, OUTPUT, or EXTENSION
OUTPUT#	sends subsequent output to file
PERFORM	executes a previously invoked routine
POP	removes one level of subroutine nesting
PRINT	prints to current output device or file
PRINT USING	prints using a given format

Table 5b: *A summary of Business BASIC statements.*

Table 5b continued on page 114

without dimensional limits can be created out of all four data types. To convert between the various data types, Business BASIC provides the numeric functions CONV, CONV%, CONV&, and CONV$, all of which will accept arguments of any type and will produce real, integer, long-integer, and string results, respectively.

An interesting feature of Business BASIC is its use of *reserved variables* to access and control certain system functions (see table 5f for a summary). Reserved variable names are used to hold error codes, the file record numbers, or the code for the last key pressed. Others may be used to hold or control the cursor position on the screen, set the listing FOR. . .NEXT loop indent level, control the listing line length, or set the SOS pathname prefix.

One of Business BASIC's most powerful features is its ability to use *invocable modules*. An invocable module is a file of external procedures and functions, written in assembly language or Pascal, that can act as an extension to the BASIC language once invoked (loaded into the system). The modules provide features that are sometimes necessary but were not built into the Business BASIC language. The modules include VOLUMES.INV, which is used to show which volumes and devices are present in the system; READCRT.INV, which is used to read characters from the video display; DOWNLOAD.INV, which is used to load special text fonts into the Apple III's character generator; and RENUMBER.INV, which provides a variety of functions including program renumber, append, and merge. Another more significant module is BGRAF.INV which provides all the graphics procedures and functions used by Business BASIC.

Once a module has been invoked, the external procedures and functions provided in that file are accessed by using the BASIC commands PERFORM and EXFN. For example, the line

```
PERFORM PENCOLOR(%BLUE)
```

would execute the procedure to set

Table 5b continued:

Statement	Description
PRINT#	prints to a particular output device or file
PRINT# USING	prints to a particular file or device using a given format
READ	reads information from DATA statements
READ#	reads information from a data file
REM	standard remark statement
RESTORE	resets read pointer to start of DATA list
RESUME	returns from on ON ERR statement
RETURN	returns from a subroutine, ON KBD or ON EOF routine
SCALE	adjusts PRINT USING decimal-point position
SPC	used in PRINT statements to output numbers of blanks
STOP	stops program execution
SWAP	swaps the values of two given variables
TAB	used in PRINT statements to position the cursor to a particular column
WINDOW	sets the text/scroll window size and position
WRITE#	writes information to a data file

Function	Description
ABS	absolute value
ASC	converts ASCII character to its numeric value
ATAN	arc tangent
BUTTON	paddle-button state
CHR$	converts number to equivalent ASCII character
CONV	evaluates expression—returns real number value
CONV$	evaluates expression—returns string value
CONV&	evaluates expression—returns long-integer value
CONV%	evaluates expression—returns integer value
COS	cosine
EXFN	executes an invoked external function that returns a real number value
EXFN%	executes an invoked external function that returns an integer value
EXP	exponential, base *e*
HEX$	returns a string that represents the hexadecimal value of the expression
INSTR	searches a string for a substring and returns location of occurrence
INT	largest integer less than or equal to argument
LEFT$	takes substring starting with first character
LEN	length of a string
LOG	natural logarithm
MID$	extracts a substring from a given string
PDL	returns a game-paddle position
REC	returns current file record number
RIGHT$	takes substring ending with last character
RND	random number
SGN	sign of argument
SIN	sine
SQR	square root
STR$	converts a number to a string
SUB$	inserts a substring into a given string
TAN	tangent
TEN	converts last four characters of a string from a hexadecimal text image to a decimal value
TYP	returns the data type of a file record
VAL	converts a string to a numeric value

Table 5c: *A summary of Business BASIC functions.*

the graphic drawing color to blue, provided that the variable BLUE has previously been defined properly.

While external procedures may be passed only integer values, external functions can return either integer or floating-point numbers. The reserved word EXFN% is used to call functions that return integers and EXFN accesses functions that return real values.

BASIC Graphics

Although you could use graphics from BASIC by simply opening the .GRAFIX driver and sending characters directly to it, the BGRAF.INV module provides a much cleaner and more powerful interface. It essentially adds a number of graphics commands to the Business BASIC language. (A similar library unit is included with Apple III Pascal.) The .GRAFIX driver must still be present and opened because you need a controller for the graphics hardware, but all graphics operations are performed by the external procedures and functions provided by BGRAF. The following two lines provide all the setup required:

```
100 OPEN#1, ".GRAFIX"
110 INVOKE "BGRAF.INV"
```

BGRAF provides all of the standard graphics operations. You can set PENCOLOR and the background FILLCOLOR, plot dots at absolute or relative positions with DOTAT and DOTREL, draw lines to absolute or relative points with LINETO and LINEREL, and position the graphics cursor with MOVETO and MOVEREL. BGRAF supports a graphics VIEWPORT that allows you to limit graphics drawing to a particular area of the display screen.

Text may be displayed with graphics by simply sending it to the opened .GRAFIX driver with a PRINT# statement. NEWFONT lets you redefine the graphics text font by specifying character form, height, and width. The SYSFONT command switches you back to the current text-mode display font.

Predefined images stored in integer arrays may be displayed with DRAWIMAGE. A given array may hold a

Circle 338 on inquiry card. ⟶

Operators	Type
+ – * / DIV MOD	arithmetic
AND OR = < > >< <> <= =< >= =>	logical
NOT	unary logical
+	string concatenation

Table 5d: *A summary of Business BASIC data operators. DIV and MOD apply only to the long-integer data type.*

Data Type	Type Name	Range
16-bit integer	integer	–32768 to 32767
64-bit integer	long-integer	±9223372036854775807 ($\pm 2^{63} - 1$)
32-bit floating point	real	$\pm 10^{38}$ with 6 digit precision
character strings	string	0 – 255 characters
arrays	(all types)	no dimensional limits

Table 5e: *A summary of Business BASIC data types and ranges.*

Variable	Description
EOF	holds reference number of file causing an EOF error
ERR	holds error type code of most recent error
FRE	holds amount of remaining bytes of memory available
HPOS	holds/controls cursor horizontal position
INDENT	holds/controls number of spaces to indent FOR . . . NEXT loops in listings
KBD	holds the ASCII value of the last key pressed
OUTREC	holds/controls the maximum line length output by the LIST command
PREFIX$	holds/sets current SOS pathname prefix
VPOS	holds/controls current cursor vertical position

Table 5f: *A summary of Business BASIC reserved system variables.*

Procedure	Description
DOTAT	plots a single dot at a given position
DOTREL	plots a dot relative to current position
DRAWIMAGE	draws a rectangular bit-map image at current position
FILLCOLOR	sets background color
FILLPORT	fills current VIEWPORT with FILLCOLOR
GLOAD	loads and displays a FOTO file from disk
GRAFIXMODE	specifies graphics mode and buffer choice
GRAFIXON	switches display to current graphics mode and buffer
GSAVE	saves current graphics display as a FOTO file on disk
INITGRAFIX	sets full-screen VIEWPORT, places cursor at upper left-hand corner and sets normal color and transfer tables
LINEREL	draws a line relative to current position
LINETO	draws a line from current to an absolute position
MOVEREL	positions cursor relative to current position
MOVETO	positions cursor at an absolute position
NEWFONT	used to specify a new graphics character font
PENCOLOR	sets current PLOT and DRAW color
RELEASE	frees highest graphics buffer memory
SETCTAB	sets a color-table entry
SYSFONT	causes normal system character set to be used as graphics character font
VIEWPORT	defines graphics-drawing window size and position
XFEROPTION	defines the logical operation that places dots on the screen
XLOC	returns graphics-cursor *x* position
XYCOLOR	returns color of dot on screen at current position
YLOC	returns graphics-cursor *y* position

Table 5g: *A summary of Business BASIC graphics procedures.*

number of images that can be selected with the DRAWIMAGE arguments.

One of the most interesting features of BGRAF is its control of color. By using two controllable processes—the color table and the transfer option—you can modify the effects of plotting and filling operations.

With 256 entries, the color table specifies which color results from plotting a dot of a given "source color" on top of a dot of a given "screen color." The color table is initialized to simply display the source color regardless of the existing color of the specified dot position. However, by altering the mapping conditions in the color table you can establish a color precedence. This precedence allows lines to appear to pass under or over existing images, or it can produce a number of other interesting effects.

To alter a color-table entry, you use the enternal function SETCTAB. The form of the statement would be:

```
SETCTAB (%SOURCECOLOR,
   %SCREENCOLOR,
   %RESULTCOLOR)
```

The following example would alter the color table so that when an orange dot was printed onto a blue background, the result would be green:

```
SETCTAB (%9, %6, %12)
```

Table 3 shows a summary of the graphics colors and their color values.

The black-and-white equivalent of the color table is the *transfer option*, which describes the logical operation used to place dots on the screen. De-

Circle 414 on inquiry card. ⟶

pending upon the option specified, a dot (or its inverse) may replace existing data, overlay it, invert it, or erase it with new data. The XFEROPTION procedure and an argument specify the transfer mode. The transfer option may also be used with color data, but predicting the results is difficult.

Although circle drawing and turtle graphics are not supported, BGRAF is still a very nice package of routines that should allow you to produce a wide variety of color graphics. (See table 5g for a summary.)

Business BASIC Performance

Although Business BASIC is much more powerful than the Apple II's Applesoft BASIC, it is not much faster. Tests with the series of sixteen benchmark programs shown in listing 1 indicated that while Business BASIC is faster than Applesoft in some areas, it is slower in others. The net result should be a slight to medium speed improvement, depending upon the program being run.

The best test in the series was probably the Sieve of Eratosthenes prime-number program used by Jim Gilbreath (see "A High-Level Language Benchmark," September 1981 BYTE, page 180). Although this program is more representative of average program execution than any of the other benchmarks, it uses only addition and subtraction and does not have a wide variety of BASIC statements. In this test, the Apple III proved to be slightly faster than the Apple II but slower than the IBM Personal Computer or the 4-MHz Z80.

The execution speed advantage of the 6502B is largely cancelled out by the complexity of Business BASIC.

From the results of this limited set of benchmarks, it seems that the execution speed advantage of the Apple III's 6502B is largely cancelled out by the increased complexity of Business BASIC. However, I suspect that in larger programs Business BASIC will turn out to be a good deal faster than Applesoft. The combination of its powerful built-in features and invocable modules will eliminate the code required in Applesoft to accomplish the same functions. Also, if the benchmark programs had included the appropriate code to turn off the video screen during time-critical calculations, an additional 30 percent speed increase could have been gained by allowing the 6502B to run at 2 MHz. This would have placed the Apple III ahead of the IBM and Z80 computers in many tests.

Although benchmarks always have some validity, they may or may not be significant in a given application. It is best to approach the results with caution—the programmer frequently

Listing 1: *Execution benchmark programs. See table 6 for a summary of their results.*

Listing 1a: *tests a null loop.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
320      NEXT I
```

Listing 1b: *tests REM execution time.*

```
100    FOR I=1 TO 5000
120      REM
140      REM
160      REM
180      REM
200      REM
210      REM
240      REM
260      REM
280      REM
300      REM
320      NEXT I
```

Listing 1c: *tests the IF. . .THEN statement.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      IF A<B THEN 320
320      NEXT I
```

Listing 1d: *tests addition.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=A+B
320      NEXT I
```

Listing 1e: *tests multiplication.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=A*B
320      NEXT I
```

Listing 1f: *tests division.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=A/B
320      NEXT I
```

Listing 1g: *tests exponentiation.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=A^B
320      NEXT I
```

Listing 1 continued on page 124

Listing 1 continued:

Listing 1h: *tests transcendental functions.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=SIN(A)
320      NEXT I
```

Listing 1i: *tests the LOG function.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=LOG(B)
320      NEXT I
```

Listing 1j: *tests the ON. . .GOTO statement.*

```
80    M=2
100    FOR I=1 TO 5000
120      ON M GOTO 80,320,100
320      NEXT I
```

Listing 1k: *tests the GOSUB/RETURN statement.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      GOSUB 1000
320      NEXT I
1000   RETURN
```

Listing 1l: *tests the INT (integer) function.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=INT(A)
320      NEXT I
```

Listing 1m: *tests the MID$ function.*

```
80    A$="abcdefghijklm"
100    FOR I=1 TO 5000
120      B$=MID$(A$,6,6)
320      NEXT I
410    PRINT""
420    END
```

Listing 1n: *tests random number speed.*

```
60    A=2.71828
80    B=3.14159
100    FOR I=1 TO 5000
120      C=RND(1)
320      NEXT I
```

Listing 1o: *tests the CHR$ function.*

```
80    A$="abcdefghijklm"
100    FOR I=1 TO 5000
120      C$=CHR$(50)
320      NEXT I
```

Listing 1 continued on page 126

Listing 1 continued:

Listing 1p: *Jim Gilbreath's Sieve of Eratosthenes prime-number program.*

```
1    SIZE=7000
2    DIM FLAGS(7001)
3    PRINT"only 1 iteration"
5    COUNT=0
6    FOR I=1 TO SIZE
7      FLAGS(I)=1
8      NEXT I
9    FOR I=0 TO SIZE
10       IF FLAGS(I)=0 THEN 18
11       PRIME=I+I+3
12       K=I+PRIME
13       IF K>SIZE THEN 17
14       FLAGS(K)=0
15       K=K+PRIME
16       GOTO 13
17       COUNT=COUNT+1
18       NEXT I
19     PRINT COUNT,"primes";""
```

Listing 2: *Disk-access benchmark programs. Listings 2a and 2b are write and read tests for the Apple III. Similar programs were used for the Apple II and the IBM Personal Computer.*

(2a)

```
40    A$="12345678123456781234567812345678"
60    B$=A$+A$+A$+A$
80    NR=500
100    OPEN#1,"TEST"
140    FOR I=1 TO NR
160      INPUT#1;B$
200      NEXT I
220    CLOSE#1
240    PRINT"DONE"
```

(2b)

```
40    A$="12345678123456781234567812345678"
60    B$=A$+A$+A$+A$
80    NR=500
100    OPEN#1,"TEST"
140    FOR I=1 TO NR
160      PRINT#1;B$
200      NEXT I
220    CLOSE#1
240    PRINT"DONE"
```

Listing #	Benchmark	Apple III Business BASIC	Apple II Applesoft BASIC	IBM Advanced BASIC	4-MHz Z80 MBASIC 4.51
1a	empty loop	8.9	6.7	6.43	5.81
1b	10 REMs	19.2	19.5	21.0	15.8
1c	IF. . .THEN	22.9	19.8	17.6	14.9
1d	addition	19.5	17.5	18.2	16.3
1e	multiplication	25.0	27.3	19.6	19.9
1f	division	27.6	28.8	23.8	24.9
1g	exponentiation	184.5	249.1	84.8	121.1
1h	sine(x)	98.0	193.1	73.9	63.1
1i	log(x)	87.1	113.6	49.4	55.4
1j	ON. . .GOTO	18.6	17.5	17.3	12.9
1k	GOSUB/RETURN	16.4	13.6	12.4	9.4
1l	INT(x)	20.0	19.3	18.1	15.5
1m	MID$	37.3	32.5	23.0	18.6
1n	RND(x)	90.5	33.1	18.4	19.7
1o	CHR$	26.8	23.5	16.2	13.4
1p	prime numbers	222.4	224.4	190.0	151.0

Table 6: *Table of execution times (in seconds) for a series of benchmark tests run on Apple III Business BASIC, Apple II Applesoft BASIC, IBM Personal Computer Advanced BASIC, and a 4-MHz Z80 computer running Microsoft's MBASIC 4.51. The results shown may or may not be indicative of performance in a particular application; they should be interpreted with caution. The results for the IBM Personal Computer and the Z80 microcomputer were taken from Gregg Williams' "A Closer Look at the IBM Personal Computer" (January 1982 BYTE, page 54). See listing 1 for the benchmark programs used.*

makes more difference than the machine. (The benchmark results are summarized in table 6.)

Apple II Emulation

The Apple III's ability to emulate an Apple II is an extremely useful feature that allows access to the tremendous volume of Apple II software. Virtually all Apple II DOS 3.3 programs in either Applesoft or Integer BASIC can be run on the Apple III without change—the few exceptions are those programs that require a RAM card or language system to operate. Also, some of the Apple II arcade games use their own routines to read the game paddles rather than calling the routines in the Apple II's monitor ROM. These programs will run but will not operate correctly.

To use the Apple II emulation mode, you must boot a special emulation disk and select either Applesoft or Integer BASIC as the available language. Since the Language Card is not emulated, only one language at a time can be resident. The Apple III serial port can be configured to emulate either an Apple II serial card or a communications card. The data rates and carriage-return handling can also be specified. Once the emulation parameters are specified or the defaults accepted, you can boot a normal Apple II DOS 3.3 disk and start running.

The emulation mode has a few minor weak points. If you have an Apple III Silentype printer, it will not be accessible in emulation mode unless you install an Apple II Silen-

At a Glance

Name
Profile Winchester-technology disk drive

Manufacturer
Apple Computer Inc.
20525 Mariani Ave.
Cupertino, CA 95014
(408) 996-1010

Price
$3499

Storage Capacity
5 megabytes (equivalent to about 35 normal Apple 5¼-inch floppy disks)

Size
Height 4.39 inches (11.5 cm), width 17.28 inches (43.89 cm), Depth 8.81 inches (22.38 cm)

Weight
11 pounds (5 kg)

Power Required
110 volts AC, (U.S.), 35 watts

Hardware Required
Apple III computer

Software Required
Apple SOS 1.1

Organization
Four data surfaces, 153 tracks per surface, 16 sectors per track, 512 bytes per sector, 2448 sectors per surface, 9792 sectors per drive

Specifications
Data transfer rate: 5 megabits per second; average seek time: 95 milliseconds; rotational speed: 3600 revolutions per minute; ready to operate: 60 seconds

Interface
Interface card occupies one Apple III expansion slot; one drive per interface card, up to four drives per system

Special features
Power-up self-test and disk scan; automatic bad-sector relocation; error checking and limited error correction

type interface card, which may violate FCC radio-frequency radiation limits. Nor can you access the Profile hard-disk drive—Apple II and Apple III files won't mix on the same disk. Also, the RGB (red-green-blue) video outputs will not provide color signals while emulating Apple II graphics, but the composite video outputs will work normally.

The Profile

The Profile hard-disk drive is the newest component of the Apple III family and a worthy occupant of an expansion slot. With a 5-megabyte capacity, integral Z8-based controller, and built-in power supply, the Profile is a self-contained intelligent subsystem with its own self-test, error checking, and bad-sector relocation facilities.

When powered up, the Profile's controller waits for the disk to come up to speed and does a data integrity check by stepping from track to track to verify that all disk sectors read correctly. If a bad sector is found, either during this process or during normal activity, the Profile attempts to correct the data errors and then relocates as much data as possible to an alternate good sector.

The key component in the Profile is the ST-506, a 5¼-inch hard-disk drive manufactured by Seagate Technology Inc. The ST-506 uses the sealed disk environment and low-altitude (10-microinch) flying heads that characterize all Winchester-technology disk drives (see photo 11). Because a number of vendors produce drives that are plug-compatible with the ST-506, Apple should have no trouble producing Profiles even if Seagate's supplies get short.

During operation the disk drive is relatively quiet, emitting a soft tone as it steps from track to track. Between accesses you can hear the main drive motor, but the sound should not be obtrusive or even audible in most office environments.

The Profile is styled to match the rest of the Apple III system and may be positioned on top of or adjacent to the computer.

I found the Profile a pleasure to use. Its capacity is equivalent to that of about 35 normal Apple floppy disks, and its data throughput is about 10 times faster. Viewing its capacity in other terms, the Profile can hold over 1200 pages of typed text or more than 300 high-resolution graphics pictures occupying 16K bytes apiece.

The Profile's performance is excellent. In the disk-access benchmark programs shown in listing 2, the Profile effectively tripled the program speed when compared to an Apple or IBM floppy disk. Considering that a significant proportion of the program execution time is used to execute the BASIC program statements, the actual increase in disk-access speed would seem to be even higher. (The results of the disk-access benchmarks are summarized in table 7.)

The weakest point of the Profile and other similar products is data backup. If a hard disk fails, you can lose a great deal of important data. The only solution is to periodically back up the most critical files onto floppy disks or onto a second Profile hard-disk unit. (Apple Computer will happily allow you to connect up to four Profiles to your Apple III, at a total cost of $13,996 in addition to the cost of the Apple III.) However, chances are very slim that the entire Profile would be wiped out if a critical component failed. After repair, it should be possible to recover virtually all the original data in most cases.

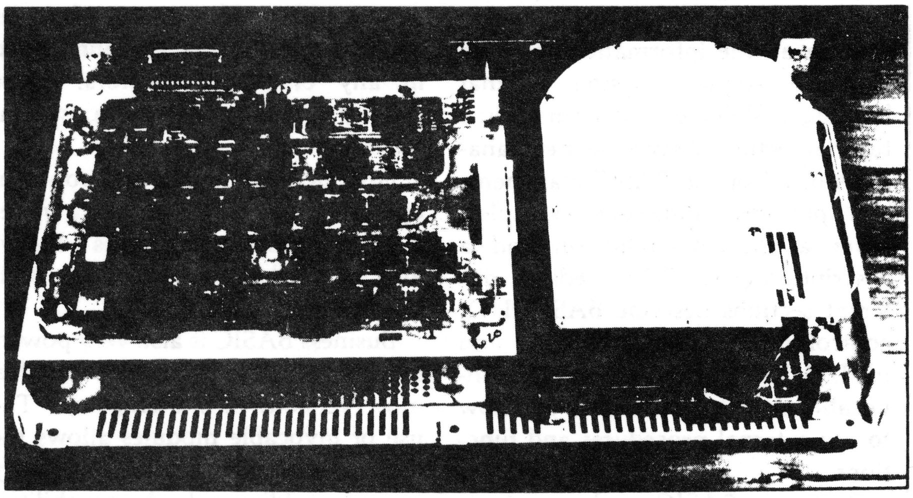


Photo 11: *The Profile with its top cover removed. The intelligent controller is shown on the left with the switching power supply beneath it. The HDA (hard-disk assembly) with its sealed internal environment is mounted on the right.*

	Apple III Profile	Apple III Floppy Disk	Apple II Floppy Disk	IBM Floppy Disk
Write	13.2	37.3	234	32
Read	10.2	33.2	273	22.9

Table 7: *A summary of disk-access-time benchmarks comparing the performance of the Apple III Profile hard-disk drive and the Apple III, the Apple II, and the IBM Personal Computer floppy-disk drives. The table shows the times (in seconds) taken to read and write 500 disk records.*

At $3499, the Profile isn't inexpensive—none of the available hard-disk subsystems are—but it provides a truly significant extension to the capabilities of the Apple III system.

Documentation

Apple Computer's documentation has always been excellent, and the manuals provided with the Apple III are no exception. All the manuals are in the familiar 6- by 8½-inch (12.8- by 21.6-cm) format, and a new flap has been added to the back cover so that the manual title is visible while the book is on the shelf. The manuals are all clearly written with numerous charts, tables, and screen photos to illustrate points described in the text.

With a Business BASIC system, you receive four manuals: the *Owner's Guide* and *Standard Device Drivers* deal with Apple III features and SOS, while volumes one and two of *Apple Business BASIC* provide a comprehensive description of the language.

The *Owner's Guide* explains how to set up the Apple III system and describes various aspects of SOS and the Apple III hardware. There are sections about system installation and start-up, the operating system, the System Configuration Program, and the machine itself. Appendixes explain error messages, describe proper disk care and handling, give I/O port specifications, and tell you how to use the Apple II Emulator. The information is presented in a clear, easy-to-read style and should be sufficient to get any novice started.

Standard Device Drivers provides complete specifications and descriptions of the operation of all of the standard I/O device-driver routines. After a short section that explains what device drivers are, the manual describes the System Configuration Program. Separate sections describe each individual driver in detail. The appendixes contain quick references for all the drivers, an explanation of the system error messages, and a description of the console data formats.

With a BASIC system, you'll get *Business BASIC* volumes 1 and 2. Although the manuals were not designed to teach BASIC, the 335 pages contain all the information required to learn Apple's version of that language. Volume 1 is primarily a tutorial section; it gives clear explanations of all of the BASIC statements and provides numerous examples. After a short introduction and a description of the BASIC editor, different sections describe BASIC I/O, control of program execution, and file I/O. The manual also explains invocable modules and shows you how to use external procedures and functions.

Business BASIC volume 2 is primarily a quick reference quide that will be of most use to people who have some familiarity with the Business BASIC language. Within the BASIC reference section, each language statement and function is described and shown in an example along with descriptions of any error messages that might be produced when it is used. Separate appendixes describe error messages and their causes, explain variable memory usage, tell how to program for maximum speed, and give syntactic definitions of the Business BASIC language. The Graphics invocable module (BGRAF.INV) is described in a 57-page section that gives detailed examples of plotting and drawing, saving pictures on disk, creating graphics text fonts, and setting up your own color and transfer tables.

If you purchase Apple III Pascal, you'll get an additional four manuals that describe the Pascal system, utility programs, and the Pascal language. One distinct benefit of Apple III Pascal is that the description of the Pascal assembler provides details about the 6502 enhanced features that are not found in any of the other manuals. Unfortunately, even though the BASIC invocable modules are written in Pascal, the manuals do not tell you how to write them. This may not be important to small-business users; nevertheless, the information should be available.

Summary

It is impossible to do the Apple III justice in one article. The machine is very flexible and has a mix of features and capabilities that are unmatched in any of its competitors. Some points, however, deserve special mention.

First, SOS is a unique and powerful operating system; it provides a variety of features that, as far as I know, are not available on any other 8-bit machine.

Business BASIC is also very powerful and includes options not found in most versions of the language. The use of invocable modules allows the user to maximize available memory space by adding only the capabilities needed. Its I/O-formatting and file-handling capabilities are extremely versatile and, for most business data-handling applications, will allow programs to be shorter and easier to debug.

As for hardware, although some people might argue that Apple should have chosen a more advanced microprocessor than the 6502B for the Apple III, I think the company made the right choice. Without the 6502B it would have been difficult, if not impossible, to transfer files and programs from the Apple II to the III, and Apple II emulation would not have been possible. Admittedly, it was a conservative choice—more powerful processors are available—but actual processor performance is much less important than software availability. Apple's choice clearly maximizes the usability of the system.

The Profile hard-disk drive is a significant enhancement to the Apple III. Its speed and high capacity will eliminate 99 percent of the disk swapping required when using only floppy disks, and the SOS nested directory structures will keep it well organized.

Finally, one of the strongest points in favor of the Apple III is Apple Computer Inc. When early Apple III users had problems with the first machines, Apple simply replaced the entire computer immediately—as many as two or three times in some cases. This unqualified backing of its products shows a commitment to customer satisfaction unequaled in the industry. ■

This author put the Apple III through its paces and found more than one application for this versatile new machine.

How Does the Apple III Stack Up?

By James E. Kelley, Jr.

About four months ago an Apple III (128K) took up residence in my engineering management consulting business, accompanied by a NEC Spinwriter (5515), a second disk drive and a black and white monitor.

I needed word processing for writing and indexing procedures manuals and for preparing technical reports and marketing materials. I also wanted program development capability for producing commercial-quality software packages. My other applications are proprietary, but are intended to exploit computer graphics and to involve top executives more personally in the use of microcomputers in their businesses.

A third application, the one my friends say is the real justification, is the analysis, comparison and classification of medieval marine charts—a long-standing interest of mine.

The Bare Bones

What comes with the Apple III? There is an owner's guide with system utilities disk, a standard devices drivers manual with Apple II emulation disk, an Apple Business Basic reference manual with system disk, Apple II DOS manual with DOS 3.3 system master and DOS 3.3 BASIC disks, two VisiCalc manuals with sampler and system disks, a system demonstration disk and two blank disks.

Address correspondence to James E. Kelley, Jr., 7602 Spring Ave., Melrose Park, PA 19106.

That is not very much. But I knew in advance that I'd be a pioneer with the new machine, and would have a lot of development work to do.

When I placed my order last fall I expected to get Apple's word processor with the equipment. I understand delivery has been delayed until 1982! Since I really needed a word processor, my dealer recommended Muse Software's Supertext for the interim. This package was developed for Apple II, but fortunately it works under the emulator.

I have been using this word processing package for several months, and although it's doing the job, it's not something I want to live with forever. Using control characters for uppercase letters and other functions is certainly awkward, and it's inappropriate for the Apple III. I have to insert four disks just to get started each session. The package is unforgiving if you accidentally hit the wrong key under emulation, so frequent saves are your only insurance against redoing large blocks of text.

Getting Started

With a little instruction from the dealer I started wading through the manuals. I soon found that you need to know something about the Apple II—the manual writers tacitly assume that the reader has experience with the earlier machine. Be sure to buy some of the Apple II manuals, especially the Applesoft and the Apple II reference manuals.

One of the quickest ways to learn the fundamentals of this equipment is to print out and study a copy of the various BASIC programs that come on the demonstration and the Apple Business Basic disks. They are mostly well-documented with remarks. You can follow what they do by blocking off sections of the programs with END statements and then executing them. There are a number of demos in Integer and Applesoft BASIC on the DOS 3.3 system master which are also useful to print out and study.

There is a helpful Pascal subroutine (READCRT.INV) on the BASIC disk which reads the Text mode console screen. To my knowledge, none of the reference manuals tells how to do this, or even refers to this subroutine.

Plaudits

I was concerned that reading screen text for long periods would bother my eyes, but this has not happened. I like the 80-character per line format, with upper- and lowercase letters. It is much superior to Apple II's 40-character format.

The Apple III provided my first experience with interactive programming. It's great—I've never written and debugged programs so rapidly. Every programmer should have this facility. Your debugged product can be compiled into machine language later on.

The edit feature is especially convenient. You can move the cursor all over the screen so you can assemble lines of coding from previously keyed material or insert corrections at will.

Text and Program Manipulation

The string manipulation features of Apple II have been extended on the Apple III to include HEX$ (decimal to hexadecimal conversion), TEN (hex to decimal), INSTR (first position of a substring) and SUB$ (substring replacement). The related CONV(ert) commands, also new, permit all data-type conversions among real, integer, string and long integer expressions.

A really nice feature of the Apple Business Basic is the ability to LIST programs, or sections of them, to a Text file and then to read them back with an EXEC(ute) command. This lets the programmer manipulate programs within BASIC. For instance, I store useful subroutines as Text files, to be called when assembling a program by the appended utility, called PATCH, which is temporarily stored at high number-line positions. This utility renumbers the subroutine and inserts it in the program being assembled (using the EXEC command). Indeed, PATCH itself is called as needed using the EXEC command, and then deleted when no longer needed.

Unfortunately, the LIST and DEL commands currently accept only explicit values of line numbers and not variables whose values can be substituted under program control. Further, the DEL command does not function in deferred mode, but requires direct operator involvement in utilities using it. The EXEC command works in deferred mode if the file pathname is placed in quotes (not mentioned in the manuals).

Since the EXEC command imitates keyboard input operation, the statements need not be in ascending sequence in the text file. This makes a limited form of numeric sorting possible. Generate BASIC REM statements, where the statement numbers are the sort keys, and the records or record identifiers are stored behind the REM. Distribute these statements to a text file and read back using an EXEC command. Voila! They are in sort order in the instruction memory. Now list them back to a text file and input them to a file for further processing under BASIC program control.

I have also used the program memory for dated text items (e.g., bibliographic references, follow-up notes). The date (or reasonable facsimile) provides the line number, a REM command avoids syntax tests, an additional digit or two provide a secondary classification (e.g., type of bibliographic item, individual involved or type of follow-up action), and finally the text in English (all in under 255 characters). This information is saved in a Text file, and read back using the EXEC command, to be updated just as you would correct a BASIC program using the cursor controls. A simple print program, selecting on the secondary classification if desired, provides up-to-date bibliographic or follow-up lists by individual or function.

Quibbles

For an old FORTRAN programmer, the Apple Business Basic I/O statements are unnecessarily confusing and unwieldly. The use of INPUT/PRINT and READ/WRITE commands depends on the type of file addressed, and the punctuation of the variable list (commas and semicolons) varies with the device type. I'm forever making syntax errors. Handling of arrays during I/O is also inconvenient compared to FORTRAN. I'm not sure yet what strategy to follow in designing file layouts to get the best speed of operation. More information on how BASIC handles I/O commands internally would be helpful here.

A simple program listing requires typing many characters, and I am forever doing it wrong. Here's the full sequence:
open#3,''.printer'':output#3:list:output#0:close#3:end.
I usually put this sequence at the end of my programs for easy call with a GOTO. I suppose these complications can be justified by the flexibility they provide—like the abominable JCL (job control language) of the IBM 360s and 370s.

In my shop, certain file numbers are standard, like standard device numbers on the old IBM systems (e.g., 1 = console [keyboard and screen], 3 = printer; 4 = graphics, etc.). When I pick up a piece of coding and read PRINT#3 I know automatically that the NEC Spinwriter is the device involved.

Big Disappointment

My biggest disappointment is being locked out of the system. No way is currently provided for programming in machine language. POKE, PEEK and CALL are not in Apple Business Basic's vocabulary. BASIC is fine for interactive operations at the console. But because it's interpretive, it's much too slow to exploit Apple III's intrinsic capabilities for number crunching, array handling or involved string manipulation. I like the idea of writing mainlines and most I/O and other overview functions in BASIC (for easy debugging at the console), while writing the inner loops (where time counts) in machine or a compiled language (e.g., Pascal,

Program listing. PATCH programming aid.

```
63000  REM  *************************************************************
63005  REM  * 5/29/81                                           VER 0.1  *
63010  REM  *                      ***  PATCH  ***                        *
63015  REM  *                                                             *
63020  REM  *               An APPLE III Programer's Aid                  *
63025  REM  *                                                             *
63030  REM  *                      Developed by:                          *
63035  REM  *                                                             *
63040  REM  *                   JAMES E. KELLEY, JR.                      *
63045  REM  *                   Melrose Park, Penna.                      *
63050  REM  *                                                             *
63055  REM  *                                                             *
63060  REM  *************************************************************
63065  DIM stmt$(200):GOTO 63620
63070  :
63075  REM   ***  PULL OUT STATEMENT #  ***
63080  line$=""
63085  line$=line$+b$:IF a%=LEN(string$) THEN 63110
63090  a%=a%+1:b$=MID$(string$,a%,1)
63095  IF b$<"0" OR b$>"9" THEN 63110:ELSE 63085
63100  :
63105  REM   ***  CALC EQUIVALENT STMNT #  ***
63110  FOR i=1 TO lim:IF line$=stmt$(i) THEN 63120
63115    NEXT:nline$=line$:RETURN
63120  nline$=STR$(min+dif*(i-1)):RETURN
63125  :
63130  REM   ***  ASSEMBLE OUTPUT STRING  ***
```

More

FORTRAN).

Pascal, when it's available, is supposed to substitute for coding in machine language. Or perhaps it will provide the means to get directly at the 6502A and give the user true ownership and control of his equipment.

For me, being locked into BASIC exclusively means deferring the development of commercial-quality systems. The graphics die in BASIC. Image development is too slow for interactive graphics—rubber-banding and the like. At least I'm not smart enough to do these things effectively with the currently available facilities for Apple III.

Glitches

Three valuable Applesoft program statements have been omitted in Apple Business Basic: FLASH (causes text to blink), ROT(ate) (a shape) and SCALE (a shape). SCALE is still in the vocabulary, but it now relates to formatting output for printing.

The first version of the BASIC system has serious bugs in the graphics package. When you use mode 2 (high-resolution) to plot data or to read the screen in the horizontal range of 512, the image is recorded in the range 256. I first encountered this when dumping a picture on the demonstration disk from the mode 2 buffer to the printer while experimenting with the NEC Spinwriter's graphics. I thought I had an error in my program. But the problem also arises when you plot curves on the high-resolution screen.

The Newfont and Drawimage features of the graphics package do not give correct results either. If you try the explanatory example in the manual you'll find this to be true.

Comments on PATCH

PATCH is to be stored as a text file and called by the statement EXEC .D2/PATCH. It is read into memory without disturbing the program in residence unless it has statement numbers of 63000 or more. Enter PATCH with a GOTO 63000. The menu will be displayed (lines 63615–63675).

In operation, PATCH is pretty well self-explanatory. However, I'm sure it won't handle all possible error conditions, although those which I commonly make are programmed to give me a second chance.

Four working options are available in PATCH as currently written:

Listing continued.

```
63135  out$=LEFT$(string$,m%)+nline$:d%=LEN(string$)-m%-LEN(line$)
63140  IF d%<=0 THEN string$=out$:RETURN
63145  string$=out$+RIGHT$(string$,d%):RETURN
63150  :
63155  REM   ***  Process THEN, ELSE & USING  ***
63160  a%=INSTR(string$,c$,i%):IF a%=0 THEN RETURN:REM No more THEN's
63165  b$=MID$(string$,a%+j%,1):IF b$<"0" OR b$>"9" THEN i%=a%+j%:GOTO 6316
       0
63170  a%=a%+j%:m%=a%-1:GOSUB 63080:GOSUB 63135
63175  IF d%>j% THEN i%=LEN(string$)-d%:GOTO 63160
63180  RETURN
63185  :
63190  REM   ***  Process GOTO, GOSUB  ***
63195  a%=INSTR(string$,c$,i%):IF a%=0 THEN RETURN:REM No more GOTO's
63200  b$=MID$(string$,a%+j%,1):IF b$<"0" OR b$>"9" THEN i%=a%+j%:GOTO 6319
       5
63205  a%=a%+j%:m%=a%-1
63210  GOSUB 63080:GOSUB 63135
63215  IF d%<=0 THEN RETURN
63220  m%=m%+LEN(nline$)+1:b$=MID$(string$,m%,1)
63225  IF b$="," THEN a%=m%+1:b$=MID$(string$,a%,1):GOTO 63210
63230  IF d%>j% THEN i%=LEN(string$)-d%:GOTO 63195
63235  RETURN
63240  :
63245  REM   ***  MAIN LINE  ***
63250  :
63255  INPUT#6;string$:IF LEN(string$)<=0 THEN PRINT#7;string$:GOTO 63255
63260  :
63265  REM  Process STATEMENT #
63270  a%=1:line$="":GOSUB 63090:d%=LEN(string$)-LEN(line$)-1
63275  string$=blnk$+nline$+RIGHT$(string$,d%)
63280  i%=1:c$="THEN":j%=5:GOSUB 63160:REM "THEN"
63285  i%=1:c$="ELSE":j%=5:GOSUB 63160:REM "ELSE"
63290  i%=1:c$="GOTO":j%=5:GOSUB 63195:REM "GOTO"
63295  i%=1:c$="GOSUB":j%=6:GOSUB 63195:REM "GOSUB"
63300  i%=1:c$="USING":j%=6:GOSUB 63160:REM "USING"
63305  PRINT#7;string$:GOTO 63255:REM Get next input string
63310  :
63315  REM   ***  INITIALIZE  ***
63320  work$=".d2/renumworkfile":ON ERR GOTO 63450:CREATE work$, TEXT
63325  OFF ERR:lim%=200:qt$=CHR$(34):blnk$=CHR$(32)
63330  PRINT:PRINT"Is Routine Currently in MEMORY (Y/N)";:INPUT q$
63335  jmp=1:IF q$="y" THEN jmp=2:x=1:GOTO 63470:REM Go CAPTURE Routine
63340  PRINT:PRINT:INPUT"KEY Pathname of SOURCE Routine: ";name$
63345  :
63350  REM   ***  SET UP STMT$ TABLE  ***
63355  i=1:ON ERR GOTO 63455
63360  OPEN#6 AS INPUT,name$:OFF ERR:ON EOF#6 GOTO 63375
63365  INPUT#6;string$:IF LEN(string$)<=0 THEN 63365
63370  a%=INSTR(string$,blnk$,2):stmt$(i)=MID$(string$,2,a%-2):i=i+1:GOTO 6
       3365
63375  CLOSE:OPEN#6 AS INPUT,name$
63380  ON EOF#6 GOTO 63530
63385  lim=i-1:OPEN#7 AS OUTPUT,work$
63390  ON ERR GOTO 63425
63395  PRINT:INPUT"KEY STARTING STATEMENT & STEP: ";min,dif:OFF ERR
63400  a=min+lim*dif:IF min<1 OR dif<1 THEN 63420
63405  IF min-INT(min)>0 OR dif-INT(dif)>0 THEN 63420
63410  IF a>63999 THEN PRINT:PRINT"New Line #'s EXCEED Capacity Limit of 63
       999":GOTO 63390
63415  GOTO 63255
63420  PRINT:PRINT"POSITIVE Integers PLEASE!!":GOTO 63390
63425  PRINT:PRINT"Illegal Quantity":GOTO 63395
63430  :
63435  REM   ***  CLOSING ROUTINE  ***
63440  CLOSE:PRINT"DONE -- RENUMBERED ROUTINE INSERTED"
63445  EXEC".d2/renumworkfile":END
63450  DELETE".d2/renumworkfile":GOTO 63320
63455  PRINT:PRINT; TAB(10);"<<< Pathname "name$" does NOT Exist >>>":PRINT
       :INPUT"Press ANY Key FOR MENU";q$:GOTO 63620
63460  :
63465  REM  ***  CAPTURE ROUTINE AS TEXT FILE  ***
63470  n$=".d2/captureworkfile":ON ERR GOTO 63515
63475  CREATE n$, TEXT:OFF ERR:OPEN#8,n$
63480  s$=" 63505 list ":t$=" del "
63485  GOSUB 63860
63490  PRINT:PRINT"EXECUTE the Following TWO Statements in IMMEDIATE Mode"
63495  PRINT:PRINT;s$:PRINT" goto 63500":PRINT CHR$(11);CHR$(11);CHR$(11);:
       END
63500  OUTPUT#8
63505  LIST 63000 TO 63999
63510  OUTPUT#0:CLOSE#8:name$=n$:ON x GOTO 63355,63755,63840
63515  DELETE n$:GOTO 63470
63520  :
63525  REM   ***  CLOSING ROUTINE  ***
```

More

List Routine. This can be done in unedited format as you would using the LIST command, or in edited format as the appended listing of PATCH illustrates. If many instructions are crammed into a single statement, you may have trouble getting wraparound on the printer, and difficulty reading the results when text and statement numbers appear in the same column position. The edited listing is obtained by capturing the program lines (asked for by PATCH) in a text file. They are then read back, edited and printed.

Capture Routine as Text File. Since this function is used as a subroutine for the other functions, I provided it as an option. I've used it for saving pieces of programs for editing as more general subroutines, or for saving information in DATA statements.

Extract, Renumber, Reinsert Routine. This is the principle function of PATCH. Input routines may be in a text file on disk or already resident in memory; the program asks you to specify. If the routine is in memory you are asked to identify the lines (they need not be contiguous groups of statements). You are then asked to specify the new starting statement number and the stepping constant. The routine is renumbered and placed in a text file, and the program asks whether you want to rename that file for later use. It then asks if you want to load the renumbered routine into memory. If the routine was taken from the program in memory, you are asked if you want to first delete the lines from which it derived. If you elect to load the routine, you stop at an END statement.

Reinitiate PATCH if you want the menu for more operations. (EXEC uses the console, so there is confusion when PRINT statements are performed while EXEC is doing its thing.) In the other cases you get back to the menu automatically.

Lines 63075–63455. These lines cover the subroutines for renumbering statements, including substituting the proper new numbers in GOTOs, GOSUBs, etc. Each new statement to be processed is picked up in line 63255. Each case requiring a new number is identified in lines 63265–63300. Other cases might be added (e.g., LIST). The processed statement is transferred to disk in line 63305. Setting up to do the renumbering is done in lines 63320–63455.

Lines 63465–63515. Here is where specified statements are captured as

Listing continued.

```
63530  CLOSE:PRINT:PRINT"Renumbered Routine is TEXT File with Pathname ";wo
       rk$
63535  PRINT:PRINT"Do you want to SAVE this Routine with a DIFFERENT Pathna
       me (Y/N)";:INPUT q$
63540  IF q$="y" THEN 63555
63545  PRINT:PRINT"Do you want to LOAD "work$" (Y/N)";:INPUT q$
63550  IF q$="y" THEN 63575:ELSE 63620
63555  PRINT:PRINT"Key NEW Pathname for ";work$;:INPUT newname$:ON ERR GOTO
        63570
63560  RENAME work$,newname$:OFF ERR
63565  SWAP work$,newname$:GOTO 63545
63570  PRINT:PRINT"   <<<  DUPLICATE or BAD Pathname - REPEAT  >>>":GOTO 63
       555
63575  ON jmp GOTO 63605,63580
63580  PRINT:PRINT"Do you want to DELETE the Captured Lines from Memory FIR
       ST (Y/N)";:INPUT q$
63585  IF q$="n" THEN 63605
63590  PRINT:PRINT"EXECUTE the Following TWO Statements in IMMEDIATE Mode"
63595  PRINT:PRINT;t$:PRINT" goto 63605":PRINT CHR$(11);CHR$(11);CHR$(11);:
       END
63600  DEL 1 TO 62999:REM Executed in IMMEDIATE Mode
63605  EXEC work$:END
63610  :
63615  REM  ***  DISPLAY MENU  ***
63620  HOME:VPOS=5:PRINT; TAB(15);"PATCH MENU"
63625  PRINT:PRINT"1. LIST ROUTINE (Unedited)"
63630  PRINT:PRINT"2. LIST ROUTINE (Edited)"
63635  PRINT:PRINT"3. CAPTURE (Save) ROUTINE as TEXT File"
63640  PRINT:PRINT"4. EXTRACT, RENUMBER, REINSERT ROUTINE"
63645  PRINT:PRINT"5. QUIT"
63650  PRINT:PRINT; TAB(10);"< MAKE SELECTION >";:INPUT s
63655  IF s=<0 THEN 63620
63660  PRINT:PRINT"Press <RETURN> to do TASK #"s"";:INPUT q$
63665  IF ASC(q$)<>-1 THEN 63620
63670  IF s=>5 THEN PRINT"BYE!":END
63675  ON s GOTO 63690,63745,63835,63320
63680  :
63685  REM  ***  LIST ROUTINES  ***
63690  PRINT:PRINT"ADJUST Printer and hit <RETURN>";:INPUT q$
63695  PRINT:PRINT"List WHOLE Program (Y/N)";:INPUT q$
63700  OPEN#3,".printer":IF q$="n" THEN 63710
63705  OUTPUT#3:LIST 1-62999:OUTPUT#0:GOTO 63620
63710  s$=" 63725  OUTPUT#3:LIST ":t$="":GOSUB 63860
63715  PRINT:PRINT"EXECUTE the Following TWO Statements in IMMEDIATE Mode"
63720  PRINT:PRINT;s$:PRINT" goto 63725":PRINT CHR$(11);CHR$(11);CHR$(11);:
63725  OUTPUT#3:LIST 1 TO 63999:REM Executed in IMMEDIATE Mode
63730  OUTPUT#0:CLOSE#3:GOTO 63620
63735  :
63740  REM  ***  EDITED LISTINGS  ***
63745  PRINT:PRINT"ADJUST Printer and hit <RETURN>";:INPUT q$
63750  OPEN#3,".printer":page=1:x=2:GOTO 63470:REM Go CAPTURE Routine
63755  OPEN#8 AS INPUT,".d2/captureworkfile":ff$=CHR$(12):line=0:bb=0:sp$=C
       HR$(32):bl$=sp$
63760  lf$=CHR$(10):cr$=CHR$(13):ON EOF#8 PRINT#3;lf$;cr$;:CLOSE:DELETE n$:
       GOTO 63620
63765  PRINT#3;cr$;bl$;" Page "page"";cr$;lf$;lf$;:page=page+1:line=line+2:
       bl$=sp$
63770  INPUT#8;string$:aa=INSTR(string$," ",2)+1
63775  IF LEN(string$)>75 THEN 63795
63780  PRINT#3;cr$;lf$; SPC(bb+1);string$;:IF line<54 THEN 63790
63785  line=0:bl$=ff$:bb=0:GOTO 63765
63790  line=line+1:bb=0:GOTO 63770
63795  FOR i=76 TO 65 STEP-1
63800    a$=MID$(string$,i,1):IF a$<>" " THEN 63810
63805    NEXT i
63810  PRINT#3;cr$;lf$; SPC(bb+1);LEFT$(string$,i);:line=line+1:bb=aa
63815  IF LEN(string$)-i=<0 THEN 63770
63820  string$=RIGHT$(string$,LEN(string$)-i):GOTO 63775
63825  :
63830  REM   ***  SAVE ROUTINE AS TEXT FILE  ***
63835  x=3:GOTO 63470:REM Go CAPTURE Routine
63840  PRINT:INPUT"Key PATHNAME for the ROUTINE";newname$:ON ERR GOTO 63570
        63845  RENAME n$,newname$:OFF ERR:GOTO 63620
63850  :
63855  REM   ***  LIST/DEL RANGE LIMITS  ***
63860  PRINT:PRINT"Key FIRST Line #  ";:INPUT l1
63865  PRINT"Key SECOND Line # ";:INPUT l2
63870  IF l1>l2 OR l1<0 OR l2>63999 THEN PRINT:PRINT"   <<<  ILLEGAL LINE #
       'S  -  REPEAT  >>>":GOTO 63860
63875  s$=s$+CONV$(l1)+" to "+CONV$(l2)+":":t$=t$+CONV$(l1)+" to "+CONV$(l2
       )+":"
63880  PRINT"Any more Sections to Pull Out (Y/N)";:INPUT q$
63885  IF q$="y" THEN s$=s$+"list ":t$=t$+"del ":GOTO 63860
63890  RETURN
```

a text file. Line 63505 is created in line 63495 by printing it followed by a GOTO (to get back into PATCH again). You are instructed to execute the two lines from the console by running the retype key (right arrow) over them individually, hitting return each time.

Lines 63525-63605. Here are some miscellaneous subroutines used at the end of a renumbering exercise. Particularly note that line 63600, which deletes the lines extracted from the resident program, is set up on the console for execution in Immediate mode (as with line 63505, and again in line 63725).

Lines 63685-End. The listing options are managed here. The width of a printed line is controlled in line 63775; control of lines per page is in line 63780.

For accommodating smaller memories there are numerous ways to reduce the size of PATCH, an exercise I will leave to the interested programmer. Other functions might be added to this utility, such as compressing the number of line numbers or copying individual files from one disk to another.■

Reborn Apple Designed For Business

Apple III computer system *CPU:* 6502A processor with 128K, expandable to 256K; 4K ROM for self-test. *CRT:* Several monitors available; Monitor III provides 80-col, 24 line B/W display. *Disks:* One 5.25", 140K floppy disk is built-in. *Other Eqpt:* Built-in audio speaker; ports for communications, printer, joysticks. *Software Included:* Operating system (SOS), Basic, Pascal. *Price:* $4690. *Supplier:* Apple Computer, 10260 Bandley Dr., Cupertino, CA 95014, (408) 996-1010.

Donald Campbell and Robert Sepe are both active in computer education at the University of Alabama.

Introduction of the Apple III computer into the marketplace has not been a smooth process. The system was first offered to the public and then there was some hesitation about its distribution. But today Apple III seems well-established as a product.

One way to explain how the Apple III fits in—as a tool for solving business problems—is to compare it to its widely-used sister product, the Apple II+.

The principal difference between the Apple III and the Apple II+ can be stated quite simply: the II+ is a small computer suitable for the huge educational and hobby markets as well as for some small businesses; the III is a more powerful and efficient computer designed specifically for use in automated offices. The Apple III has the capacity to run more powerful and efficient software. It can accommodate more user-friendly word processing packages, more sophisticated accounting packages, more powerful graphics, and more flexible electronic spread sheets than the Apple II+.

When you purchase a stereo, you usually buy component parts to build a system. The Apple II+ and III are marketed in the same manner—you must configure a usable system by buying components. The Apple II+ keyboard and computer are housed in a single unit. In addition to the keyboard and computer, one disk drive is housed in the basic Apple III unit. To run any practical business applications beyond the most elementary with either computer, you must also purchase a video monitor, external memory storage devices such as disk drives, and a printer.

Like the II+, the III is a single-user, interactive, stand-alone machine. This means that only one person at a time can operate the computer, and that when information is keyed into the machine it is immediately stored or processed.

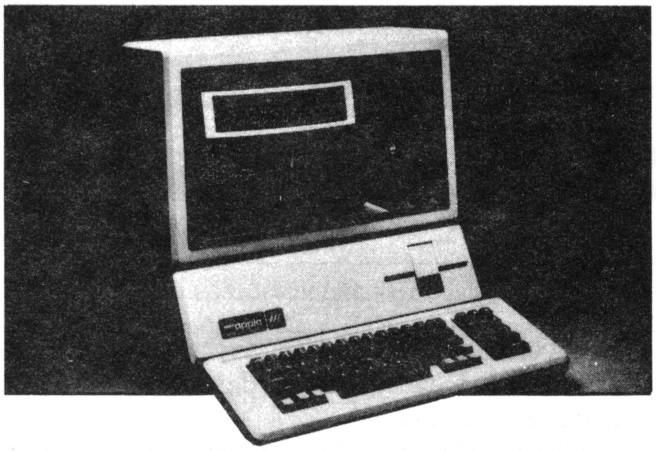
Keyboard differences

Apple III data entry is through a keyboard, and that is where the first obvious difference between the two machines lies. The Apple II+ keyboard has but 52 keys. This Spartan keyboard can lead to problems in many business applications. For example, it is especially difficult to develop user-friendly word processing packages because there are not enough "natural" keys to assign to all the manipulations required in comprehensive word processing programs. Since good word processing programs require more commands than there are keys, many keys must be assigned multiple uses. Another disadvantage for business applications is the lack of a separate Apple II keypad for high-speed input of numeric data. However, a separate numeric keypad can be purchased and plugged directly into the machine to overcome this problem.

In contrast, the Apple III keyboard has 61 keys and a built-in 13-key numeric pad. All character-keys automatically repeat at a rate of 33 characters per second by merely maintaining key pressure. Five special function keys include ESCAPE, CONTROL, ALPHA LOCK,

In Our Opinion

Compared to the Apple II+, the Apple III seems better designed for business uses. Software is scarce, but with emulation the III will run Apple II programs. The Apple III offers more built-in I/O capabilities, but expansion possibilities are greater for the II+.


Apple III system with Monitor III display mounted on CPU-disk unit.

and two programmable keys. Keys that can be programmed by the user increase flexibility for unique situations. The Apple III type-ahead buffer permits a competent typist to enter characters at optimum rates.

Both machines use 8-bit microprocessors. Many computer manufacturers are now marketing 16-bit machines, which are able to use more sophisticated programs and operate on larger data bases. The problem is that there just isn't enough 16-bit software available to take full advantage of this increased power and sophistication. In our opinion, business people should not opt for a 16-bit machine unless they can find suitable software to solve the problems for which the machine is being purchased. We believe that it will be 18 months to two years before programs which take full advantage of 16-bit capabilities are in adequate supply.

Microcomputers are becoming so inexpensive that they can almost be viewed as "throw-away" items. If the proper groundwork is done before purchasing an 8-bit machine, the cost can be recouped in a very short time. Developments in the field are coming along so rapidly that it does not appear feasible to us to wait for new technology. No matter what you buy, something will be in the pipeline which will make your computer out of date. If you can afford to wait for new technology, perhaps you don't really need a microcomputer at this time.

The Apple II+ operating system is built into the machine's Read Only Memory (ROM) and is available to the machine at the moment power is applied. However, the limitation to a ROM-resident operating system is that it cannot be changed or modified; it is forever! For example, the character set (alphanumeric and special symbols appearing on the video display) is stored in ROM and cannot be changed.

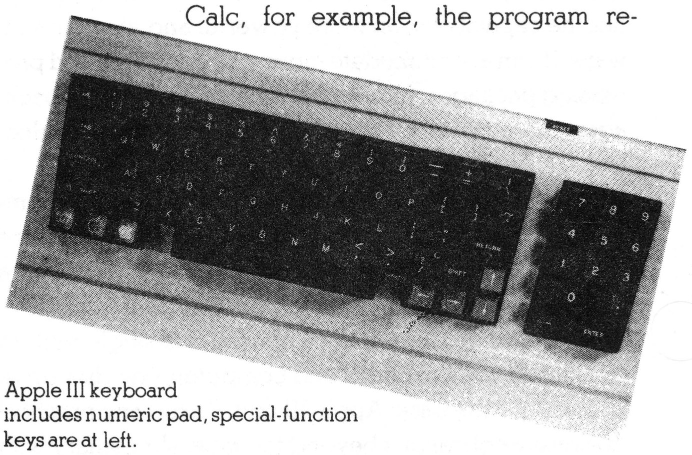
In contrast, the Apple III operating systems, programming languages, and character sets all consist of software that is loaded into the machine from disks. Improved business software or customized operating systems can be used with the Apple III as needed, or as they become available. This flexibility permits you to expand the system as your business grows. Suppose, for example, that you are working on an application where the standard typewriting font is not acceptable. The Apple III permits you to load Roman, Apple, or Byte fonts and to create other fonts such as Greek, Arabic, or Cuneiform.

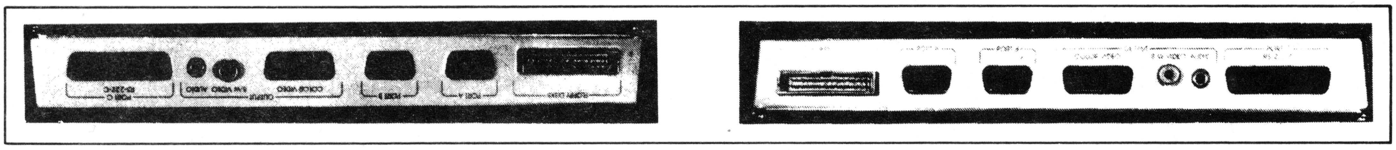
In the text mode, the Apple II+ offers a 40-column by 24-line white on black, or black on white video display. This is inconvenient for word processing and some long program statements, but is easy on the eyes. To achieve 80-character and upper/lower case display, special attachments must be purchased.

The Apple III offers an 80-column by 24-line normal or inverse video display which is adequate for word processing applications and for viewing long programming statements. The character set can be changed under program control at any time. Like the II+, the III also offers up to 16 colors when color video monitor is used.

More memory

An Apple III with maximum 256K processor main memory offers significant advantages over the 48K memory of the II+: A VisiCalc application is one example of the value of the III's greater memory. The Apple III version of VisiCalc will allow you to build a file of up to 192K as opposed to an 18K file on the Apple II+ version of VisiCalc. The 80-column video display on the III will enable you to display eight columns of nine characters on the screen; only four columns are possible on the Apple II. Larger main memory makes it possible to perform work more quickly and to handle data in larger chunks. It must be remembered that part of the computer's memory is used to store the program being used. With VisiCalc, for example, the program re-


Apple III keyboard
includes numeric pad, special-function keys are at left.


Apple III I/O ports; those labeled A and B are for printer, joystick and similar equipment.

quires about 30K of memory, leaving only 18K for data handling on the Apple II+.

Both the II+ and III use random access 5.25" floppy disk drives, with 140KB per drive. Six disk drives may be added to the Apple II+, for a total of 840KB of file storage. The III will support four disk drives, for a maximum capacity of 560KB.

However, the Apple III also has an Apple-supplied hard disk capability, currently up to 5MB, which should be sufficient to handle many business applications. A number of reputable manufacturers offer Apple II+ and III compatible hard disk equipment which allows users to expand file storage to 20MB or more.

If the standard font is not acceptable you can load Roman, Apple, or Byte fonts, and create others such as Greek, Arabic, and Cuneiform

Better Basic

Floating point Basic accompanies both the II+ and the III. This means that the Basic programming language is capable of executing decimal arithmetic, which is essential in many business applications. The II+ version of the language, Applesoft, is installed in ROM. The III's version, Apple Business Basic, is loaded into the machine's working memory from a disk furnished with the computer. Business Basic is more powerful than Applesoft, enabling software developers to use programming techniques unavailable to Apple II+ programmers. For example, Business Basic enables the programmer to use IMAGE and PRINT USING statements which make formatting and filling-in business reports relatively easy.

The III's standard 128K main memory will support Pascal with no additional hardware purchases, and its 80-character display is well suited for the long program statements commonly used by Pascal programmers. Fortran and Pascal are available with the II+, but additional hardware to expand internal memory of 16K must be purchased before these languages can be used. Unsubstantiated rumors have circulated that an Apple III version of Fortran is forthcoming.

The Apple II+ has an analog-to-digital port capable of supporting devices such as a game controller or a 13-key numeric keypad. Also built-in is a cassette tape recorder interface. However, cassette tape is cumbersome, slow, and too unreliable for use in business settings. A video port is used for CRT display. Additional ports require purchase and installation of an expansion card in one of the machine's eight expansion slots.

The Apple III has significantly more built-in peripheral capability than the II+. For example, if you wish to use a disk drive with your II+, you must purchase and install a disk controller card. The III has a built-in controller which will support four disk drives. The III also has an RS-232 input/output port, which enables the user to plug in any RS-232 compatible serial printer or modem without the purchase of additional hardware. Two ports are available to support peripherals, such as a joystick or an Apple Silentype printer. Other peripheral ports provide for color video, black and white video, and audio. To connect these peripherals to the Apple II+ it is necessary to remove the cover and install a firmware card. Peripheral connections on the Apple III are made using external sockets at the back of the machine, and four additional expansion slots are available inside the Apple III cabinet.

The user-definable expansion ports in the II+ and III are a most important feature. Independent electronic manufacturers market a wide variety of peripheral devices which can be purchased and connected via these ports to meet individual requirements. For example, one of the Apple II+ slots may be used to support a 16-bit CPU card. The availability of expansion slots significantly enhances machine flexibility and enables the user to add enhancements as they are introduced.

There seems to be more Apple-compatible software available than for any other machine presently marketed. The Apple III will run most software programs written for the Apple II+. An emulator program, furnished on a 5.25" floppy disk with each Apple III, enables the III to "think and act" like a II+. This is an important advantage; it means that the thousands of programs already written for the Apple II+ can be used with the Apple III. (See Table on page 51)

Continued from page 46

Reborn Apple Design

Apple III vs Apple II+ At A Glance

	APPLE II+	**APPLE III**
Cost	$1,530*	$4,690**
Main memory	16 to 64K	128 to 256K
Video Display	40 characters	80 characters
Number of Keys	52	61
Numeric Keypad	No	13 Key
Programmable Keys	None	2
Automatic Repeat	Limited	All Keys
Type-ahead Buffer	No	Yes
Built-in I/O ports	1	5
I/O expansion slots	8	4
Floppy drives: Built-in and Add-on	0 and 6	1 and 3
Max floppy storage	840 K	560K
Programming Languages	Basic	Basic/Pascal
Software Availability	Abundant	Moderate

*Apple II+ with 48K memory, no disk drives, no video monitor, no printer, no other languages.

**Apple III with 128K memory, one built-in disk drive, a four disk drive controller, two analog-to- digital interfaces, one color video interface, serial interface, no video monitor, no printer.

APPLE /// INVOKABLE MODULES

by Alan Anderson

In the last issue we talked about how SOS calls are put into Assembly language programs, and how to link those Assembly language programs to Pascal programs. This issue, by popular demand, we'll reveal how to go about writing invokable modules to enhance the power of Business BASIC. The things mentioned at the end of last issue will appear next time (unless, of course, popular demand again dictates otherwise).

But First, a Word from our Sponsor . . .

Great news for all us Apple /// programming hackers. The new edition of the Standard Device Drivers Manual is out, and it's superb. It includes in-depth (and I mean in-depth) discussions of what each of the standard device drivers can do. The list of standard drivers includes .CONSOLE, .GRAFIX, .PRINTER, .RS232, and .AUDIO. The power in these drivers, particularly .CONSOLE and GRAFIX, is incredible. The information included in this manual is extremely complete. We'll talk about some of the secrets it reveals later in this article.

The PERFORMing Arts

The ink was hardly dry on the preliminary Business BASIC Reference Manual (yep, they've upgraded that one, too) before the good old Apple II crew noticed that horrible oversight in the new BASIC: they left some things out, specifically PEEK POKE, and CALL. To Apple II programmers, this seemed like something very close to betrayal. The PEEK, POKE, and CALL statements meant unlimited expandability for Applesoft and Integer BASIC. Anything that couldn't be done in the high-level language could be written in Assembly language and linked to the main program with these statements. Their absence from Business BASIC seemed to mean that the language was forever locked up.

Of course, that was not the case. Upon closer examination, the new BASIC had a couple of unfamiliar new statements: INVOKE and PERFORM, and the manual said something about them allowing the use of Assembly language with your BASIC programs. Hurray! But why were good old PEEK, POKE, and CALL replaced? Are INVOKE and PERFORM as powerful? How do we write assembly language programs which can be invoked, known as invokable modules?

The reason why the old Applesoft statements were replaced should be fairly obvious to anyone who read last issue's article or who is familiar with the memory organization of the Apple ///. As stated last time, the operating system (SOS) handles the odious task of loading things into memory and remembering where they are, thus freeing the programmer from such time-consuming housekeeping chores. This makes the Apple ///'s huge memory space (128K RAM standard, expandable to 256K) into a virtual memory workspace. Whenever memory is needed for a buffer, a program, some data, or anything else, SOS automatically finds memory and allocates it. A consequence of this system, as talked about last issue, is that the programmer does not know where in memory items will be from one application to the next, so other methods of finding things must be used.

Last time we talked about the SOS call mechanism, which allows the programmer to call the operating system's routines without knowing where they reside. INVOKE and PERFORM provide a similar capability to the Business BASIC programmer. INVOKE loads one or more Assembly language routines from disk, while PERFORM executes the routines. Notice that the BASIC programmer never needs to worry about where the invokable module is when loaded into memory! This is why we don't have to worry about PEEK, POKE, and CALL. Everything that those three old friends did for us on the Apple II can be accomplished on the /// with INVOKE and PERFORM, and without the pain and hassle of wondering whether we're poking the right place, or if our recently BLOADed assembly language program has accidentally been loaded on top of DOS.

The INVOKE statement replaced BLOAD and some POKEs. It automatically finds a place in memory for the desired routines, loads them and relocates them. The PERFORM statement replaces CALL, and implicitly replaces PEEK and POKE as well. It not only finds and executes the desired routine (as CALL does in the Apple II BASICs) but it also allows the programmer to pass parameters to the routine (replacing one use of POKE) and receive parameters from the routine (replacing a common use of PEEK). Also, since the programmer doesn't know where his routine has been loaded, the routine is called by name instead of by address.

The theory behind the creation of INVOKE and PERFORM is unique for a personal computer BASIC, but is by no means new. In fact, Apple borrowed it directly from another system: Apple II Pascal. Those of you familiar with that operating system, which was derived from the UCSD Pascal Operating System, know that in Apple Pascal, know that Assembly language routines can be linked to Pascal programs, that the Assembly language is loaded and relocated by the operating system automatically, and that the Assembly routines are called by name since their addresses are not known. Parameters can be passed between the Pascal program and the Assembly language routine, and the whole business uses a much nicer and cleaner interface than PEEKing, POKEing, and CALLing—ask any Pascal programmer.

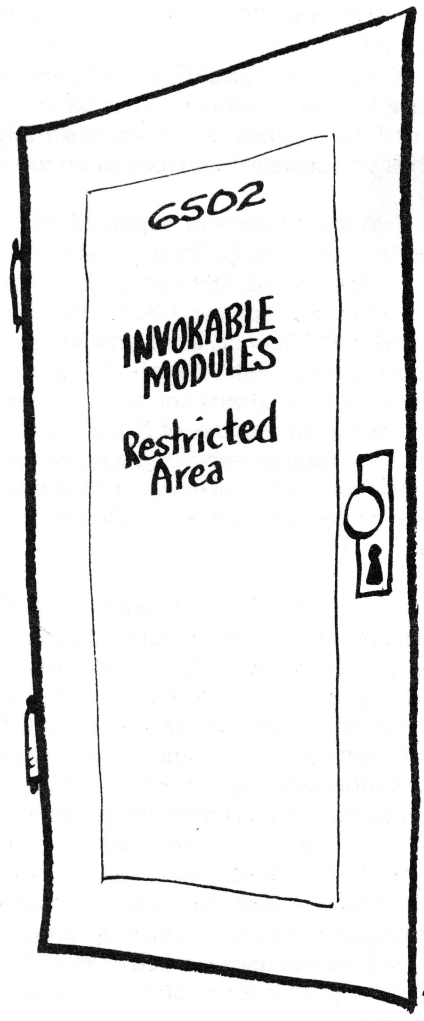
If you read last issue's installment, you may remember that that article involved interfacing an Assembly language routine to an Apple /// Pascal program. The procedure and techniques we used were exactly the same as those used in Apple II Pascal. Those of you out there who are familiar with how Apple Pascal links up with Assembly language will see that the way invokable modules are created and used directly parallels the Apple Pascal technique. Therefore, I would highly recommend the Apple II Pascal Operating System Reference Manual chapter on The Assembler for some background material on writing Business BASIC ivokable modules.

Getting Down to Details

Alright already, enough discussion and reference. How di I do it? Glad I asked. Well, here goes.

Assembly language routines which can be used from Business BASIC are written with the Apple /// Pascal system. In general, the code is created in the Editor, assembled with (surprise!) the Assembler, saved to disk, then used from the Basic program.

Here's a nice bonus effect of having both Pascal and BASIC served by the same operating system and disk format: the same Assembly language routine can often be used by both BASIC and Pascal with little or no modification. As a demonstration of that claim, our first BASIC invokable module will use exactly the same Assembly language file as was used last time with Pascal, which prepares the system for a cold start. If you didn't enter the file last time, have since erased it, or (shame on you!) don't have the previous issue, I'll repeat it here, with instructions. (Back issues of the March-April 1982 issue of Apple Orchard are available.)



First, enter the Pascal Editor, and type in this file: (some of the comments have been omitted this time)

```
                .PROC RESTART
COLD_START   .EQU 65                    ;COLD_START's
$65                                     number is
                BRK                     ;To signal a SOS
                                        call
                .BYTE COLD_START        ;Tells which call
                .WORD PARAM_TABLE ;Points to the
parameters

PARAM_TABLE.BYTE 00                     ;COLD_START
params                                  has no
                .END
                ;see Mar-Apr Apple
                Orchard for details
```

Then, leave the Editor and save the file by typing "QW.D2/RESTART", substituting a different disk drive name for .D2 if you wish. Then press "A" to assemble the file. Answer ".D2/RESTART" to the "Assemble what text?" question, using something other than .D2 if you didn't save to Drive 2 from the Editor. When asked "To what code file?", answer by typing a dollar sign. This will cause the output file to be called RESTART.CODE on the same disk as the source file. Press [RETURN| when asked for an output file name.

Once the assembly has been finished, you should have a real, live, bona fide invokable module!. To prove it, boot your Business BASOC 1.1 diskette, then put the diskette with RESTART.CODE on it in the built-in disk drive. Type "INVOKE .D1/RESTART.CODE" to load the module from disk. Then type "PERFORM RESTART" and watch the fun. Your screen should say "INSERT SYSTEM DISKETTE AND REBOOT" in 40-column text mode. You've just written and executed an invokable module! That wansn't so bad, was it?

Now that you've got the basics of writing invokable modules, you should be able to handle the whole story, which appears in (ta-daa!) Apple's new technical note, Apple /// Business BASIC and its Assembly Language Interface. This note should be available from the IAC soon, or write to the author (that's me) in care of Apple Orchard for more information on writing invokables or unlocking any of the other Apple /// mysteries.

For further information: be sure and read (and reread) your Standard Device Drivers Manual. It's got great gobs of goodies in there, and it will likely answer some of your favorite technical questions about the Apple ///. Also, the Apple /// Pascal manuals are worht a perusal or two.

And now, folks, it time to practice your invocations!

Apple Orchard March/April 1982

UNLOCKING APPLE ///

by Alan Anderson

What's so special about the Apple ///? Why does it have a Sophisticated Operating System? Why is Apple always talking about SOS being such an important tool? This article will address the reasons why the Apple /// and its operating system are important to the programmer and also demonstrate some of the fabulous things you can do with it.

OUR APPLE II HERITAGE

Anyone who has spent more than a few hours programming the Apple II is familiar with, to some extent, having to interface with the machine at its own level. In BASIC, this usually takes the form of POKE, PEEK, and CALL statements. Many Pascal programmers are familiar with the free union variant, or "tricky" record, used to access specific memory locations. Of course, the assembly language programmer deals with the machine level constantly.

Moving over to the Apple /// presents some fascinating new problems which the programmer must deal with. For example the Apple /// gives you: memory which can be bank-switched, thus leaving your program in the Twilight Zone; a virtual memory system, in which device drivers may be loaded almost anywhere; a zero page and a stack which can move around in memory; a microprocessor which can run at two different speeds; memory which can be write-protected; a character set which is defined in RAM and can be changed on the fly; and the ability to handle lots of interrupt driven devices. With all of this stuff going on, the old method of going straight to the hardware just doesn't cut it any more. In order to control all of these features and to give the programmer an interface through which he can gain access to the system's resources, SOS was born.

Simply stated, SOS is the program which acts as a sort of clearinghouse or central command post for everything that happens in the Apple ///. The parts of SOS are appropriately named "managers," since they manage and control the system.

THE PIECES OF A PROGRAM

Any bootable diskette on the Apple /// consists of at least three files. They are named SOS.KERNEL, SOS.DRIVER, and SOS.INTERP. SOS.KERNEL is the operating system as supplied by Apple. We'll discuss it in more detail shortly. SOS.DRIVER is the group of device drivers used to control various Apple /// devices, such as the keyboard, printer, speaker, ProFile hard disk drive, etc. SOS.DRIVER files are usually created by the System Configuration Program on the System Utilities diskette. SOS.INTERP is a machine-language program such as Business BASIC, VisiCalc, or Apple Writer ///. A diskette which does not contain files with these names will not boot, and an appropriate error message will appear if you try to boot it.

One of the definitions of "kernel" is "the nucleus, essence, or core." From this we get SOS.KERNEL, the nucleus of the operating system. As mentioned earlier, the kernel consists of parts called mangers which control and serve various aspects of the system. The main parts are: the Memory Manager, which allocates and keeps track of usage of the system's main memory; the Device Manager, which handles communication between the system's various devices through the device drivers; the File Manager, which works with the Device Manager to handle the flow of information between the system and its physical devices; and the Utility Manager, which handles certain miscellaneous tasks, such as reading the joysticks and preparing the system for a cold start by clearing memory.

What makes the Apple /// so special from a programmer's point of view is the job SOS does of managing the system. In fact, the programmer rarely has to worry about switching memory banks, where the stack is, where the device drivers are, or, in most cases, even where his program is located in that vast expanse of memory.

Some of the more hardened assembly language veterans among you must be simmering in your skepticism by now. How, you say, can the programmer take advantage of calling the operating system if he doesn't know where anything is? The answer is simple: let SOS find it. When a program wishes to call SOS, it isn't done with the traditional JSR. In fact, since you don't know where the operating system is, it's impossible to JSR to a fixed location. Instead, the Apple /// takes advantage of the 6502's software interrupt capability, better known as the BRK or Break instruction. When this instruction is executed, the Apple jumps to a memory location pointed at by two bytes in the high part of memory. SOS then takes over and executes the routine, returning control when it's done. How does this look in practice? Read on.

A call to SOS actually consists of three items. The first, already mentioned, is a BRK instruction. Following tht comes a SOS call number. Each user-accessible subroutine (or SOS call) within the kernel is assigned a one-byte identifying number. The third item in a SOS call is a two-byte pointer to a table of parameters. Each call has certain parameters associated with it. By passing SOS a pointer to these parameters instead of the parameters themselves, the programmer can use the same table more than once. In addition, the numbers of parameters can be changed in future versions of the operating system with minimized effect on the user's programs. The first byte in the parameter table tells how many parameters are in the table. SOS compares this to the number of parameters it is expecting for that call, using this byte for error checking. In using the Apple /// Assembler (part of the Apple /// Pascal system) a typical SOS call would look like this:

```
BRK
.BYTE <call_number>
.WORD <param_table>
```

The .BYTE pseudo-op generates a byte of the value of <call_number>. The .WORD pseudo-op generates a two-byte pointer to <param_table>.

Each of the four managers listed above contains certain SOS calls. For example, the File Manager includes calls to create, destroy, rename, open, close, read, and write files, among other things; the Device Manager has calls to find, request, and release chunks of memory, as well as other functions. Each SOS call has a unique set of parameters associated with it.

After a SOS call, SOS returns an error code in the 6502's accumulator. If no error occurred, a $00 is returned.

Because of the flexibility of the SOS call mechanism, the assembly language programmer can think of SOS calls as extensions to the 6502's instruction set. The SOS call system does a superb job of uncomplicating sophisticated programming at the mahcine level.

CALLING ALL SOS

Let's go through a hand-on example of a SOS call and how it would be generated. For demonstration, we'll use the simplest of all SOS calls, one called COLD_START. If you have the Apple /// Pascal system, you can try this example on your Apple. This call, a part of the Utility Manager, is used to provide a neat, clean exit from a program. It clears out memory, displays "INSERT SYSTEM DISKETTE AND REBOOT" on the screen and waits for a CONTROL-RESET to be pressed. You may have noticed that certain applications, such as VisiCalc /// and Apple Writer ///, use this method to exit. Here's how it's done in assembly language:

```
BRK                          ;To signal a SOS call
.BYTE COLD_START             ;Tells which call
.WORD PARAM_TABLE            ;Points to the parameters
```

Now we have to define our labels. First, COLD_START is defined:

```
COLD_START    .EQU 65    ;COLD_START's number is $65
```

Now the parameter table for COLD_START is defined. COLD_START's parameter table is uniquely simple:

```
PARAM_TABLE   .BYTE 00   ;COLD_START has no params
```

That's it! Now, let's make the program work by assembling it an using it. First, here is a complete listing of the whole file to be assembled. This should be created in the Apple Pascal editor.

```
              .PROC RESTART        ;Standard heading for
                                   assembly language
                                   (see Apple /// Pascal
                                   Program Preparation
                                   Tools manual, chapter
                                   2)
COLD_START    .EQU 65              ;COLD_START's
                                   number is $65
              BRK                  ;To signal a SOS call
              .BYTE COLD_START     ;Tells which call
              .WORD PARAM_TABLE    ;Points to the
                                   parameters
PARAM_TABLE   .BYTE 00             ;COLD_START has no
                                   params
              .END                 ;Tell the assembler
                                   we're done
```

Once you have entered this file in the editor, quit the editor and save the file on disk, then assemble it and save it into a file called RESTART.CODE. Return to the editor and enter the following Pascal program.

```
program example;
procedure restart; external; (*Tells Pascal that restart is an
                                     assembly language procedure*)
begin
  restart;
end.
```

Exit the editor, save the file, compile it, and save it to a file named EXAMPLE.CODE.

Now enter the Linker. When the Linker asks for your Host file, use EXAMPLE; for the Lib file, use RESTART, which contains the Assembly language program. Press <RETURN> when asked for another Lib file and when asked for a Map file. For your Output file, answer REBOOT.

Finally, when the Linker is done, you should have an executable file (REBOOT). So execute it! Voila—if you've done everything correctly, you should get the "INSERT SYSTEM DISKETTE AND REBOOT" message. What a thrill, eh? If you don't get the message, double check your work. If you don't yet feel comfortable with the Apple Pascal system, you should probably spend some time learning it if you're going to use Assembly language on the ///, whether it links with BASIC, Pascal, or stands alone.

Well, that's just great. But what if you have aspirations of doing things other than just restarting the system? All right, we'll take a look at some other slightly more sophisticated SOS calls.

DEALING IN VOLUME

One of the calls within the file manager is VOLUME. This call tells us the name of the physical volume within a given mass storage device, the total number of blocks on that device, and the current number of free blocks on the volume. If we wanted to find out this information for the built-in disk drive, the parameter list would look like this:

```
PARAM_TABLE   .BYTE 04          ;Number of parameters
                                in table
              .WORD DEVNAME     ;Pointer to the device
                                name to examine
              .WORD VOLNAME     ;SOS returns the name
                                of the volume at the
                                address pointed to by
                                this field
              .BLOCK 2          ;SOS returns the
                                number of free blocks
                                in this field
DEVNAME       .BYTE 03          ;Length of device name
              .ASCII ".D1"      ;Device name itself
VOLNAME       .BLOCK 10         ;SOS puts volume
                                name here
```

There are several important new concepts included here. First of all, note that variable length parameters like device names are not included in the main parameter list. Instead, the list contains pointers to the actual locations of these items. This furthers the philosophy of standardization: since the pointers are always two bytes long, the length of the parameter list always stays the same.

Second, notice that the device name, .D1, is preceded by a byte indicating its length, three characters. This is a standard for all pathnames you pass in all SOS calls: the name itself is preceded by a byte indicating the length of the name.

Third, notice that some parameters are supplied by the caller (us); an example of this is the device name. These are sometimes called input parameters. Other parameters are returned by SOS in places we provide; for example, the volume name. Note that we, the caller, provide the place for this parameter, but we don't fill it in—SOS does. This is known as an output parameter. There is a third type as well, the input/output parameter, in which the caller passes something to SOS, and SOS passes something different back in the same place. Input/output parameters are pretty rare in SOS calls.

Now that we've got the parameter table all set, we need to add the SOS call itself. It looks like this:

```
VOLUME          .EQU OC5               ;Define VOLUME
                BRK                    ;SOS call
                .BYTE VOLUME           ;Call number
                .WORD PARAM_TABLE      ;Pointer to params
```

This time, it would be a good idea to add some error checking. Remember, SOS returns error codes in the accumulator; $00 means no error has occurred. So all we need is something like this:

```
BNE ERROR_HANDLER ;Non-zero, an error
                    occurred
```

. . . assuming, of course, that we later write a routine with a label ERROR_HANDLER.

After executing this call, the designated areas in the parameter table would contain the number of total blocks and the number of free blocks on the volume, and the area pointed by the VOLNAME pointer in the parameter table would contain the name of the disk in the built-in drive (.D1). We won't make this into a complete procedure, but instead will leave it as the ever-popular exercise for the reader.

Two SOS calls controlled by the Device Manager are worth mentioning here. Their names are D_STATUS and D_CONTROL. These calls, known more simply as just STATUS and CONTROL, allow the programmer to change the way a driver or device does things. Many of the things we did by POKEing and PEEKing on the Apple II are accomplished on the /// with these calls. For example, you can tell the .CONSOLE driver to use a different character set by sending it a CONTROL call; you can ask the .RS232 driver what kind of handshake it's using with a STATUS call. In general, STATUS inquires about the state of a device; CONTROL causes the device to perform a certain function or to set a certain mode. Each device can handle different CONTROL and STATUS calls. If you're interested in what kind of STATUS and CONTROL calls are available for each device driver, check out the new edition of the Standard Device Drivers Manual.

The CONTROL and STATUS calls are so important that Pascal has a built-in interface to them, called UNITSTATUS, and the Business BASIC Version 1.1 diskette includes an invokable module called REQUEST which allows the BASIC programmer to use CONTROL and STATUS. Documentation for UNITSTATUS is on Pages 211-213 of the Apple /// Pascal Programmer's Manual Volume 1; the BASIC module is documented on the master disk in a file called REQUEST.DOC.

At this point some of you may be wondering what manual provides all of this great documentation. The answer is that it's a new book—so new that it's not out yet. The manual is apparently called the Apple /// SOS Reference Manual. It's mentioned several times in the new Standard Device Drivers Manual, so one may assume that it will in fact exist. As for now, Apple has been working with serious software developers to provide information they need, mostly in the way of special classes.

As for now, I'd like to begin a service of answering questions dealing with any aspect of the Apple ///. If you have questions about the /// that you'd love to get answered, try sending a letter to:

Alan Anderson
c/o Apple Orchard
910 A George St.
Santa Clara, CA 95050

Next issue: —The Big Event (SOS Event mechanism)
—Magic with the Keyboard
—answers to your questions

. . . and more. Okay, everybody, time to hit the SOS!

An Apple /// Guide for Humans

Alan Anderson

Apple's official company line about the Apple /// is that the /// is the most powerful personal computer in its class. The truth, of course, is that the Apple /// is the most powerful personal computer in its class. But if you've just bought an Apple /// for your business, how is that power translated into benefits that you can see? While it's true that the /// has something called a Sophisticated Operating System, or SOS, what does that mean in the real world?

In many ways, the Apple /// is not remarkably different from the personal computers which came before it. It still runs the traditional high-level languages, BASIC and Pascal. It still talks to you via a screen or printer, and it is still talked to with a typewriter keyboard. However, those sneaky folks at Apple hid the real power of this thing deep inside. The most significant real advantage of the Apple /// is the way the computer manages its resources, that is, the operating system. Don't discount the importance of that one point, though. It can make all the difference in the world, and that's what this article is about.

One of the single most important things that SOS does is make it easier for programmers to create more sophisticated, easier to use software. This becomes more obvious as the programmer gets closer to the computer itself (*i.e.*, Assembly language), and is less obvious in the high level languages, such as Pascal and BASIC. Since not very many folks have poked around in Assembly language on the ///, the significance of SOS in simplifying programming has not been all that well-reported.

Does SOS have any importance if you're not interested in doing any programming at all, just in using off-the-shelf software? The answer to that is an emphatic "Yes!" In addition to making life easier for programmers, SOS does a number of things to increase the applications software user's control over the system. These things fall mainly into two categories: configurability and device independence.

Now that I've thrown out a couple of buzzwords, let's work on translating them into English. The first is configurability. This simply means that you have the ability to add new devices, change others, and all the time (this is the significant part) retain compatibility with your software. What does that mean? It means that, in general, more software will work with more hardware. The Apple ///'s SOS lets you plug in devices like printers, mass storage devices, modems, and anything else, and handles much of the necessary translation between the computer and the external device, so the software author doesn't have to worry about it.

Does this system work? Yes, it does. As an example, consider the release last fall of the ProFile hard disk drive. As soon as this drive was released, it was instantly usable for data files by virtually all Apple /// software. While this may not seem terribly remarkable if you haven't been around this business for a long time, the fact is that the release of a new disk drive generally requires some "patches" or modifications to the applications software before it can be used. A living example of this is the IBM Personal Computer's recent introduction of larger-capacity disk drives. While the drives are there, the software can't do anything with them until it is rewritten.

The reason SOS is able to perform this feat is through the magic of complex little programs called device drivers. A device driver is a sort of translator between the device itself and the application program. As an example, the device driver for the floppy disk drives translates your request to load your budget file into VisiCalc into the proper series of head movements and motor controls feeded to retrieve the information. This means that almost any device can be connected to the Apple ///, if a device driver is written for it.

The other important end-user benefit of SOS is something called device independence. This item is related somewhat to configurability, but it moves one step further. Device independence means that all the things plugged into the Apple ///, such as keyboard, screen, disk drives, printers, modems, voice synthesizers, magnetic card readers, and anything else real or imagined, are all treated equally under one big umbrella called devices.

What does that do for you? Let's say you're about to print out your VisiCalc budget. You give VisiCalc the "**/P**" command, and it says "Print: File or Printer". Normally, you just press **P** for printer, and the report shows up at the printer, but what happens if you choose the other option, File? Since SOS treats all devices equally, you can also "print" your budget somewhere else — like to a file on disk, for example. Think about this for a minute. When you print your file, all that happens is that each character to be printed is shipped off, one by one, to the designated device. That device then processes it however it wishes. If the device is a printer, the process involves making the printer create the character on a piece of paper; if it's a disk drive, it stores that character in a disk file; if it's a modem, it sends the character across the phone line; if it's a voice synthesizer, it speaks the character; you get the idea.

The reason this becomes useful is that it allows you to create on the disk an exact "picture" of a VisiCalc report that you can then load into your favorite word processor and edit, enhance, or add a report to. It's not just VisiCalc files, of course. Almost any program which prints reports and lets you specify where the report is going can be used the same way: Quickfile and PFS, for example, can do the same thing.

The Path to Success

I've discussed the fact that the Apple ///'s SOS presents you with a world of devices: disk drives, printers, modems, and more. In order to get the most out of your system, it's important to know how to communicate with all these guys. The fundamental rule is simple: all devices have a name, and in order to cause information to come from or go to a certain device, you need only know its name. In this section, we'll talk about those names, where they come from, and how to use them.

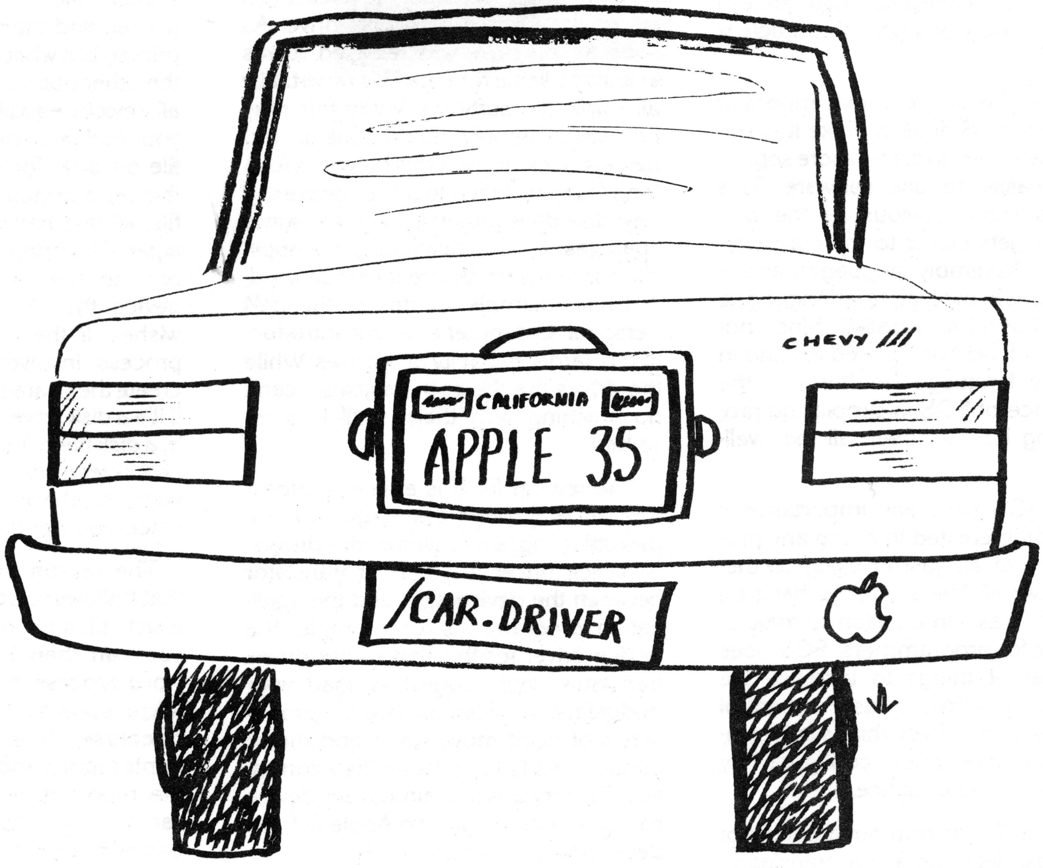
Earlier, I mentioned that every device is associated with a program called a device driver. This is the source of the device's name. The name is written into the driver and can be changed with the System Configuration Program (SCP) on the System Utilities diskette. There are a few rules for these names: they must begin with a "**.**" (called "dot" and not "period" in the jargon); the dot must be followed by one to 14 more characters (so the whole name is between two and 15 characters long); the second character must be alphabetic, and the remaining characters must be alphabetic or numeric.

The Apple /// has a few special devices, special mainly because of their importance. These include: **.D1**, **.D2**, **.D3**, and **.D4**, which are the drivers for the four possible floppy disk drives; **.CONSOLE**, which drives the keyboard and screen as one device; **.PRINTER**, which drives a printer through the built-in RS-232 interface; **.GRAFIX**, which drives the graphics screen, treated as a separate device from the .CONSOLE screen, which displays text; **.RS232**, which is the driver for a modem hooked up through the built-in RS-232 port; and **.AUDIO**, which is the driver for the speaker.

The driver for the Silentype printer is usually called **.SILENTYPE**. I say "usually" because, as noted earlier, you can change device names with the System Configuration Program. So, **.SILENTYPE** may occasionally be called **.PRINTER**, as it is in VisiCalc ///, for example. How do you know what's going on? You must rely on the documentation or your local dealer, or you must investigate with SCP to see which driver is really which. Actually, this problem isn't as bad as it sounds, since most device drivers hold on to their standard names all the time. About the only confusion that exists is with different printer drivers acquiring the .PRINTER name. If you have only one printer, you should probably name it .PRINTER, and you'll never have to worry.

Some devices, like printers, modems, and the console, either take in or put out one character at a time. These devices are called **character devices** (clever name). There is another group of devices, however, which can handle (1) large chunks of information at one time, and (2) many different groups of information, or files, at one time. These are called **block devices**, and they are, of course, disk drives. This distinction is important because, when referring to the printer, **.PRINTER** tells the data where to go. But if you want to record your VisiCalc printout on the disk, just entering **.D1** or **.PROFILE** won't do; you must add a File Name to the device name.

Let's talk about file names. Their rules are similar to device names, but without the dot: one to 15 characters long, the first character must be alphabetic, the rest must be alphabetic, numeric, or dots. So, when you want VisiCalc to print a report to a disk file, you could direct it to **.D1/REPORT**. The "slash" character separates the device name from the file name. The entire specification for a file, any file, is called a



pathname, since it specifies the path that must be taken to get to the file.

Every diskette has a name, too. The rules for diskette names are exactly the same as file names: one to 15 characters long, the first character must be alphabetic, the rest must be alphabetic, numeric, or dots. Actually, these are not called diskette names, but are more correctly called **volume names** (think of disk *volumes* as in a *library*), since the name applies to any medium in a block device, such as a hard disk drive, tape drive, or even a high-speed cassette drive. This name is assigned by the System Utilities diskette when the volume is formatted, and can be changed with the RENAME command the same way a file name is changed.

Let's review some of what we've gone over so far: every physical device has a name, like **.PRINTER**, **.PROFILE**, or **.D1**; every mass storage volume has a name; and every mass storage volume can have many files, each with its own name. If you have a diskette named "ROGER" and you place it in the built-in disk drive, do you address it as .D1 or ROGER? You can use *either* name. Note that if you address ROGER, the Apple will valiantly search for a volume called ROGER, wherever it may be. If you use .D1, the computer will go to the built-in drive, no matter what diskette is there, as long as it's a valid SOS diskette.

One phenomenon that occurs when using a hard disk drive like ProFile is that it begins to accumulate a remarkable number of files, just like you used to accumulate diskettes before you had the ProFile. Under most operating systems, having a few hundred files on a hard disk drive was a massive organizational pain. However, SOS provides for something called subdirectories, which are ways to carve your ProFile up into smaller, more manageable pieces, each piece with a name, just like any other file. Actually, subdirectories can be built on any mass storage volume, but they're most practical and important on ProFile. For example, you could have one subdirectory called **VC.FILES** for all your VisiCalc spreadsheets; one called **BG.FILES** for your Business Graphics data, and so on.

These files are addressed simply by adding the name of the subdirectory to the pathname. For example, if we wanted to store our VisiCalc report in a subdirectory called VC.FILES on the ProFile, we could use a pathname of **.PROFILE/VC.FILES/REPORT**, with the slashes separating each level of the pathname.

You can see that entering an entire pathname for a file in a subdirectory can be tedious — the above pathname, for example, is 24 characters long. Once again, SOS provides us with a tool to make it easier: the Prefix. The prefix is part of a pathname that specifies a default path for disk files. For example, we could set the prefix to read .PROFILE/VC.FILES. If we then specified file names such as REPORT, SOS would automatically add the prefix to the front of REPORT, giving the full pathname. This is especially handy when using a number of files in the same subdirectory or on the same volume.

How do you set the Prefix? Most applications programs and language systems provide a way to do it. In BASIC, the prefix is contained in a variable called PREFIX$. In Pascal and in the System Utilities program, it's set in the Filer. VisiCalc sets the prefix implicitly based on the last pathname you used. AppleWriter /// has a menu option to set the prefix.

One other point crops up when talking about addressing files in the Apple ///. What happens if the prefix is set to address the ProFile and you want to save a file onto a floppy disk? How can you cause the prefix to be temporarily ignored so that you can address a file somewhere else? The rule is this: if the pathname you enter starts with a dot or a slash, the prefix will not be used. That means that if the prefix is set for the ProFile and you want to save a file on Drive 1, all you have to do is specify **.D1/FILENAME** and the prefix will be ignored because of the leading dot. Similarly, if you wanted to save a file on the floppy disk called ROGER and the prefix was set for the ProFile, you would specify a path of /ROGER/FILENAME and again the prefix would be ignored, this time because of the leading slash.

Please note that the slash has two separate and distinct functions: one is to separate the levels in a pathname, and the other is to suppress the prefix when it is the first character in a pathname. Not realizing that this character has two meanings has caused an awful lot of confusion in the Apple /// world.

Congratulations! You now know an awful lot about how to use your Apple /// system to its fullest power. If everything didn't catch on, try re-reading this article and see if that helps. With some practice, I hope you'll see that the Apple /// and SOS provide power not just for programmers, but for regular humans, too.

Unlocking Apple /// - Part 3

Alan Anderson

Hacker's Haven

Welcome back! This issue's discussion of the ever-less mysterious Apple /// will deal with Everything You Ever Wanted to Know about Writing Assembly Language (But Couldn't Find Anybody to Ask). This time, I'll discuss the two different ways of creating Assembly language programs for the ///, and in addition, we'll get into head-spinning detail on how to use a few important SOS calls, integral knowledge for most any Assembly language application. Also, we'll create a real, working, good-for-fun interpreter!

To Boot

As you know if you've read your Apple /// manuals (and who hasn't?), in order for a SOS diskette to boot, it has to have three files: **SOS.KERNEL**, **SOS.INTERP**, and **SOS.DRIVER**. **SOS.DRIVER** files are created by the System Configuration Program, which is documented in the *Standard Device Drivers Manual*. **SOS.KERNEL** is the operating system itself, supplied by Apple Computer, and not modified unless you want to disassemble it yourself (have fun and send me the source code, please). The **SOS.INTERP** file contains a machine language "control" program; that is, when the diskette is booted, the program in **SOS.INTERP** is executed after all the operating system stuff is installed. Some examples of **SOS.INTERP** files are Business BASIC, Pascal, Apple Writer ///, and VisiCalc ///. Note, however, that no matter what the true nature of this file (*i.e.* BASIC, VisiCalc, etc.), it must be called **SOS.INTERP**. Let's summarize the SOS booting process:

1. Powering on the Apple /// or pressing **Ctrl-{RESET}** causes a jump to the computer's only ROM: a small self-diagnostic program and diskette boot routine.
2. The diskette boot routine from the ROM then reads in a small chunk of code from the diskette in the built-in disk drive. Along the way, the ROM manages to be mapped out of memory, giving the Apple nothing but wide open RAM.
3. The code read from the diskette then reads in the directory of that diskette. It looks for our friend **SOS.KERNEL** and reads it in if found or issues an error message if not found.
4. **SOS.KERNEL** then proceeds to read in and relocate **SOS.INTERP** and **SOS.DRIVER**. After doing so, it finishes up the boot process by executing **SOS.INTERP**.

So, from this little scenario, it is obvious that one way to implement Assembly language programs on the Apple /// is by making them **SOS.INTERP** files. How practical is this? Well, in the case of large, independent applications like languages, VisiCalc, and Apple Writer, this is the ideal method. You share control of the machine only with the operating system and you don't have to worry about any non-essential code hanging around. On the other hand, if you want your Assembly language programs to coexist with BASIC or Pascal, remember that there can only be one **SOS.INTERP** per diskette. That means that making your program an INTERP file is not the way to go if you want a high-level language around.

The Module Squad

Is all hope lost? Of course not! As regular readers of this series know, there is a second method for implementing Assembly language software on the Apple ///. This method, in which the programs are called modules, is used to link the Assembly language code with Pascal or BASIC programs. The Assembly language thingie called Restart which we've been playing around with in the last two installments is an example of a module. A module is simply an Assembly language program which is loaded, relocated, and generally baby-sat by Pascal or BASIC.

There are a few rules to writing modules, and just about all of them are covered in the *Apple /// Pascal Program Preparation Tools Manual* (I keep telling you to read those!). The rules are pretty much the same as those which govern the use of Assembly language routines in Apple II Pascal. In fact, many parts of SOS appear to be descended from the Apple II Pascal Operating System, so a knowledge of that system doesn't hurt when you're working with the ///.

On the other hand, making a **SOS.INTERP** has thus far been documented only in the information received in Apple's OEM class for the ///. Basically, the syntactical rules for writing interpreters are quite simple, and I'll give them to you right here.

An interpreter (which becomes a **SOS.INTERP** file) is an Assembly language codefile with a few identifying items attached to the front. Specifically, these items are:

1. The eight ASCII characters "**SOS NTRP**", which is SOS INTERP with the

vowels removed. (Note the blank between the second 'S' and the 'N'.)

2. Two bytes giving the length of an optional header information block. This block can be used for a copyright notice. The optional header block (if used) follows these two bytes.

3. Two bytes giving the loading address of the interpreter. An interpreter is not relocatable. SOS will automatically load the interpreter at the address given here. Since the interpreter is not relocatable, the source text must contain the **.ABSOLUTE** command.

4. Two bytes giving the length of the code part (everything but this header stuff). The interpreter should be constructed so that it does not use any memory beyond $B7FF.

Could You Interpret That For Me?

If you've read the previous installments of this column, you've already experienced the wondrous thrill of creating and using a module in Pascal and BASIC. Well, in this very magazine, we're going to make an interpreter. But not yet! (Awwwww.) First, we're going to delve into a few essential housekeeping calls to the operating system: SOS calls. For all my noise about SOS calls in this series, I've only documented two, and boy, have I heard it from you folks! So, let's move on into some real SOSsy stuff.

Omniam SOSam in quartes partes divisus est

There are four distinct groups of SOS calls. They are the File System calls, the Device System calls, the Memory System calls, and the Utility System calls. The file calls are probably the most commonly used. They're the ones that let you create, open, close, read from, write to, delete, rename, and otherwise manage files on devices in the system.

The device calls are related to the file calls since files are physically implemented on devices. Device calls let you modify the way the device does something, inquire about the status of devices, and do some other things.

The memory system calls allow SOS to reserve sections of memory for a program's use, and they also allow the programmer to get information about the current use of memory in the Apple.

The utility calls manage some miscellaneous resources in the Apple ///, such as the joysticks and the system date and time.

I'd like to introduce a standard format for SOS call information. To recap briefly, a SOS call is performed with an Assembly language BRK, followed by a byte indicating the call number, followed by a self-relative pointer to a parameter list. It looks like Listing 1.

This chunk of code, called the *Call Block*, is placed in your program just like any other instructions. When SOS sees the BRK it finds the parameter list and attempts to execute the call. An error code is returned in the accumulator. If no error has occurred, the accumulator contains a zero. For a list of possible errors which the calls in this article can produce, see Table 1.

The information which is essential to making SOS calls is the call number and a description of its *parameters*. Parameters come in four flavors: value, result, value/result, and pointer:

Value: Data passed to SOS from the calling program's parameter list. This data is not modified by SOS. Values are 1, 2, or 4 bytes, as specified.

Result: Data passed to the calling program's parameter list from SOS. SOS puts this data in a specified location in the parameter list. Results are 1, 2, or 4 bytes, as specified.

Value/result: Data passed to SOS from the calling program's parameter list. SOS receives this data and passes back a modified value in the same location. This is basically a value parameter and a result parameter which share the same location in the parameter list.

Pointer: a 2-byte address pointing to an area into which SOS places data (for example, in a read from a file), or from which SOS takes data (for example, when writing to a file).

The first parameter in a SOS call's parameter list is always (always, always) a value which gives the number of parameters in the list. For example, if a SOS call has 3 parameters (as does our first example below), the parameter list will begin with a byte containing a 3. In practice, it looks like Listing 2.

Table 1

Possible errors
(returned in the accumulator):

01	Bad system call number
02	Bad caller zero page
03	Bad pointer extend byte
04	Bad system call parameter count
05	System call pointer out of bounds
27	I/O error
2A	Checksum error
2B	Volume is write protected
40	Invalid pathname syntax
41	Too many open character files
42	Too many open block files, or too many block devices
43	Invalid reference number
44	Path not found
45	Volume not found
46	File not found
47	Duplicate file name
48	Not enough room on volume (disk full)
49	Directory full
4A	Incompatible file format
4B	File storage type is neither 1 nor D
4C	End of file has occurred
4D	Position out of range
4E	Access not allowed
4F	Buffer too small
50	File already open, access denied
51	Directory structure has been damaged
52	Not a SOS volume
53	Invalid value in list parameter
54	Out of memory
55	Buffer table full
56	Invalid system buffer parameter
57	Duplicate volume error
58	Not a block device
59	Bad file level
5A	Invalid bit map address

```
BRK              ;Software interrupt triggers SOS call
.BYTE Callnum         ;Each call has an i.d. number
.WORD Params         ;Each call has a parameter list
```

Listing 1

```
PARAMS      .BYTE 03           ;Three parameters to come
         (first param)
         (second param)
         (third param)
```

Listing 2

When describing a SOS call, I will give the call's number (always one hexadecimal byte) and a description of its parameters. This description will give the order of the parameters, the name and type of each one, description of its use, and any other relevant information. I will also give an example of each SOS call. This reserves my place in documentors' heaven. However, please note that my examples will not contain any error checking, so beware.

Other notes of interest: some SOS calls have parameters that are optional; that is, the call can be made with or without these parameters. In these cases the call will have two special required parameters: a pointer to the optional parameter list, and a value which tells the number of optional parameters used. You can tell SOS you have zero optional parameters, in which case the pointer to the list is ignored. If this sounds a bit confusing now, it will probably become clear when you see it used in a SOS call.

Often a SOS call parameter will be a pathname, device name, or volume name. Whenever this occurs, a standard mechanism for the name is used. The parameter list will have a pointer to the name, and the name itself will consist of a byte giving the length of the name, followed by an ASCII representation of the name itself. In source code, it looks like Listing 3.

Those are the fundamentals, so let's get right into it!

File These Away for Reference

The first group of calls I'll present come from the file system. Some file calls work on closed files and some work on open files; none work on both. The trick to making a closed file into an open file is the OPEN call; the way to make an open file closed is with the (I can hear your mind racing) CLOSE call. We'll deal with some calls for closed files first.

CREATE

This call creates a new file on a block device, *i.e.*, a disk drive. Actually, it doesn't actually work with a closed file — it makes a new one.

call number: $C0

parameters: 3

1. Pathname pointer (2 bytes). The pathname of the file to be created.
2. Optionlist pointer (2 bytes). Points to the optional parameter list, if the Length (see next parameter) is between 1 and 8; otherwise, ignored.
3. Length value (1 byte). Length of the optional parameter list. Range is 0 through 8. Meaning:
   0 No optional parameters used
   1 or 2 File type parameter used
   3 File type and Aux type parameters used
   4 . . 7 File type, Aux type, and Stor type parameters used
   8 File type, Aux type, Stor type, and Eof parameters used

Optional parameters:

File type value (1 byte)
This byte tells the file's type. Range is 0 through FF.
Meaning: (the last column shows how the file is reported by the System Utilities filer)

00	typeless or unknown file (Unknown)
01	file containing bad blocks (Badfile)
02	Pascal or Assembly code file (Codefile)
03	Pascal text file (Textfile)
04	BASIC text or Pascal ASCII file (ASCIIfile)
05	Pascal data file (Datafile)
06	General binary file (Datafile)
07	Font file (character set) (Fontfile)
08	Screen image file (Fotofile)
09	BASIC program file (Basicprog)
0A	BASIC data file (Basicdata)
0B	Reserved (WPfile) ???
0C	SOS system file (SOSfile)
0D	Reserved (Datafile)
0E	Reserved (Datafile)
0F	Directory file (Directory)
10 - FF:	Reserved (Datafile)

The file type defaults to 00 (unknown) if this optional parameter is not used.

Aux type value (2 bytes)
An auxiliary type identifier for the file. Used to store further information about the file. For example, BASIC uses this byte to store the record size of data files. Range is 0 through FFFF. The default is 0.

Stor type value (1 byte)
Indicates whether the file is a subdirectory (Stor type = D) or not (Stor type = 1). These are the only legal values. The default is 1.

Eof value (4 bytes)
Gives an amount of space in blocks to preallocate for a file. Files can grow and shrink dynamically, but if a file is known to be very large at creation type, using this parameter can help make access to it faster since the file will be contiguous. The range is 0 through FFFFFF. The default is 0.

An Example: (Listing 4)

Create a file named ASPHALT on the volume called MORK. The file will be used to contain a font.

```
                .WORD PATHNAME       ;Pointer to the name

        PATHNAME     .BYTE 09          ;Length of the name itself
                .s095ASCIIs100 "/JOE/FRED"    ;Pathname is /JOE/FRED
```

Listing 3

```
        BRK              ;SOS call
        .BYTE 0C0            ;CREATE
        .WORD CR_PARAMS        ;pointer to parameters

CR_PARAMS     .BYTE 03             ;3 parameters
        .WORD CR_PATH        ;pointer to file pathname
        .WORD CR_OPTNS       ;pointer to optional params
        .BYTE 01          ;use File_type optional param

CR_PATH      .BYTE 0D            ;length of name
        .s095ASCIIs100 "/MORK/ASPHALT" ;pathname

CR_OPTNS     .BYTE 07            ;font file
```

Listing 4

DESTROY

As long as we're creating 'em, we might as well destroy some, too. This call deletes a file from a block device.

call number: $C1

parameters: 1

1. Pathname pointer (2 bytes)
   The pathname of the file to be destroyed.

An Example (Listing 5).

Delete a file called LENDER in a subdirectory called HAPPY.TIMES on a volume named THURSDAY.

OPEN

Before we can read from or write to a file, we have to open it. This is call that performs that function.

call number: $C8

parameters: 4

1. Pathname pointer (2 bytes)
   The pathname of the file to be opened.
2. Refnum result (1 byte)
   When a file is opened, SOS assigns it a reference number (refnum). This number is then used in subsequent reads and writes with that file.
3. Optionlist pointer (2 bytes)
   Points to the optional parameters list, if the Length (see next parameter) is between 1 and 3; otherwise, ignored.
4. Length Value (1 byte)
   Length of optional parameter list.
   Meaning:

   0 No optional parameters used
   1 . . 3 Req access parameter used

Optional Parameters:

Req access value (1 byte)
   Allows the file to be opened only for reading or only for writing. Range is 0 through 3.

Meaning:

   00 open for as much access as permitted
   01 open for reading only
   02 open for writing only
   03 open for reading and writing

The access defaults to 0 (open for as much access as permitted) if this optional parameter is not used.

An Example (Listing 6).

Open the file we created earlier (/MORK/ASPHALT).

After this file is opened, we would use the result returned at location OPEN-REF to refer to this file in read and write calls (read on!).

WRITE

This is the call you use to transfer information from a buffer to a file.

call number: $CB

parameters: 3

1. Refnum value (1 byte)
   The Refnum assigned to the file when it was opened.
2. Buf pointer (2 bytes)
   Points to a buffer area where the information to be sent comes from.
3. Bytes value (2 bytes)
   The number of bytes to be written.

An Example (Listing 7).

Write 10 bytes to the file we opened earlier (/MORK/ASPHALT).

Executing this call after using the preceding OPEN to open the file and get the Refnum would cause the 10 bytes listed above to be written to the file. Remember that when we created this file, we gave it a 'Type' parameter indicating that it was to contain a font. However, when dealing with files at the SOS call level, SOS doesn't really care what a file contains or is supposed to contain — it simply reads and writes data.

```
        BRK                 ;SOS call (but you knew that already)
        .BYTE 0C1               ;DESTROY's i.d. number
        .WORD DES_PARAMS          ;pointer to parameters

DES_PARAMS    .BYTE 01              ;1 parameter
        .WORD DES_PATH          ;pointer to file pathname

DES_PATH     .BYTE 1C              ;length of name
        .s095ASCIIs100 "/THURSDAY/HAPPY.TIMES/LENDER" ;pathnamc
```

Listing 5

```
        BRK           ;Guess what (have you been reading along?)
        .BYTE 0C8           ;I.D. number for OPEN
        .WORD OPEN_PARAMS       ;pointer to parameteters

OPEN_PARAMS    .BYTE 04            ;4 parameters
        .WORD OPEN_PATH         ;pointer to file's name
OPEN_REF     .BLOCK 1            ;reserve 1 block for Refnum result
        .WORD 0000          ;we're not using any optional params...
        .BYTE 00           ;...so we make these all zeroes

OPEN_PATH     .BYTE              ;length of name
        .s095ASCIIs100 "/MORK/ASPHALT" ;the pathname itself
```

Listing 6

```
        LDA OPEN_REF         ;move the Refnum we obtained earlier
        STA WRITE_REF         ;into WRITE's parameter list
        BRK              ;call up SOS (hello, SOS?)
        .BYTE 0CB            ;call i.d. number
        .WORD WRIT_PARAMS       ;pointer to parameters

WRIT_PARAMS    .BYTE 03            ;3 parameters
WRIT_REF     .BLOCK 1            ;the above STA puts the proper Refnum
                  ;value in this byte
        .WORD DATA_BUF         ;pointer to our data buffer
        .WORD 000A          ;write 10 decimal (0A hex) bytes

DATA_BUF     .BYTE 01,23,45,67,89,AB,CD,EF,FF,FF ;10 randomly
chosen data
```

Listing 7

READ

This call attempts to transfer a given number of bytes from a file to a specified buffer. The other half of the world-famous read/write team!

call number: $CA

parameters: 4

1. Refnum value (1 byte)
   The Refnum assigned to the file when it was opened (as in the WRITE call).
2. Buf pointer (2 bytes)
   Points to a buffer area where the information will be placed after it is read (again, note the symmetry with the WRITE call).
3. Bytes value (2 bytes)
   The number of bytes to be read.
4. Bytes-read result (2 bytes)
   SOS returns the number of bytes actually read in these locations.

An Example (Listing 8):

Attempt to read 10 bytes from the file we opened earlier, named (/MORK/ASPHALT).

Where's **DATA-BUF**? Remember, we defined it in the WRITE call. Can this buffer area be reused? Sure! In fact, that's one of the benefits of SOS's system of parameter lists and pointers. You can use the same area in memory as a read and write buffer.

If you executed this call after just having written to the file earlier (as we have done in this article), you would get an error #4C, End of file. Wait a minute, you may say — we just wrote 10 bytes of data, so why won't they be read? The answer lies in the fact that whenever SOS reads from or writes to a file, it maintains a pointer, or mark, into that file, kind of like a book marker, so that it knows where to read from or write to next. After we wrote the 10 bytes out (with our example WRITE call), that marker was pointing to the end of file. When the subsequent READ came up, there was nothing left to read.

What do you do if you want to move the mark without reading or writing anything? Why, there just happen to be a couple of SOS calls (GET-MARK and SET-MARK) that let you look at and modify the mark. I won't go into them in depth here, but be advised of their existence.

CLOSE

This is the call to use to finish up the use of an open file.

call number: $CC

parameters: 1 (this is a simple one)

1. Refnum value (1 byte)
   The Refnum assigned to the file when it was opened.

An Example (Listing 9).

Close the file we've been working with.

You now have the basic tools necessary to work with SOS's file system. **CREATE** makes the files, **OPEN** gets them ready for reading and writing, **READ** and **WRITE** perform the actual transfer of data to and from the files, **CLOSE** finishes the reading and writing process, and **DESTROY** gets rid of the files.

I Promised You an Interpreter

Yes, I did, way back at the beginning of this article, say that we'd create a real, working interpreter before we were done, and we're about to do just that. Just as our first SOS call and module examples were simple, we'll begin with a fairly mindless interpreter. This one will simply print a welcoming message on the screen and then sit there. Not terribly exciting, I admit, but we need a place to start! (We'll get fancy later).

As noted earlier, **SOS.INTERP** files start with a special header block, then get right into the code. Well, our code will consist of three things:

1. OPENing the .CONSOLE device (so that we can print on the screen).
2. WRITing the message to the .CONSOLE.
3. Looping infinitely.

Since the how-to of all this stuff has been explained, let's proceed with the source text listing, Listing 10.

```
         LDA OPEN_REF          ;move the Refnum we obtained earlier
         STA READ_REF          ;into READ's parameter list
         BRK              ;now call SOS
         .BYTE 0CA          ;call i.d. number
         .WORD READ_PARAMS      ;pointer to parameters

READ_PARAMS    .BYTE 04          ;4 parameters
READ_REF     .BLOCK 1          ;our STA instruction above loads this
;byte with the proper Refnum value
         .WORD DATA_BUF         ;pointer to the buffer where data read
                    ;will go
         .WORD 000A          ;0A hex is 10 decimal; read 10 bytes
BYTES_READ    .BLOCK 2          ;reserve two bytes for SOS to put the
;number of bytes actually read
```

Listing 8

```
          LDA OPEN_REF         ;as we did with READ and WRITE,
          STA CLOSE_REF         ;bring in the desired Refnum
          BRK             ;then do the SOS call itself
          .BYTE 0CC          ;CLOSE call i.d. number
          .WORD CLOSE_PARAMS      ;parameter list

 CLOSE        .BYTE 01          ;one parameter only
 CLOSE_REF      .BLOCK 1           ;reserve a space for the Refnum
```

Listing 9

```
;first, some administrative stuff

        .ABSOLUTE          ;required for interpreters
        .PROC MYINTERP       ;this is the title (clever, huh?)

START       .EQU 0B000        ;code will load here
        .ORG START-0E       ;move back 14 bytes for header

;required header follows

HEADER      .ASCII "SOS NTRP"     ;required header information
        .WORD 0000         ;no optional header block (length 0)
        .WORD START          ;loading address
        .WORD CODELENG        ;length of code

;this is the working program

        BRK             ;the code itself: OPEN call
        .BYTE 0C8
        .WORD OP_LIST

        LDA OP_REF         ;put file's Refnum in WRITE's
        STA WR_REF         ;parameter list

        BRK             ;WRITE call
        .BYTE 0CB
        .WORD WR_LIST

LOOP        JMP LOOP          ;run around in circles forever

;parameter lists come next

OP_LIST     .BYTE 04          ;OPEN has four parameters
        .WORD CONS_PATH       ;pointer to the pathname to open
OP_REF      .BLOCK 1          ;reserve a place for SOS to put Refnum
        .WORD 0000         ;no optional parameters needed...
        .BYTE 00          ;...so these are zeroes

CONS_PATH    .BYTE 08          ;length of pathname
        .ASCII ".CONSOLE"     ;the file to open

WR_LIST     .BYTE 03          ;three parameters for WRITE
WR_REF      .BLOCK 1          ;save space for Refnum
        .WORD WR_BUF        ;pointer to our data
        .WORD 001F         ;length of our message

;the greeting message

WR_BUF      .ASCII "Hi, I'm Irving the interpreter!"

;close up shop

CODELENG     .EQU *-START       ;figures length of code for header

        .END           ;all done
```

Listing 10

```
program interp_maker;  ( By Alan Anderson; from Apple Orchard )

  var
   infile, outfile : file;
   inname, outname : string;
   data : packed array [1..512] of 0..255;
   block_num, count : integer;


begin
  write ('Enter the pathname of the codefile to be converted
-->');
  readln (inname);
  write ('Enter the pathname for the output file -->');
  readln (outname);
  reset (infile, inname);
  rewrite (outfile, outname);
  count := blockread (infile, data, 1);
   while not eof (infile) do
     begin
      count := blockread (infile, data, 1);
      count := blockwrite (outfile, data, count);
     end;
  close (infile);
  close (outfile, lock);
end.
```

Listing 11

That's it! Type Listing 10 in the Pascal Editor and assemble it and . . .you're almost there. There's one more item to consider: When the Pascal assembler writes a codefile, it writes a single block of information which is placed at the front of the codefile. However, although the assembler always writes this block, the information therein is useful only for modules, and this block must be removed from the front of interpreter files. The ideal solution to this situation would be a pseudo-op (called, perhaps, .MAKEINTERP), which would generate files without the information block. The current solution, though, is to have a Pascal program to rewrite the file without the information block.

When the *Apple /// SOS Reference Manual* is distributed, Apple plans to include a program to perform this function. Until then, here is a program, Listing 11, (without any error checking) to accomplish the same purpose.

After you've assembled the interpreter listed above, enter and compile this program, then execute it and convert the codefile. The final product is now a real, live, almost useful interpreter. What do you have to do to use it? Just format a diskette and put **SOS.KERNEL** and **SOS.DRIVER** on it. Then copy the converted codefile from our interpreter maker to the new disk and call it (naturally enough) **SOS.INTERP.** If you've done everything right and the stars are smiling upon you, you should then be able to boot the diskette and have it say "Hi" to you! All right!

You have just created an Assembly language program which executes all by itself, without BASIC or Pascal or any other high level language hanging around. Although it performs no useful function other than as a demonstration, it allows you to view the basic structure needed to write your own interpreters.

I hope all these goodies about interpreters and SOS calls will be enough to keep you going until next time. If not, please write to me. I can be had at:

Alan Anderson
c/o **Apple Orchard**
910 A George St
Santa Clara, CA 95050

Next time, I'll probably present some more tools for programming the bejeebers out of the Apple ///, probably in the form of more SOS calls and ways to exploit all the power in the .CONSOLE driver. However, this is changable according to your whims, so let me know what you want.

Okay, everybody. . .
HIT THE SOS!