NO. 46 MARCH 1982

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £.80

MICRO™

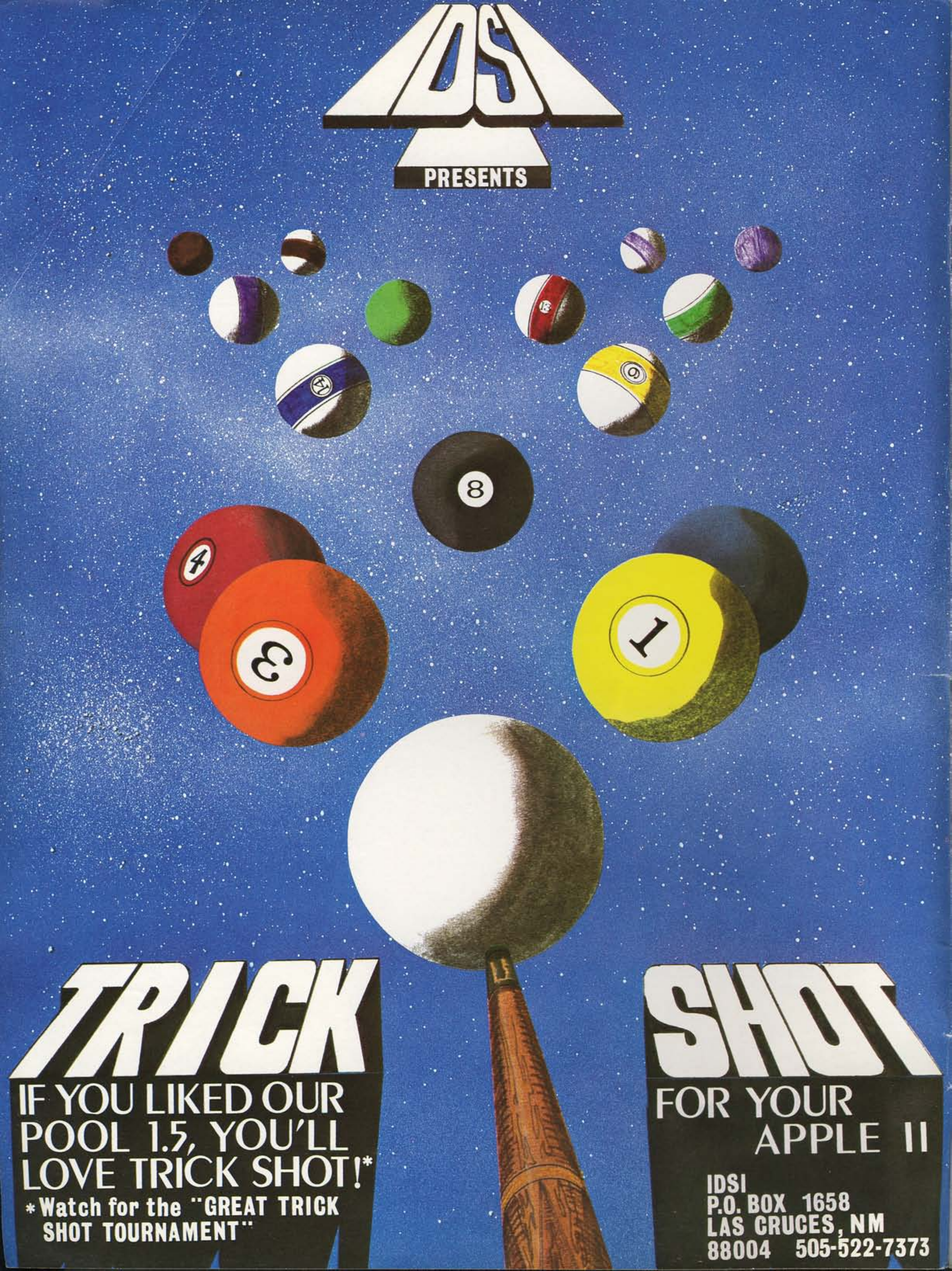
THE 6502/6809 JOURNAL

OSI Feature

Math Applications

A Disassembler for the 6809

I/O Expansion for the AIM


IDSI
PRESENTS
8
4
3
1
TRICK SHOT
IF YOU LIKED OUR POOL 1.5, YOU'LL LOVE TRICK SHOT!*
* Watch for the "GREAT TRICK SHOT TOURNAMENT"
FOR YOUR APPLE II
IDSI
P.O. BOX 1658
LAS CRUCES, NM
88004 505-522-7373

JUDGE THE REST, THEN BUY THE BEST

Only GIMIX offers you **SOFTWARE SWITCHING** between **MICROWARE's OS-9** and **TSC's FLEX**. Plus you get the power of the GMXBUG system monitor with its advanced debugging utility, and memory manipulation routines. A wide variety of languages and other software is available for these two predominant 6809 Disk Operating Systems.

You can order a system to meet your needs, or select from the 6809 Systems featured below.

JUDGE THE FEATURES AND QUALITY OF GIMIX 6809 SYSTEMS

GIMIX' CLASSY CHASSIS™ is a heavyweight aluminum mainframe cabinet with back panel cutouts to conveniently connect your terminals, printers, drives, monitors, etc. A 3 position keyswitch lets you lock out the reset switch. The power supply features a ferro-resonant constant voltage transformer that supplies 8V at 30 amps, + 15V at 5 amps, and − 15V at 5 amps to insure against problems caused by adverse power input conditions. It supplies power for all the boards in a fully loaded system plus two 5 ¼'' drives (yes! even a Winchester) that can be installed in the cabinet. The Mother board has fifteen 50 pin and eight 30 pin slots to give you the most room for expansion of any SS50 system available. 11 standard baud rates from 75 to 38.4K are provided and the I/O section has its own extended addressing to permit the maximum memory address space to be used. The 2 Mhz 6809 CPU card has both a time of day clock with battery back-up and a 6840 programmable timer. It also contains 1K RAM, 4 PROM/ROM/RAM sockets, and provides for an optional 9511A or 9512 Arithmetic Processor. The RAM boards use high speed, low power STATIC memory that is fully compatible with any DMA technique. STATIC RAM requires no refresh timing, no wait states or clock stretching, and allows fast, reliable operation. The system includes a 2 port RS232 serial interface and cables. All GIMIX boards use gold plated bus connectors and are fully socketed. GIMIX designs, manufactures, and tests in-house its complete line of products. All boards are twice tested, and burned in electrically to insure reliability and freedom from infant mortality of component parts. All systems are assembled and then retested as a system after being configured to your specific order.

56KB 2MHZ 6809 SYSTEMS WITH GMXBUX/FLEX/OS-9 SOFTWARE SELECTABLE

With #58 single density disk controller **$2988.59**
With #68 DMA double density disk controller **$3248.49**
to substitute Non-volatile CMOS RAM with battery back-up, add 300.00
for 50 Hz export power supply models, add 30.00

Either controller can be used with any combination of 5'' and/or 8'' drives, up to 4 drives total, have data recovery circuits (data separators), and are designed to fully meet the timing requirements of the controller I.C.s.

5 ¼'' DRIVES INSTALLED IN THE ABOVE with all necessary cables

	SINGLE DENSITY		DOUBLE DENSITY			
	Formatted	**Unformatted**	**Formatted**	**Unformatted**		
40 track (48TPI) single sided	199,680	250,000	341,424	500,000	2 for **$700.00**	**Chart shows total**
40 track (48TPI) double sided	399,360	500,000	718,848	1,000,000	2 for **900.00**	**capacity in Bytes for**
80 track (96TPI) single	404,480	500,000	728,064	1,000,000	2 for **900.00**	**2 drives.**
80 track (96TPI) double	808,960	1,000,000	1,456,128	2,000,000	2 for **1300.00**	

Contact GIMIX for price and availability of 8'' floppy disk drives and cabinets; and 5'' and 8'' Winchester hard disk system.

128KB 2Mhz 6809 DMA Systems for use with TSC's UNIFLEX or MICROWARES's OS-9 Level 2

(Software and drives not included) **$3798.39**
to substitute 128KB CMOS RAM with battery back-up, add 600.00
for each additional 64KB NMOS STATIC RAM board, add 639.67
for each additional 64KB CMOS STATIC RAM board, add 988.64
for 50 Hz export power supply, add 30.00

NOTE: *UNIFLEX can not be used with 5'' minifloppy drives.*

GIMIX has a wide variety of RAM, ROM, Serial and Parallel I/O, Video, Graphics, and other SS50 bus cards that can be added now or in the future. Phone or write for more complete information and brochure.

THE SUN NEVER SETS ON GIMIX USERS

GIMIX Systems are found on every continent, except Antarctica. (Any users there? If so, please contact GIMIX so we can change this.) A representative group of GIMIX users includes: **Government Research and Scientific Organizations** in Australia, Canada, U.K., and in the U.S.; NASA, Oak Ridge, White Plains, Fermilab, Argonne, Scripps, Sloan Kettering, Los Alamos National Labs, AURA. **Universities:** Carleton, Waterloo, Royal Military College, in Canada; Trier in Germany; and in the U.S.; Stanford, SUNY, Harvard, UCSD, Mississippi, Georgia Tech. **Industrial users** in Hong Kong, Malaysia, South Africa, Germany, Sweden, and in the U.S.; GTE, Becton Dickinson, American Hoechst, Monsanto, Allied, Honeywell, Perkin Elmer, Johnson Controls, Associated Press, Aydin, Newkirk Electric, Revere Sugar, HI-G/AMS Controls, Chevron. **Computer mainframe and peripheral manufacturers,** IBM, OKI, Computer Peripherals Inc., Qume, Floating Point Systems. **Software houses;** Microware, T.S.C., Lucidata, Norpak, Talbot, Stylo Systems, AAA, HHH, Frank Hogg Labs, Epstein Associates, Softwest, Dynasoft, Research Resources U.K., Microworks, Analog Systems, Computerized Business Systems.

TO ORDER BY MAIL

SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear.

U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00.

Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S.. Our bank is the Continental Illinois National Bank of Chicago, account #73-32033 Visa or Master Charge also accepted.

GIMIX INC. reserves the right to change pricing and product specifications at any time without further notice.

are registered trademarks of GIMIX Inc.

FLEX AND Uniflex are trademarks of Technical Systems Consultants Inc. OS-9 is a trademark of Microware Inc. See their ads for other **GIMIX** compatible software

GIMIX Systems are chosen by the Pros because of quality, reliability and features.

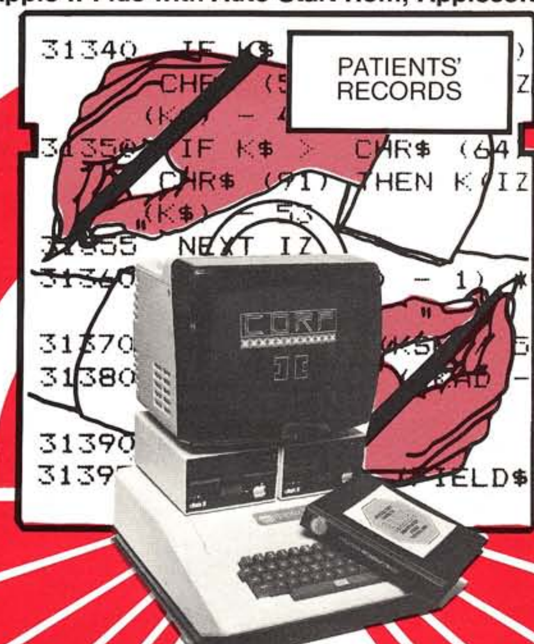
GIMIX INC.

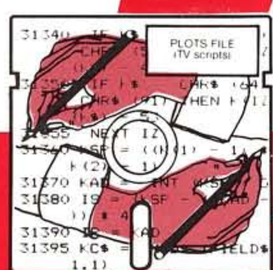
The Company that delivers
Quality Electronic products since 1975.

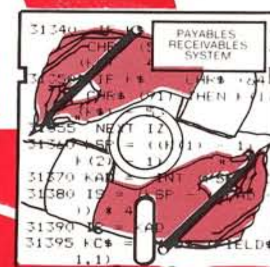
1337 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

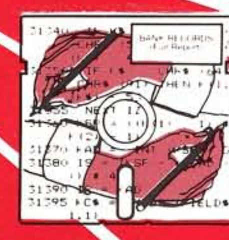
your power is ready... software that writes software
for Apple II™

NEEDED: Apple II Plus with Auto Start Rom, Applesoft in Rom, 48K.









C.O.R.P.™ writes software. Perhaps you should read that again. Not "simplifies programming" or "makes debugging easier." C.O.R.P. writes APPLESOFT BASIC—complete, stand-alone programs which run, bug-free, the very first time. You need not type a single character of basic code — ever. C.O.R.P. does that. Your answers to questions in simple English "design" the program. In minutes, C.O.R.P. writes all the program code, scrolls it to the screen and automatically saves it to your disk. Your program because you designed it. Once written, your program runs without C.O.R.P. You may list your program, examine it, modify it further or even sell it—as you wish, royalty free. No mere "data base manager," at any price, will do that.

The applications are almost limitless. Your C.O.R.P. program Generation System will: ● handle nearly as much data as your disks will hold (113K per disk!) ● find any record in less than a second ● let you design your own screens as quickly as you can move the curser around ● sort, re-sort and update your data for you ● examine and re-examine your data and then print reports, checks, invoices, statements, mailing labels, lists, memos—you name it. C.O.R.P. allows you to use your Apple II plus as the working tool it really is. Isn't that why you bought one?

Computers should write code. You should design programs. Now you have the power.

C.O.R.P. is a trade mark of Maromaty Scotto Software Corporation.

CORTECHS CORPORATION
900 North Franklin, Chicago, Illinois 60610
See your dealer today or call toll-free . . .

C.O.D. welcome.

C.O.R.P. I features:
● a data entry program generator ● a full sort sub-system ● an update sub-system ● a full diagnostic package ● a print program generator . . . $235.
Dealer inquiries welcome.

C.O.R.P.™

1-800-621-4109

C.O.R.P. II features:
● all of C.O.R.P. I plus: ● a system menu generator ● a complete forms letter ● a full files editor ● a system demo package ● an Applesoft tutorial . . . $425.
(In Illinois, call (312) 943-0700

NO. 46 MARCH 1982

MICRO™

THE 6502/6809 JOURNAL

DEPARTMENTS

MATH APPLICATIONS

OSI BASIC

UTILITIES

HARDWARE

THE CHIEFTAIN™ 5¼-INCH WINCHESTER HARD DISK COMPUTER

SO ADVANCED IN SO MANY WAYS . . . AND SO COST-EFFECTIVE . . . IT OBSOLETES MOST OTHER SYSTEMS AVAILABLE TODAY AT ANY PRICE.

• HARD DISK SYSTEM CAPACITY

The Chieftain series includes 5¼- and 8-inch Winchesters that range from 4- to 60-megabyte capacity, and higher as technology advances. All hard disk Chieftains include 64-k memory with two serial ports and DOS69D disk operating system.

• LIGHTNING ACCESS TIME

Average access time for 5¼-inch Winchesters is 70-msec, comparable to far more costly hard disk systems. That means data transfer ***ten-times faster*** than floppy disk systems.

• 2-MHZ OPERATION

All Chieftains operate at 2-MHz, regardless of disk storage type or operating system used. Compare this to other hard disk systems, no matter ***how*** much they cost!

• DMA DATA TRANSFER

DMA data transfer to-and-from tape and disk is provided for optimum speed. A special design technique eliminates the necessity of halting the processor to wait for data which normally transfers at a slower speed, determined by the rotational velocity of the disk.

• RUNS UNDER DOS OR OS-9

No matter which Chieftain you select . . . 5¼- or 8-inch floppy, or 5¼- or 8-inch Winchester with tape or floppy back-up . . . they ***all*** run under DOS or OS-9 with ***no need*** to modify hardware or software.

• UNBOUNDED FLEXIBILITY

You'll probably never use it, but any Chieftain hard disk system can drive up to 20 other Winchesters, and four tape drives, with a single DMA interface board!

• SMOKE SIGNAL'S HERITAGE OF EXCELLENCE

This new-generation computer is accompanied by the same ***Endurance-Certified*** quality Dealers and end-users all over the world have come to expect from Smoke Signal. And support, software selection and extremely competitive pricing are very much a part of that enviable reputation.

The Chieftain Computer Systems:

Here are the Chieftain 6809-based hard disk computers that are destined to change data processing . . .

☐ **CHIEFTAIN 95W4**
4-megabyte, 5¼-inch Winchester with a 360-k floppy disk drive (pictured).

☐ **CHIEFTAIN 95XW4**
4-megabyte, 5¼-inch Winchester with a 750-k octo-density floppy disk drive.

☐ **CHIEFTAIN 98W15**
15-megabyte, 5¼-inch Winchester with a 1-megabyte 8-inch floppy disk drive.

☐ **CHIEFTAIN 9W15T20**
15-megabyte, 5¼-inch Winchester with a 20-megabyte tape streamer.

20-Megabyte Tape Streamer Back-Up Option

Available with all Chieftain hard disk configurations. This cartridge tape capability provides full 20-megabyte disk back-up in less than five minutes with just one command, or copy command for individual file transfers. Transfers data tape-to-disk or disk-to-tape. Floppy back-up is also available in a variety of configurations.

Write or call today for details (including the low prices) on the Chieftain Series. . . and on dealership opportunities

SMOKE SIGNAL BROADCASTING®

31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

Name ____________________

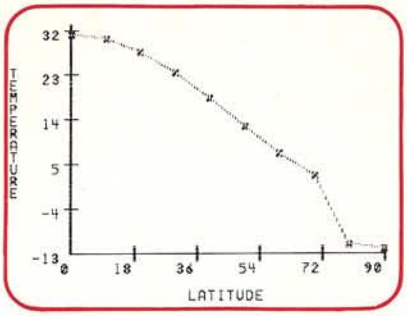
Company ____________________

Address ____________________

City ____________ State ________ Zip ________

Telephone ( ) ____________________

About the Cover



This month's cover photo brings us to the Homecoming '81 bonfire at Dartmouth College, the birthplace of BASIC. BASIC, one of the world's most popular programming languages, is the most important language in microcomputing since it is supplied with every microcomputer.

The graphic overlay was generated by the program "Glacier" which calculates surface temperatures by latitude. ("Glacier" by Compress, Inc., Wentworth, N.H.)

Cover photo by Ford Cavallari

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to
MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

MICRO Editorial

Hello, OSI?

What is OSI doing with its line of personal computers? A simple question, but even OSI can't seem to answer it — yet.

For months now we've been receiving letters from frustrated OSI users who wonder why we don't cover OSI machines more thoroughly (more than 30% of our readers are OSI users). These same readers also ask why OSI isn't advertising, and why OSI support in general seems to be fading. We decided to see how much of the fog could be cleared away.

Contacting OSI was naturally the first step, but not necessarily the easiest or most informative. One OSI source admitted that the company is beginning to stress the business line rather than the home market. According to this source, OSI is considering the elimination of the C1P and Superboard product lines. However, other officials insisted that no firm decision had been made yet.

Another view held by knowledgeable sources inside and outside OSI is that the company will sell off their low-end computer line rather than terminate it completely. OSI, according to these sources, would not want to leave owners of these products in the lurch. To sum up the problem, an OSI marketing official admitted that, for now, the situation regarding their line of personal computers is "up in the air."

Several factors can help account for this confusion. Much of OSI's staff has been busy lately, not only with deciding the fate of the personal computer line, but with moving operations from Ohio to Massachusetts. OSI was purchased in December 1980 by a Massachusetts-based firm, M/A COM, and the transition is still in progress. Also, preparations for an upcoming distributor's meeting in Florida are tying up key OSI officials.

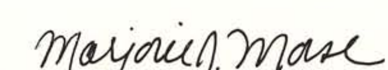
The distributor's meeting (which will have occurred by the time this issue is published) will include a presentation of new systems. OSI's recent lack of advertising can be explained partially by their need to wait until after the meeting to announce new products to the general public.

Several OSI dealers offered a variety of insights into the OSI personal computer problem. Although one dealer was optimistic and believed that OSI was "getting its act together," another considered the company "schizophrenic" and said that he thought OSI was "dumping the personal computer market." The general consensus among dealers is that OSI is developing a new line of computers aimed at the business instead of the hobbyist market.

One optimistic dealer said that the OSI personal computers will develop into a "nice market in spite of OSI." He believed this will be possible because other manufacturers will offer OSI-compatible hardware support.

Customer service has always been an issue for OSI users. According to one dealer, OSI has regarded customer service as a dealer obligation. But, since service is rarely a lucrative business, many dealers choose not to provide it. Therefore OSI users are neglected. In addition, some dealers are understandably reluctant to sell OSI personal computer products because of the uncertainty and confusion.

Although all the questions haven't been answered, and some of the answers we did receive are vague, we believe users will soon have a clearer picture of OSI's personal computer plans. We'll keep you informed as best we can.



MICRO New Publications

Reference

The Index, The ultimate information index for all personal computer users, W.H. Wallace, Indexor. Missouri Indexing, Inc. (P.O. Box 301, St. Ann, MO 63074), 1981, iii, 489 pages, 5¼ × 8¼ inches, paperbound.
$14.95

Here you'll find more than 30,000 entries covering over six years of articles, editorials, and columns from 45 computer publications. There are fourteen system-specific sub-indexes. All articles are listed alphabetically, along with the author, magazine and issue in which it appeared.

CONTENTS: *Introduction; How to Use the Index; Apple Articles; Atari Articles; CP/M Articles; North Star Articles; Ohio Scientific Articles; PET Articles; Southwestern Technical Products Articles; S-100 Articles; TRS-80 Articles; Z-80 Articles; 6502 Articles; 6800 Articles; 8080 Articles; General Articles; Magazine/Newsletter Abbreviations and Ordering Information.*

General

Introduction to Word Processing by Hal Glatzer. Sybex Inc. (2344 Sixth St., Berkeley, CA 94710), 1981, xiv, 210 pages, 6 × 9 inches, paperbound.
ISBN: 0-89588-076-8 $12.95

Learn what a word processor is, what it does, how to use one, and how to choose one. The author also provides a feature-by-feature comparative analysis of currently available equipment.

CONTENTS: *What Word Processors Can Do For You*—Why Doesn't Everybody Have One? What Do People Want? Will a Word Processor Help? *What The Newspapers Learned*—Getting the Lead Out; The Computers Arrive; The Price of Freedom; Embracing the Copper Wire. *Why Secretaries Are Going Back To School*—Word Processing Is Only the Beginning; "The £12 Look"; Larger Files in Smaller Cabinets; Will Machines Replace People? *How To Teach A Small Computer Big Tricks*—Game Players Are Computer Operators; Where Does Word Processing Come In? *Which Type of Word Processor Is Best?*—Electronic Typewriters; Stand-Alone Machines; Microcomputers; Mainframe and Minicomputers; There Is No One Solution; Benchmark Test. *Writing And Editing With A Word Processor*—Writing; Editing. *How To Manage Your Files*—About Bytes; Storage Devices; Electronic Filing. *Formatting What You Write*—Previewing; Basic Formatting; Advanced Formatting. *Putting Text On Paper*—Printing; Printer Enhancements. *Extending Your Reach*—Typesetting; Computer-Output Microfilm (COM); Optical Character Recognition (OCR); Multiple Work Stations; Telecommunications. *Will A Word Processor Pay For Itself?*—Comparing Costs; Holding Down Costs; Becoming A Customer; Avoiding Costly Problems. *How To Get Hands-On Experience*—Do Your Homework; Meet The People; How To Select A Vendor; In Conclusion. *Appendix: Where To Go For More Information. Glossary. Index. Library.*

6502

Beyond Games: Systems Software for Your 6502 Personal Computer by Ken Skier. BYTE/McGraw-Hill, Book Division (70 Main St., Peterborough, NH 03458), 1981, iv, 433 pages, 7¼ × 9¼ inches, paperbound.
ISBN: 0-07-057860-5 $14.95

A guided tour of your Apple, Atari, Ohio Scientific, or PET computer. This book takes you through basic concepts, such as "memory" and "program," right into assembly language programming. Several subroutines and programming aids are presented, including screen utilities, print utilities, a machine language monitor, a hexadecimal dump tool, a disassembler, and more.

CONTENTS: *Introduction; Your Computer; Introduction to Assembler; Loops and Subroutines; Arithmetic and Logic; Screen Utilities; The Visible Monitor; Print Utilities; Two Hexdump Tools; A Table-Driven Disassembler; A General MOVE Utility; A Simple Text Editor; Extending the Visible Monitor; Entering the Software Into Your System; Appendices.*

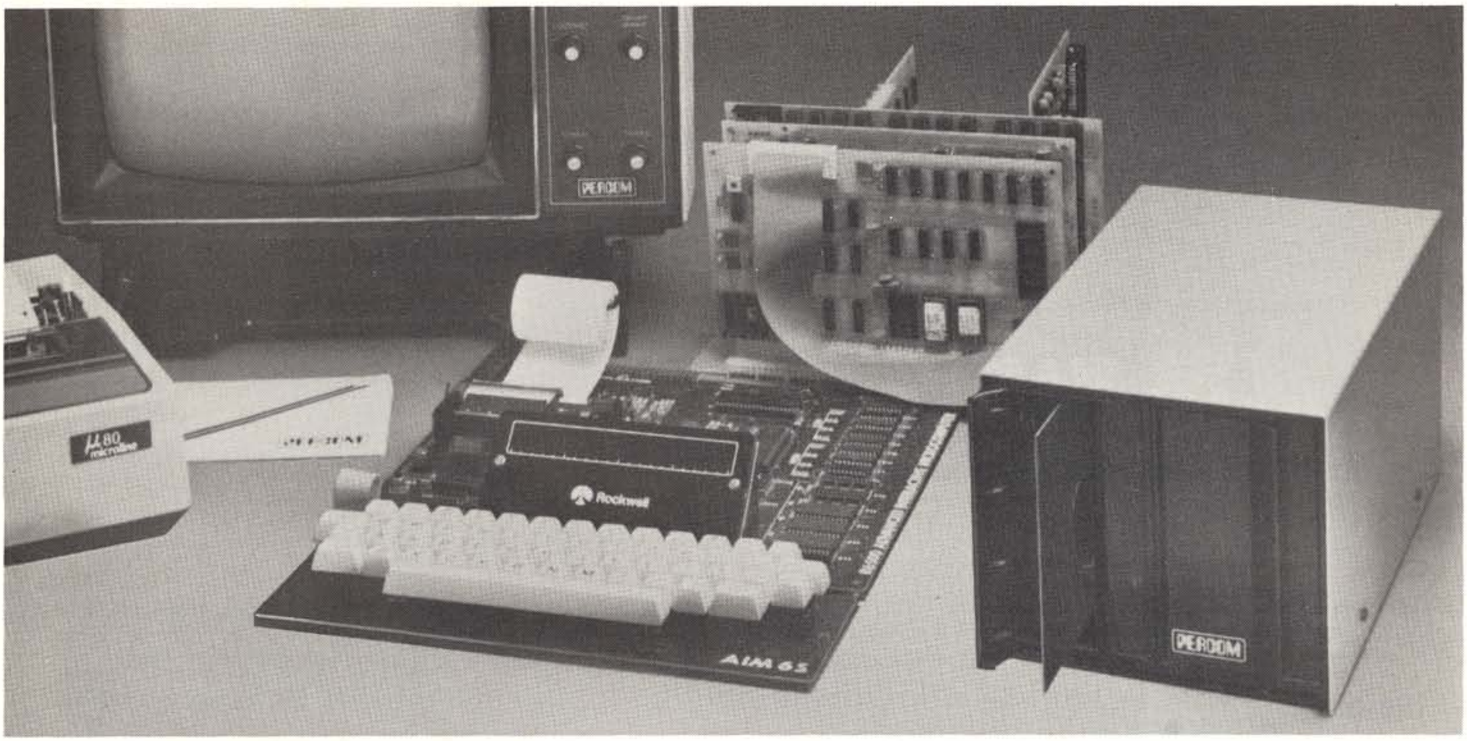
Games

Apple Pascal Games by Douglas Hergert and Joseph T. Kalash. Sybex Inc. (2344 Sixth St., Berkeley, CA 94710), 1981, xiii, 371 pages, 7 × 9 inches, paperbound.
ISBN: 0-89588-074-1 $14.95

A collection of games written in Apple Pascal, ranging from simple exercises to more advanced, strategic challenges. For each game the book includes game rules, and a guide to understanding the program. A "structure chart" demonstrates the organization of each program.

CONTENTS: *Introduction; Acknowledgements; Simple Games; More Advanced Games; Games that use TURTLEGRAPHICS; Cribbage. Appendices*—Reserved Words and Functions; Summary of Pascal.

MICRO

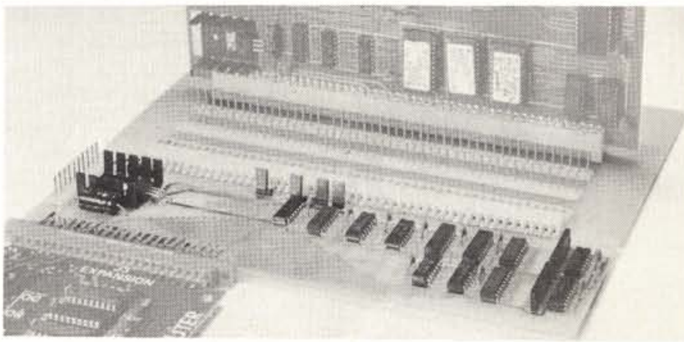


The **PERCOM 'M'** Line

Low-Cost Expansion for Your AIM, KIM or SYM Computer

Complete Disk Storage Systems from $599.95

- **The right storage capacity** – Available in 1-, 2- and 3-drive systems, with either 40- or 80-track drives.
- **Flippy storage** – Flippy drives (optional) let you flip a diskette and store data and program's on the second recording surface.
- **High Storage Capacity** – Formatted, one-side storage capacity is 102 Kbytes (40-track drive), 205 Kbytes (80-track drive).
- **Proven Controller** – The drive controller design is the same as the design used in the Percom 680X LFD mini-disk system. This system – introduced in 1977 – has given reliable service in thousands of applications. Two versions are available: the MFD-C65 for the AIM-65 expansion bus, and the MFD-C50 for the System-50 (SS-50) bus.
  - Includes an explicit data separator circuit that's reliable even at the highest bit densities.
  - Provides for on-card firmware.
  - Includes a motor inactivity time-out circuit.
  - Capable of handling up to four drives.
  - Capable of reading both hard- and soft-sectored diskettes.
- **DOS included** – The MFD disk-operating system works with the AIM monitor, editor, assembler, Basic and PL/65 programs, interface is direct, through user I/O and F1, F2 keys. Diskette includes DOS source code and library of 20 utility commands.
- **Reliability assurance** – Drives are burned-in 48 hours, under operating conditions, to flag and remove any units with latent defects.
- **Full documentation** – Comprehensive hardware and software manuals are included with each system.



Expand Your Computer with Proven System-50 Modules

The Percom M65/50 Interface Adapter connects your computer to Percom's System-50 (SS-50) motherboard, allowing you to expand your AIM, KIM or SYM with proven System-50 modules. You can add disk storage, memory modules,even a video display system. The M65/50 provides buffer-amplification of address, data and control lines. On-card decode circuitry lets you allocate address space either to the computer or to the expansion motherboard. Price: only $89.95, *including* System-50 motherboard.

System Requirements: AIM-65, KIM or SYM computer with expansion bus and four Kbytes RAM (min).

Yes . . . I'd like to know more about Percom MFD drive systems. Rush me free literature.

Send to
PERCOM DATA COMPANY, Inc., Dept. 65-M
11220 Pagemill Rd. Dallas, TX 75243

name

address

city | state

zip | phone number

MAIL TODAY!

PERCOM

PERCOM DATA COMPANY, INC.
11220 PAGEMILL RD. DALLAS, TX 75243
(214) 340-7081

Toll-Free Order Number: **1-800-527-1222**

PRICES AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE.

VisiCalc Formulas for Depreciation

These formulas are ready to key into your VisiCalc sheet. The author explains how the three different depreciation methods are used.

Kim G. Woodward
6526 Delia Dr.
Alexandria, Virginia 22310

VisiCalc, which is available in versions for Apple, Atari, PET/CBM, and TRS-80, is made by Personal Software. In this article I'll show one application of this fine program — that of depreciation schedules. I currently use this application for depreciating equipment for income tax purposes. (Before you use this program, please consult your tax advisor or IRS representative for the proper application of depreciation to your situation.)

If the formulas in listing 1 are keyed onto a blank VisiCalc sheet, the depreciation application will be up and running. (For further information, please refer to your VisiCalc reference book.) To explain how to use the depreciation application, I must discuss depreciation and the formulas as they are used.

The Depreciation Formulas

Webster's New International Dictionary defines depreciation as a "decline in [the] value of an asset due to such causes as wear and tear, action of the elements, obsolescence and inadequacy." Business uses depreciation to write off the cost of equipment and real estate. There are three kinds of depreciation in wide use today: straight line (SLD), declining balance (DB), and sum of the years digits (SYD).

In straight line depreciation, an equal amount is depreciated each year. In the declining balance method, a large amount is depreciated in the early life, then gradually tapers off toward the later life of an asset. It is generally used to write off the largest amount in the earliest time period. The sum of the years digits method is generally between these two.

There are three entities used in the calculation of depreciation:

1. the starting book value (SBV) — what the item cost;
2. the salvage value — what the item would sell for after its useful life;
3. the expected life of the item.

This depreciation application of VisiCalc provides the depreciation (DEP), the accumulated depreciation (ACD), the remaining depreciation left (RDV), and the remaining book value (RBV) for each of the three depreciation types. The formulas are given in terms of year j, the current year in the schedule we are looking at. The formulas for the three types are as follows:

1. Straight line depreciation

   DEP(j) = (SBV − salvage)/life

   ACD(j) = j * (DEP(j))

   RDV(j) = DEP(j) * (life − j)

   RBV(j) = RDV(j) + salvage

2. Sum of years digits depreciation

   sum of years digits = (INT(life) + 1) * (INT(life) + 2)/2

   DEP(j) = (life + 1 − j) * (SBV − salvage)/(sum of years digits)

   RDV(j) = (life − j + 1) * (life − j + 2) * (SBV − salvage)/(2 * (sum of years digits))

   RBV(j) = RDV(j) + salvage

   ACD(j) = (SBV − salvage) − RDV(j)

3. Declining balance depreciation

   DEP(j) = (SBV * (DBF/life)) * (1 − DBF/life) ^ (j − 1)

   ACD(j) = SBV * (1 − (1 − DBF/life) ^ j)

   RDV(j) = (SBV − salvage) − ACD(j)

   RBV(j) = RDV(j) + salvage

where DBF is the declining balance rate factor ranging from 1 to 2.

Note that these formulas are for each year of the schedule. Thus, by adjusting the way the calculations are performed, we can project the schedule out for as many years as the sheet will carry. In the case of this depreciation application, I have chosen a 5-year schedule.

Use of the Depreciation Application

Looking at the example, the starting book value is entered into position B3 (1330.00) on the sheet. Likewise, salvage value is entered into position B4 (20.00) and life of the item is entered into position B5 (5) of the sheet. The declining balance factor (from 1 to 2) can be entered into position E3 (1.50) of the sheet. I generally leave this position as 1.5 for comparison purposes.

Listing 1

VISICALC formulas for depreciation

```
>F29:@MAX(F10,F16,F22)
>E29:@MAX(E10,E16,E22)
>D29:@MAX(D10,D16,D22)
>C29:@MAX(C10,C16,C22)
>B29:@MAX(B10,B16,B22)
>A29:"MAX DEP
>F28:@MIN(F10,F16,F22)
>E28:@MIN(E10,E16,E22)
>D28:@MIN(D10,D16,D22)
>C28:@MIN(C10,C16,C22)
>B28:@MIN(B10,B16,B22)
>A28:"MIN DEP
>G27:/--
>F27:/--
>E27:/--
>D27:/--
>C27:/--
>B27:/--
>A27:/--
>G26:(1-E4)*F22
>F26:(E4*F22)+((1-E4)*E22)
>E26:(E4*E22)+((1-E4)*D22)
>D26:(E4*D22)+((1-E4)*C22)
>C26:(E4*C22)+((1-E4)*B22)
>B26:+E4*B22
>F25:+F24+B4
>E25:+E24+B4
>D25:+D24+B4
>C25:+C24+B4
>B25:+B24+B4
>A25:"DB  RBV
>F24:(B3-B4)-F23
>E24:(B3-B4)-E23
>D24:(B3-B4)-D23
>C24:(B3-B4)-C23
>B24:(B3-B4)-B23
>A24:"DB  RDV
>F23:+B3*(1-(@EXP(F8*@LN(1-(E3/B5)))))
>E23:+B3*(1-(@EXP(E8*@LN(1-(E3/B5)))))
>D23:+B3*(1-(@EXP(D8*@LN(1-(E3/B5)))))
>C23:+B3*(1-(@EXP(C8*@LN(1-(E3/B5)))))
>B23:+B3*(1-(@EXP(B8*@LN(1-(E3/B5)))))
>A23:"DB  ACD
>F22:@EXP(@LN(B3)+((F8-1)*@LN(1-(E3/B5)))+@LN(E3/B5))
>E22:@EXP(@LN(B3)+((E8-1)*@LN(1-(E3/B5)))+@LN(E3/B5))
>D22:@EXP(@LN(B3)+((D8-1)*@LN(1-(E3/B5)))+@LN(E3/B5))
>C22:@EXP(@LN(B3)+((C8-1)*@LN(1-(E3/B5)))+@LN(E3/B5))
>B22:@EXP(@LN(B3)+((B8-1)*@LN(1-(E3/B5)))+@LN(E3/B5))
>A22:"DB  DEP

>G21:/--
>F21:/--
>E21:/--
>D21:/--
>C21:/--
>B21:/--
>A21:/--
>G20:(1-E4)*F16
>F20:(E4*F16)+((1-E4)*E16)
>E20:(E4*E16)+((1-E4)*D16)
>D20:(E4*D16)+((1-E4)*C16)
>C20:(E4*C16)+((1-E4)*B16)
>B20:+E4*B16
>F19:(B3-B4)-F17
>E19:(B3-B4)-E17
>D19:(B3-B4)-D17
>C19:(B3-B4)-C17
>B19:(B3-B4)-B17
>A19:"SYD ACD
>F18:+F17+B4
>E18:+E17+B4
>D18:+D17+B4
>C18:+C17+B4
>B18:+B17+B4
>A18:"SYD RBV
>F17:(B5-F8+1)*(B5-F8)*(B3-B4)/(2*E5)
>E17:(B5-E8+1)*(B5-E8)*(B3-B4)/(2*E5)
>D17:(B5-D8+1)*(B5-D8)*(B3-B4)/(2*E5)
>C17:(B5-C8+1)*(B5-C8)*(B3-B4)/(2*E5)
>B17:(B5-B8+1)*(B5-B8)*(B3-B4)/(2*E5)
>A17:"SYD RDV
>F16:((B5+1-F8)/E5)*(B3-B4)
>E16:((B5+1-E8)/E5)*(B3-B4)
>D16:((B5+1-D8)/E5)*(B3-B4)
>C16:((B5+1-C8)/E5)*(B3-B4)
```

Listing 1 *(Continued)*

```
>B16:((B5+1-B8)/E5)*(B3-B4)
>A16:"SYD DEP
>G15:/--
>F15:/--
>E15:/--
>D15:/--
>C15:/--
>B15:/--
>A15:/--
>G14:(1-E4)*F10
>F14:(E4*F10)+((1-E4)*E10)
>E14:(E4*E10)+((1-E4)*D10)
>D14:(E4*D10)+((1-E4)*C10)
>C14:(E4*C10)+((1-E4)*B10)
>B14:+E4*B10

>F13:+F12+B4
>E13:+E12+B4
>D13:+D12+B4
>C13:+C12+B4
>B13:+B12+B4
>A13:"SLD RBV
>F12:(B5-F8)*F10
>E12:(B5-E8)*E10
>D12:(B5-D8)*D10
>C12:(B5-C8)*C10
>B12:(B5-B8)*B10
>A12:"SLD RDV
>F11:+F10*F8
>E11:+E10*E8
>D11:+D10*D8
>C11:+C10*C8
>B11:+B10*B8
>A11:"SLD ACD
>F10:(B3-B4)/B5
>E10:(B3-B4)/B5
>D10:(B3-B4)/B5
>C10:(B3-B4)/B5
>B10:(B3-B4)/B5
>A10:"SLD DEP
>G9:/--
>F9:/--
>E9:/--
>D9:/--
>C9:/--
>B9:/--
>A9:/--
>G8:/FI6
>F8:/FI+E8+1
>E8:/FI+D8+1
>D8:/FI+C8+1
>C8:/FI+B8+1
>B8:/FI+B6
>A8:"YEAR
>G7:/--
>F7:/--
>E7:/--
>D7:/--
>C7:/--
>B7:/--
>A7:/--
>B6:/FI1
>A6:"START YR
>E5:/FI+B5*(B5+1)/2
>D5:"SOYD
>B5:/FI5
>A5:"LIFE

>E4:244/366
>D4:"PRCNT YR
>B4:0
>A4:"SALVAGE
>E3:1.5
>D3:"FACT
>B3:617.75
>A3:"BOOK VAL
>A1:"ITEM:
/W1
/GOR
/GRA
/GF$
/GC9
```

Example of 5-year depreciation schedule.

ITEM: APPLE II

Book Val	**1330.00**	**Fact**	**1.50**
Salvage	**20.00**	**Prcnt Yr**	**0.70**
Life	**5**	**SOYD**	**15**
Start Yr	**1**		

Year	**1**	**2**	**3**	**4**	**5**	**6**
SLD DEP	262.00	262.00	262.00	262.00	262.00	
SLD ACD	262.00	524.00	786.00	1048.00	1310.00	
SLD RDV	1048.00	786.00	524.00	262.00	0.00	
SLD RBV	1068.00	806.00	544.00	282.00	20.00	
	182.54	262.00	262.00	262.00	262.00	79.46
SYD DEP	436.67	349.33	262.00	174.67	87.33	
SYD RDV	873.33	524.00	262.00	87.33	0.00	
SYD RBV	893.33	544.00	282.00	107.33	20.00	
SYD ACD	436.67	786.00	1048.00	1222.67	1310.00	
	304.23	375.82	288.49	201.15	113.82	26.49
DB DEP	399.00	279.30	195.51	136.86	95.80	
DB ACD	399.00	678.30	873.81	1010.67	1106.47	
DB RDV	911.00	631.70	436.19	299.33	203.53	
DB RBV	931.00	651.70	456.19	319.33	223.53	
	277.99	315.60	220.92	154.65	108.25	29.05
MIN DEP	262.00	262.00	195.51	136.86	87.33	
MAX DEP	436.67	349.33	262.00	262.00	262.00	

Next, by entering the starting year of the schedule in position B6 (1), we can get depreciation schedule for a period of five consecutive years. Since the IRS will allow the percentage of the depreciation for the balance of the year, I have made provision for the entry of the balance. The balance of the year as a decimal can be placed into position E4 (0.70). The fifth line for each of the types gives the depreciation that may be taken in that year (valid only with a starting year of 1).

Position E5 (15) of the sheet is for the calculation of the sum of the years digits. The minimum and maximum values for depreciation in each of the years is provided as the last two lines. As there is no way of zeroing years past the life, note that the RDV will become negative in this case.

In Conclusion

This application of the VisiCalc program has been provided as a tool to help make difficult calculation easy. It will provide a quick glance at alternatives for use in business decisions and the big "if" question as well.

MICRO

Numerical Solution of Differential Equations

A brief discussion of the Runge-Kutta method of solving differential equations is accompanied by an Applesoft program that prints out and plots the points for the resulting curve.

Robert D. Walker
6100 F. Woodchase Lane
Marietta, Georgia 30067

Mathematical analysis of many physical phenomena, especially in engineering, requires solution of differential equations. Generally these equations are solved by a hodgepodge of techniques learned in an elementary differential equations course. Most higher-order equations require laborious techniques, and yet fail to yield solutions in the closed form. No simple formula is attained.

These complex equations are best solved by using numerical methods on a computer. This article includes an Applesoft program and short description of a fourth-order Runge-Kutta method for solving ordinary differential equations, given initial conditions (initial value problems). Although this program was written for the Apple computer, with minimal changes it is capable of running on another system.

The Runge-Kutta Method

An exhaustive derivation on the Runge-Kutta method will be omitted from this article, but may be found in most differential equations texts. The general idea behind this method is fairly simple. Let's assume you're given the following differential equation:

Equation 1

$$dY/dX = Y' = f(X,Y)$$

In addition you're given the initial conditions:

$$Y = Yn \text{ at } X = Xn$$

With this information one can easily compute the slope of the line tangent to the solution curve $(Y = g(X))$ at (Xn,Yn). This will simply be equal to $f(Xn,Yn)$.

Now let's assume Xn is incremented by some small value, Xi. We'll call this new value of X, $Xn+1$.

$$Xn+1 = Xn + Xi$$

The problem now is to approximate the corresponding Y value, $Yn+1$. Intuitively it should seem reasonable that for a very small increment of X the following approximation is true:

Equation 2

$$Yn+1 \doteq Yn + f(Xn,Yn)*Xi$$

Editor's Note: $\doteq$ means approximately equal to.

This is known as Euler's one-step method.

Provided the curve in question was linear (a straight line), the left and right sides of the equation 2 would be exactly equal. Obviously this is not true except in the most trivial cases, where $f(X,Y)$ is equal to some constant. As a result, it is necessary to replace the value $f(Xn,Yn)$ with a better approximation of the slope between the points (Xn,Yn) and $(Xn+1, Yn+1)$, particularly if accuracy is important. In effect this is what the Runge-Kutta method does. It uses a "weighted average" of slopes within the interval $Xn <= X <= Xn+1$.

The formula for the fourth-order Runge-Kutta method using Runge's coefficients is as follows:

Equation 3

$$Yn+1 \doteq Yn + M*Xi$$

where

Equation 4

$$M = (m0 + 2*m1 + 2*m2 + m3)$$

Equation 4a

$$m0 = f(Xn,Yn)$$

Equation 4b

$$m1 = f(Xn + Xi/2, Yn + (m0/2)*Xi)$$

Equation 4c

$$m2 = f(Xn + Xi/2, Yn + (m1/2)*Xi)$$

Equation 4d

$$m3 = f(Xn + Xi, Yn + m2*Xi)$$

Note that $f(Xn, Yn)$ in equation 2 has been replaced by M in equation 3. The value M is the "weighted average" of the slopes. The computed values $m0$, $m1$, $m2$, and $m3$ are the slopes used to compute M. Figure 1 includes a geometric interpretation of these values.

Let's summarize what we've accomplished so far. Given a first-order ordinary differential equation and initial conditions, we are able to iteratively approximate values of Y along an interval of X.

Up to this point we have limited our discussion to first-order equations. Solving higher-order equations, however, is just as easy. In fact the Runge-Kutta method described above is not changed. The ability to transform a higher-order equation into a system of first-order equations is the key.

For example, let's assume you're given the following second-order equation:

Equation 5

$$Y1'' + Y1' + Y1 = \sin(X)$$

Make the substitution

Equation 6

$$Y2 = Y1'$$

From this substitution you acquire the following system of first-order equations:

Equation 7

$$Y1' = Y2$$

Equation 8

$$Y2' = -Y1 - Y2 + \sin(X)$$

Our intention is to approximate the values *Y1* and *Y2* along a specified *X*-interval. This requires us to use the Runge-Kutta method twice for each *X* value. In particular, *Y1'* (equation 7) is essential for approximating *Y1*. Likewise, it would be impossible to approximate *Y2* without *Y2'* (equation 8). This same procedure may be extended for higher-order equations.

The above example is straightforward. Nonetheless, complications may arise in the way in which these first-order equations are coupled. For instance, we could have a system of equations whereby some derivatives are functions of other derivatives. In this case, the order in which the derivatives are calculated becomes important. Further explanation of this problem is beyond the scope of this article, and left to the reader.

Program Implementation

As with all programs, it was first necessary to determine exactly what the program was to accomplish. After careful consideration I decided to have the program compute *Y1* and *Y2* (= *Y1'*) along a specified *X*-interval. These values would then be printed in tabular form. As you know, tables of numbers do not readily reveal the behavior of functions as well as graphs. Consequently, I decided that the values *Y1* and *Y2* would be plotted adjacent to the table.

The finished program (see listing) is divided into three main parts — introduction, calculations, and printout. Table 1 is a list of variables used in the program.

The introduction (lines 100-225) prompts the user for the parameters used to compute the *Y* values. These parameters include the number of first-order equations, the *X*-interval, the *X*-increment for calculations (*Xi*, equation 3), the *X*-increment for printout, and the initial conditions *Y(1..N)*.

These initial conditions are then sent to a subroutine (lines 800-845) which saves *X*, *Y(1)*, and *Y(2)* in three arrays X( ), P1( ), and P2( ) for printing at a later time. In addition, this subroutine will determine minimum and maximum values for *Y(1)* and *Y(2)*. These extremes will be necessary for plotting.

Lines 300 through 410 include the actual implementation of the Runge-Kutta method. This will require some explanation.

At the beginning of this section, X is equal to XB (beginning of X-interval) and the values Y(1..N) are set to the initial conditions. When line 305 is executed, the derivatives F(1..N) are calculated at the points (Xn,Y(1..N)). These are equivalent to *m0* (equation 4a).

Lines 310 through 320 update Y(1..N). First, the values Y(1..N) are saved in the array YN(1..N). Then, the values of the slopes F(1..N) are saved in the array M(1..N). Finally, the values Y (1..N) are updated so that *ml* can be computed next.

Line 325 increments X by *XI/2*. At this point $Xn = Xn + Xi/2$ and $Yn = Yn + (m0/2*Xi$. Line 330 then calculates *m1* (equation 4b). Lines 335 through 350 sum M(1..N) and update Y so that *m2* can be calculated next (line 355).

Likewise, lines 360 through 375 sum M(1..N). In addition, X is again incremented by *XI/2* (line 380), and $Y = Yn + m2*Xi$. M3 is calculated in line 385.

Lines 390 through 400 actually calculate the $Yn+1$ values.

Line 405 checks to see if X is sufficiently close to XP. If it is, then the

***Figure 1:* Slopes used by 4th order Runge-Kutta method.**

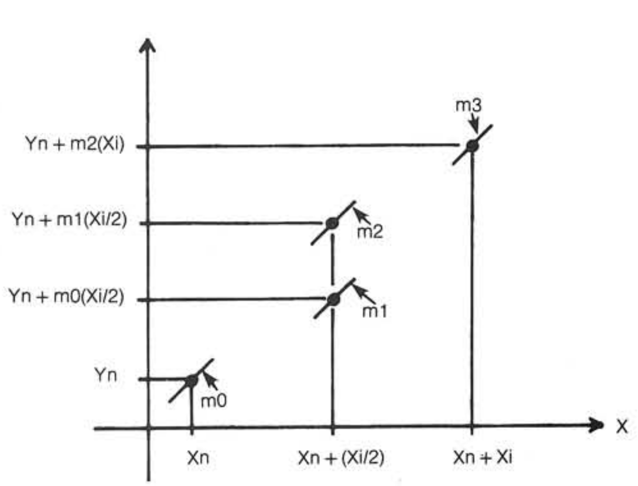


***Table 1:* List of Variables**

A$	General
E	Index, first-order eqs. 1..N
F( )	Derivatives of Y( ); ex. F(1) = Y(1)'
F1	Tabbing factor Y(1)
F2	Tabbing factor Y(2)
I	Index, initial conditions Y(1..I)
IC	X-increment for calculations
IP	X-increment for printout
J	Index, printout X(1..J), P1(1..J), and P2(1..J)
M( )	Sum of slopes for each first-order eqs. 1..N
N	Number of first-order eqs.
N1	Minimum value of Y(1)
N2	Minimum value of Y(2)
P1( )	Printed value of Y(1)
P2( )	Printed value of Y(2)
T1	Number of spaces right of col. 47 for plot Y(1)
T2	Number of spaces right of col. 47 for plot Y(2)
X	X value along X-interval
X1	Maximum value of Y(1)
X2	Maximum value of Y(2)
XB	Beginning of X-interval
XE	End of X-interval
Y( )	Value of Yn for first-order eqs. 1..N
YI( )	Initial values of Y(1..N)
YN( )	Temp. value of Yn for first-order eqs. 1..N

values X, Y(1), and Y(2) are saved by the subroutine on lines 800 through 845.

Line 410 compares X to see if XE (end of the *X*-interval) has been reached. If it hasn't then $Yn+1$ is calculated. Upon reaching the end of the *X*-interval, the results are printed.

The third main section, lines 500 through 700, prints the results that have been stored in arrays X(1..J), P1(1..J), and P2(1..J), where J is the number of values stored. Lines 500 through 600 print general information about the solution. This is self-explanatory. However, note that line 510 may be deleted for some printers.

Lines 605 through 620 calculate the tabbing factors, F1 and F2, for plotting Y(1) and Y(2). These variables are used to scale the plotted points so that the minimum Y value falls on column 47 and the maximum Y value falls on column 79.

The table heading is printed by lines 625 and 630. Next, lines 635 through 685 print the table and plot the results. Columns 47 through 79 are reserved for the graph. T1 and T2 (lines 655 and 660) are the amount of spaces to the

```
10  REM *****************************************
20  REM **   4TH-ORDER  RUNGE-KUTTA METHOD    **
30  REM **            FOR SOLUTION OF         **
40  REM **        DIFFERENTIAL EQUATIONS      **
50  REM **                                    **
60  REM **          WRITTEN BY R. WALKER      **
70  REM **               WICHITA, KS          **
80  REM *****************************************
90  DIM Y(5),YI(5),YN(5),F(5),M(5),XP(200),P1(200),P2(200)
95 :
96 :
100  REM ** INTRODUCTION ***************************
105  HOME
110  HTAB (6): PRINT "4TH-ORDER RUNGE-KUTTA METHOD"
115  HTAB (12): PRINT "FOR SOLUTION OF"
120  HTAB (9): PRINT "DIFFERENTIAL EQUATIONS"
125  PRINT : PRINT : PRINT
130  PRINT "ENTER SYSTEM OF FIRST-ORDER EQUATIONS"
135  PRINT "ON LINES 1001-1998."
140  LIST 1001,1998
145  INPUT "CONTINUE (Y/N)? ";A$
150  IF A$ = "N" THEN  END
155  HOME
160  VTAB (3)
165  INPUT "NUMBER OF 1ST-ORDER EQS.- ";N
170  INPUT "INTERVAL OF X (BEGIN,END)- ";XB,XE
175  INPUT "INCREMENT OF X (CALC)- ";IC
180  INPUT "INCREMENT OF X (PRINT)- ";IP
185  PRINT : PRINT "INITIAL VALUE(S):"
190  FOR I = 1 TO N
195  PRINT "  Y(";I;")";: INPUT "= ";Y(I)
200 YI(I) = Y(I)
205  NEXT
210 XP(1) = XB:P1(1) = Y(1):P2(1) = Y(2): REM FIRST PRINTED VALUES
215 J = 1: REM NUMBER OF PRINTED VALUES
220 X = XB:XP = XB: REM INITIALIZED BEGINING X AND XP
225 XP = XP + IP: REM NEXT VALUE TO BE PRINTED
230 :
235 :
300  REM ** CALCULATE YN+1 *************************
305  GOSUB 1000: REM CALCULATE MO FROM (XN,YN)
```

(Continued)

```
310  FOR E = 1 TO N
315 YN(E) = Y(E):M(E) = F(E):Y(E) = YN(E) + IC * F(E) / 2
320  NEXT
325 X = X + IC / 2
330  GOSUB 1000: REM CALCULATE M1 FROM (XN+IC/2,YN+(M0/2)*IC)
335  FOR E = 1 TO N
340 M(E) = M(E) + 2 * F(E): REM (M0+2*M1)
345 Y(E) = YN(E) + IC * F(E) / 2
350  NEXT
355  GOSUB 1000: REM CALCULATE M2 FROM (XN+IC/2,YN+(M1/2)*IC)
360  FOR E = 1 TO N
365 M(E) = M(E) + 2 * F(E): REM (M0+2*M1+2*M2)
370 Y(E) = YN(E) + F(E) * IC
375  NEXT
380 X = X + IC / 2
385  GOSUB 1000: REM CALCULATE M3 FROM (X+IC,YN+(M2)*IC)
390  FOR E = 1 TO N
395 Y(E) = YN(E) + (M(E) + F(E)) * IC / 6: REM CALCULATE YN+1
400  NEXT
405  IF  ABS (X - XP) < .0001 OR X > XP THEN  GOSUB 800: REM  CLOSE ENOUGH
     !-- NOW SAVE VALUES
410  IF X < XE GOTO 300
415 :
420 :
500  REM ** PRINT RESULTS **************************
505  PRINT  CHR$ (4);"PR#1"
510  PRINT  CHR$ (9);"80N": REM NOT REQUIRED ON SOME PRINTERS
515  PRINT : PRINT "4TH-ORDER RUNGE-KUTTA SOLUTION TO THE FOLLOWING 1ST-OR
     DER SYSTEM:"
520  LIST 1001,1998
525  PRINT "INITIAL CONDITIONS:"
530  FOR E = 1 TO N
535  PRINT "  Y(";E;")= ";YI(E)
540  NEXT
545  PRINT
550  PRINT "X INTERVAL ";XB;" TO ";XE
555  PRINT "X INCREMENT (CALC)= ";IC
560  PRINT "X INCREMENT (PRINT)= ";IP
565  PRINT
570  PRINT "Y(1) MIN= ";N1
575  PRINT "Y(1) MAX= ";X1
580  IF N = 1 GOTO 600
585  PRINT
590  PRINT "Y(2) MIN= ";N2
595  PRINT "Y(2) MAX= ";X2
600  PRINT
605  IF X1 = N1 THEN F1 = 0: GOTO 615
610 F1 = 32 / (X1 - N1): REM TAB FACTOR FOR Y(1)
615  IF X2 = N2 THEN F2 = 0: GOTO 625
620 F2 = 32 / (X2 - N2): REM TAB FACTOR FOR Y(2)
625  PRINT "     X             Y(1)          Y(2)                PLOT Y(1) A
     ND Y(2)"
630  PRINT "-------------------------------------------------------------
     ------------------"
635  FOR I = 1 TO J
640  PRINT XP(I);
645  HTAB (15): PRINT P1(I);
650  IF N <  > 1 THEN  HTAB (31): PRINT P2(I);
655 T1 =  INT ((P1(I) - N1) * F1)
660 T2 =  INT ((P2(I) - N2) * F2)
665  IF N = 1 THEN  POKE 36,T1 + 47: PRINT "1": GOTO 685
670  IF T1 < T2 THEN  POKE 36,T1 + 47: PRINT "1";: POKE 36,T2 + 47: PRINT
     "2": GOTO 685
675  IF T2 < T1 THEN  POKE 36,T2 + 47: PRINT "2";: POKE 36,T1 + 47: PRINT
     "1": GOTO 685
680  POKE 36,T1 + 47: PRINT "*"
685  NEXT
690  PRINT : PRINT
695  PRINT  CHR$ (4);"PR#0"
700  END
705 :
710 :
800  REM ** SUBROUTINE- SAVE X, Y(1), AND Y(2) ****
805 J = J + 1: REM COUNT NUMBER OF ORDERED PAIR TO BE PRINTED
810 XP(J) =  INT (X * 10000 + .5) / 10000:P1(J) = Y(1):P2(J) = Y(2)
815  IF P1(J) < N1 THEN N1 = P1(J): REM COMPARE FOR MINIMUM P1
820  IF P1(J) > X1 THEN X1 = P1(J): REM COMPARE FOR MAXIMUM P1
825  IF N = 1 GOTO 840
830  IF P2(J) < N2 THEN N2 = P2(J): REM COMPARE FOR MINIMUM P2
835  IF P2(J) > X2 THEN X2 = P2(J): REM COMPARE FOR MAXIMUM P2
840 XP = XP + IP: REM INCREMENT XP BY IP
845  RETURN
850 :
855 :
1000  REM ** SUBROUTINE- FIRST ORDER SYSTEM ********
1001  REM EXAMPLE PROBLEM
1002  REM  Y1'' + Y1' + Y1 = SIN(X)   (EQ. 5)
1100 F(1) = Y(2): REM  (EQ. 7)
1200 F(2) =  - Y(1) - Y(2) +  SIN (X): REM  (EQ. 8)
1999  RETURN
```

right of column 47 that the points Y(1) and Y(2) should be plotted. Lines 670 through 680 determine which value (Y(1) or Y(2)) should be plotted first. If T1 = T2 then an asterisk will be printed in this position (line 680).

In Applesoft, the HTAB command does not seem to work for any value greater than 40 when using a printer. This is the reason for using the POKE command in lines 665 through 675. Lines 650 and 665 are used to handle first-order equations, in which Y(2) is not calculated or plotted.

Program Operation

Operation of this program is straightforward. To illustrate this, we will solve equation 5 (mentioned earlier). But first, let's relate this equation to some physical phenomenon.

The movement of a suspended mass-spring system obeys this equation. Let's assume we have an object suspended from a spring to which we are applying a force. Furthermore, assume that there exists a dampening force which is proportional to the velocity of the mass. This dampening force is usually exerted by a dashpot mechanism. The general equation then becomes:

Equation 9

$$m*Y'' + c*Y' + K*Y = F(X)$$

where,

Y'' = acceleration of mass
Y' = velocity of mass
Y = position of mass
m = mass in slugs (1bm/32)
c = dampening constant (1bf/ft/s)
k = spring constant (1bf/ft)
F(X) = external force

Now equation 5 has physical significance. It describes the movement of a mass-spring system where:

m = 1 slug (32 1bm)
c = 1bf/ft/s
k = 1bf/ft
F(X) = sin(X)

Note that *X* is actually time in seconds.

Once the program is loaded, it is first necessary to delete lines 1001 through 1998. This clears the system of first-order equations. Next, the new system of first-order equations will be entered on these lines. In this example equations 7 and 8 would be entered as shown in the listing, lines 1100 and

1200. Lines 1001 and 1002 are for documentation purposes. Now we are ready to run the program.

In this example the number of first-order equations will be two. (Y(1) and Y(2) will be calculated in the *X*-interval 0 through 7. Next, the increment for calculations will be set to 0.1. In general, the smaller the value of IC, the more accurate the calculations. However, for this program IC should be no smaller than 0.001. This will prevent excessive roundoff errors when calculating X and will also shorten the run time.

The next value requested by the program is the increment at which we would like X, Y(1), and Y(2) to be printed. The value 0.2 was selected for this example. We are now ready to enter the initial conditions.

In this example the suspended object will start at rest. Thus Y(1) (position) will be entered as zero. Likewise Y(2) (velocity) will be entered as zero. In less than a minute the printer will begin printing the results.

One important item should be mentioned concerning the graphs of Y(1) and Y(2). Except in special cases, these two graphs are not superimposable, for two reasons. First, the values Y(1) and Y(2) are not scaled equally. Second, the graphs have been translated along the Y-axis, so the points Y(1) = 0 and Y(2) = 0 will not be plotted at the same location on the paper.

As mentioned earlier, many ordinary differential equations are difficult to solve and do not yield a solution in a closed form. The above example, however, is easily solved and does yield a solution in a closed form. Without showing the intermediate steps, the particular solution to equation 5, given the initial conditions, is as follows:

$$Y(1) = \frac{1}{\sqrt{3}} e^{-\frac{1}{2}x} \sin\left(\frac{\sqrt{3}}{2}x\right) + e^{-\frac{1}{2}x} \cos\left(\frac{\sqrt{3}}{2}x\right) - \cos(x)$$

Using this closed form of Y(1), I have calculated Y(1) at various points along the same X-interval specified in the sample run above. Table 2 compares the values of Y(1) attained by using both methods. In addition, the error introduced by using the fourth-order Runge-Kutta method has been calculated to four significant digits.

***Table 2:* Comparison Y(1) (Runge-Kutta Method) with Y(1) (closed form).**

X	YR	YC	%ERROR
0	0	0	0
0.2	1.26418842E-03	1.2641776E-03	0.0009
0.4	9.52598452E-03	9.5260127E-03	−0.0003
0.6	0.0300806999	0.030080796	−0.0003
0.8	0.0662560792	0.066256253	−0.0003
1.0	0.119397604	0.119397847	−0.0002
2.0	0.566721104	0.566721202	0.0000
3.0	0.865638477	0.865637729	0.0001
4.0	0.500521919	0.500520853	0.0002
5.0	−0.358252617	−0.358252752	0.0000
6.0	−0.962461007	−0.962459781	0.0001
7.0	−0.728262772	−0.728261216	0.0002

*%ERROR = 100 * (YR − YC)/YC*

where,

YR = Y(1) calculated by Runge-Kutta method

YC = Y(1) calculated by closed form

Table 2 demonstrates that the fourth-order Runge-Kutta method for solution of ordinary differential yields very accurate results, with minimal effort. For those interested in the derivation of this method, the references listed below should be consulted.

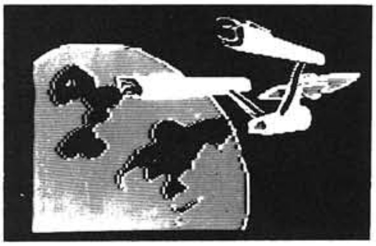
References

1. B. Carnahan, H.A. Luther, and J.O. Wilkes, *Applied Numerical Methods*, Wiley, New York, 1969.
2. W.E. Boyce and R.C. DiPrima, *Elementary Differential Equations*, Wiley, New York, 1977.

MICRO

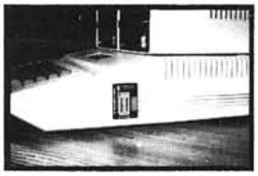
VersaWriter & APPLE II:
The Keys to Unlimited Graphics

DRAWING TABLET
Although VersaWriter operates on a simple principle, it produces graphics which match or exceed those of other digitizers. Rugged construction, translucent base, easy to use — plugs directly into APPLE II.



GRAPHICS SOFTWARE
Easily the most capable and complete graphics software for the home computer available. Fast fill drawings in 100 colors. All text in five sizes, compile and display shapes, edit, move and much more!

UNIQUE OFFER
See VersaWriter at your local dealer and pick up a copy of our demonstration disk. The complete VersaWriter hardware and software package is a real bargain at $299. For more information call or write:



EZ Port Will Solve Your Game I/O Problem!

How many times have you gone through the hassle of changing from game paddles to joystick, VersaWriter, or any other device using the game I/O? First, you have to remove whatever is sitting on top of the Apple–a video terminal, disk drives, printer, etc.

Next you remove the computer cover and try to see what you're doing as you switch plugs to the I/O. Then you replace the computer cover and whatever was on top of the Apple.

After all this, you find that you can't run the program because the I/O device is plugged in backwards or is 'off by a pin'.

Sound familiar?

EZ PORT GAME I/O EXTENDER FOR APPLE II

WHAT IS EZ PORT?

EZ Port is a specially designed extension unit for the Apple game I/O port. It's a board with a socket and a two foot long cable which plugs into the internal I/O port. You attach EZ Port wherever you prefer on the outside-on the side, the back, or on top.

EZ Port has a ZIP DIP II socket (ZIP=zero insertion force). These sockets are meant to be plugged into many times and will not wear out like ordinary sockets. All you do is plug in the appropriate device (joystick, paddles, etc.) and flip the switch to the ON position. No pressure is exerted on the 16-pin plug until you switch, so all the connectors will last longer, too!

EZ PORT MAKES GAME I/O CHANGES CONVENIENT, QUICK & SAFE.

Suggested price **$24.95**
Ask your local computer retailer for EZ Port, or contact:

VersaComputing, Inc.
3541 Old Conejo Rd. Suite 104
Newbury Park, CA 91320
(805) 498-1956

Dealers inquiries welcome.

Apple and Apple II are registered trademarks of Apple Computer, Inc.

ZIP DIP is a trademark of Textool Products, Inc.

Versa Computing, Inc.

3541 Old Conejo Road, Suite 104
Newbury Park, CA. 91320 **(805)498-1956**

Legrange Interpolating Polynomial

Use this routine to fit a curve to your data. Runs on any machine with floating point BASIC.

Paul H. Muller
331 Mackay Ave.
Ventura, California 93004

Technical literature today is crowded with undocumented correction coefficients or "flywheel factors." They are the interface between theory and the real world. If they were easily understandable, then they would be logically derived with an appropriate explanation.

The problem facing scientific programmers is the reduction of such empirical data to a readily usable mathematical function. Many graphs or tables that appear in the literature are resolved by using the standard techniques of analytical geometry and statistics. But there are some which will make you a candidate for the rubber room. Fortunately, there is a mathematical tool called the Legrange interpolating polynomial which can be used to approximate even the most bizarre-looking functions. It is a technique that requires less than 1K of memory, yet will produce surprisingly accurate results.

The Legrange polynomial is based on the idea that by knowing the coordinates of *n* number of points, you can deduce the coefficients of a polynomial of *n*-1 degree which must pass through those coordinates. That polynomial can then be used to estimate the value of the function between the known points.

Use of the Legrange polynomial can be illustrated by the interpolation of the sine function of any angle between 0 and 90 degrees, given the actual sines for just four angles. Of course this technique will work with any set of data and is not limited to the approximation of trigonometric functions. For this example, assume you know only the following:

Degrees (x)	Sine (y)
0.0	0.0
30.0	0.5
60.0	0.866025
90.0	1.0

For any other angle, designated x', the sine function for that angle, designated y', can be estimated using the Legrange polynomial as follows:

$$y' = y_1 L_1 + y_2 L_2 + y_3 L_3 + \dots y_n L_n$$

In this case:

$$x_1 = 0.0 \qquad y_1 = 0.0$$
$$x_2 = 30.0 \qquad y_2 = 0.5$$
$$x_3 = 60.0 \qquad y_3 = 0.866025$$
$$x_4 = 90.0 \qquad y_4 = 1.0$$

For this example, we will find the interpolated value of the sine of 45 degrees. Thus, x' = 45.0.

Where i = 1 to *n*, L_i is calcualted in figure 1.

Figure 1

$$L_1 = \frac{(x'-x_2)(x'-x_3)(x'-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} = \frac{(45\text{-}30)(45\text{-}60)(45\text{-}90)}{(0\text{-}30)(0\text{-}60)(0\text{-}90)} = -0.0625$$

$$L_2 = \frac{(x'-x_1)(x'-x_3)(x'-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} = \frac{(45\text{-}0)(45\text{-}60)(45\text{-}90)}{(30\text{-}0)(30\text{-}60)(30\text{-}90)} = 0.5625$$

$$L_3 = \frac{(x'-x_1)(x'-x_2)(x'-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)} = \frac{(45\text{-}0)(45\text{-}30)(45\text{-}90)}{(60\text{-}0)(60\text{-}30)(60\text{-}90)} = 0.5625$$

$$L_4 = \frac{(x'-x_1)(x'-x_2)(x'-x_4)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)} = \frac{(45\text{-}0)(45\text{-}30)(45\text{-}60)}{(90\text{-}0)(90\text{-}30)(90\text{-}60)} = -0.0625$$

A more concise way to define the value of L_i uses the product sign $\prod$ as follows:

$$L_i = \prod_{\substack{j=1 \\ i \neq j}}^{n} \frac{(x' - x_j)}{(x_i - x_j)}$$

Thus to arrive at a value for y':

$$y' = 0\,(-0.0625) + .5\,(.5625) + .866025\,(.5625) + 1\,(-0.0625)$$

$$y' = 0.7059$$

The actual value for the sine of 45 degrees is given in most references as 0.7071, giving an error of 0.0012 on the interpolated value. The sine value for other angles could be similarly estimated.

Listing 1 is a BASIC program which automates the Legrange technique. The program was originally written for an OSI Superboard, but should run on any BASIC system with only minor modifications. The maximum number of known coordinates that can be entered into the program is arbitrarily set at 25, but more can be accommodated by changing the dimensioned size of the X and Y strings, and by changing the "TO" value in the data entry loop. The program in listing 1 consumes 930 bytes, and will run in about 12 seconds on the 6502-based Superboard when 25 data points are used.

The program first queries for the x,y coordinates of the known points. Any number of pairs up to 25 may be entered. To get out of the data entry routine, simply input END,END in response to the query. The program then asks for x', and displays the interpolated value y' a few seconds later. To generate another interpolation, enter R in response to the program query. Entering E at this point will exit the program. To change or add data points, the program must be run from scratch. The results of running the sine interpolation example are shown in the sample run.

There are a few guidelines to follow for best results. It is essential to understand that the program generates a polynomial that will exactly intersect only the given points. The assumption is that the interpolated points will closely (but not exactly) fit the curve of the unknown function. When using the program to emulate a graphical flywheel factor, it is best to plot a few interpolated points on the same graph to see how well the program is predicting the actual values. You may need to adjust the data points to achieve a more precise fit. The more rapidly the instantaneous slope of a function changes, the greater the number of data points needed to obtain a good interpolation.

Take a piece of graph paper and draw a smooth continuous function, freehand. (Only one value of y for each x; no fair doubling the curve back on itself!) Take about four coordinates spaced equally along the curve and enter them into the program. The interpolations for intermediate values will surprise you with their accuracy.

This program is intended only to demonstrate the basic method of using the Legrange polynomial. It can be easily adapted as a subroutine for larger programs where the known points could be taken from DATA statements. There are intriguing possibilities for systems with advanced graphics. Also, integrals and roots may be estimated for functions where there are only a few known data points.

Sample Run

```
RUN

LEGRANGE POLYNOMIAL

ENTER X 1 , Y 1
0,0
ENTER X 2 , Y 2
30,.5
ENTER X 3 , Y 3
60,.866025
ENTER X 4 , Y 4
90,1
ENTER X 5 , Y 5
END,END
ENTER X' 45

Y' = 0.705889

RUN AGAIN OR EXIT? R

ENTER X' 25



Y' = 0.423322

RUN AGAIN OR EXIT? E
  END
```

Listing 1

```
 10 REM ***** LEGRANGE INTERPOLATING POLYNOMIAL ****
 20 REM ***** BY PAUL H. MULLER - JULY 1981      ****
 30 DIM X(25),Y(25)
 40 N=0
 50 PRINT
 60 PRINT" LEGRANGE POLYNOMIAL"
 70 PRINT
 80 FOR I=1 TO 25
 90 PRINT" ENTER X";I;",";"Y";I
100 INPUT X$,Y$
110 IF X$="END" THEN 160
120 X(I)=VAL(X$)
130 Y(I)=VAL(Y$)
140 N=N+1
150 NEXT I
160 PRINT
170 PRINT
180 INPUT" ENTER X'";XP
190 F=0
200 FOR I=1 TO N
210 S=1
220 D=1
230 FOR J=1 TO N
240 IF J=I THEN 270
250 S=S*(XP-X(J))
260 D=D*(X(I)-X(J))
270 NEXT J
280 L=S/D
290 F=F+(Y(I)*L)
300 NEXT I
310 FOR I=1 TO 5
320 PRINT
330 NEXT I
340 PRINT" Y' =";F
350 PRINT
360 PRINT
370 INPUT "RUN AGAIN OR EXIT";C$
380 IF C$="R" THEN 160
390 IF C$="E" THEN 410
400 GOTO 350
410 END

Note:  VAL(X$) converts the string variable X$ to its
               numerical value
```

MICRO

SIN(X) The Hard Way

Microsoft BASIC uses a series expansion formula to calculate the sine of an angle. The logic of this machine language routine is emulated here in a BASIC program.

Earl Morris
3200 Washington
Midland, Michigan 48640

Nearly every BASIC interpreter has built-in trigonometric functions. A simple call, Z = SIN (X), magically produces the sine of the angle X. This function is very useful in plotting intricate patterns and in games for finding the trajectory of phasers, rockets, bombs, and the like. This article probes the algorithm used by Microsoft BASIC to calculate the sine function. A way to increase the speed of the sine routine is suggested if some loss of accuracy can be tolerated.

Before we look at the programming, we must understand the mathematics. The sine of X is defined by an infinite series expansion.

$$\text{SIN}(X) = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!}$$

to infinity

This equation is valid for all values of X, but the equation has an infinite number of terms. It is difficult, even for a computer, to add up all the terms. Any desired accuracy can be obtained if enough terms in the series are used. How many terms are enough? The answer depends on the magnitude of X. The series converges quickly for small values of X, but more slowly as X becomes larger. If $X << 1$, then X to a positive power rapidly becomes vanishingly small. For example if $X = 0.1$, then $X^3 = .001$ and $X^5 = .00001$. All the terms except the first can be ignored, leaving

$$\text{SIN}(X) = X$$

This simplest approximation begins to fail as X is increased above .4. The following table shows the actual values of sine for small X.

SIN (.05) = .04998
SIN (.1) = .09983
SIN (.2) = .19867
SIN (.4) = .38942

If X = 1, then in the series expansion all the terms $X^n = 1$. However, each higher order term is becoming smaller due to the N! in the denominator. In the 5th term, 9! = 362880 so that this term adds 1/9! = .0000027 to the sine. Higher order terms can certainly be ignored.

If $X >> 1$, then the X^n in the numerators can also be large. If X = 10, then the 5th term becomes 1,000,000,000/362880 and beyond the ability of my pocket calculator to carry enough significant decimal places. Eventually the N! in the denominator will be greater than the X^n in the numerator and any further terms will become insignificant. However, every term in the sum must be calculated to be accurate to as many decimal places as you wish in the final sine value. For large values of X, this becomes impossible in a practical sense.

Since the sine is a periodic function, several tricks are used to shorten the amount of calculation involved. Large values of X can be avoided by using the trigonometric identity $\text{SIN}(X + 2\pi) = \text{SIN}(X)$. That is, any angle greater than one revolution can be reduced by multiples of 2π without affecting the sine. Thus the argument X can always be reduced to less than 6.28 or 2π. Using the additional relationship $\text{SIN}(X) = \text{SIN}(\pi/2 - X)$ the argument can be further reduced to $-\pi/2 < X < \pi/2$. Thus the sine of any angle can be expressed as the sine of an angle between -90^0 and 90^0. Since X must always be reduced to less than 1.57, the sine can be calculated to better than six-digit accuracy by using only the first five terms of the infinite series.

The form of the sine equation given above is fine for human use, but a little rearranging is necessary for an efficient computer routine. First a change in variables is made by substituting $Y = X/2\pi$.

$$\text{SIN}(X) = 2\pi Y - \frac{(2\pi Y)^3}{3!} + \frac{(2\pi Y)^5}{5!} - \frac{(2\pi Y)^7}{7!} + \frac{(2\pi Y)^9}{9!}$$

Then, substituting the numerical value for π and evaluating the factorials gives

$$\text{SIN}(X) = A{*}Y + B{*}Y^3 + C{*}Y^5 + D{*}Y^7 + E{*}Y^9$$

A = 6.2831
B = −41.3417
C = 81.6052
D = −76.7058
E = 42.05869

Again, for the benefit of the computer, the equation is rearranged to give

$$\text{SIN}(X) = Y(A + Y^2(B + Y^2(C + Y^2(D + Y^2(E)))))$$

This rather strange equation is very neatly solved by a programming loop. Starting with the innermost value E, the next term is always found by multiplying by Y^2 and adding the next constant. This procedure is repeated for as many terms in the series as are desired. The final step is to multiply by Y.

Following is a BASIC program to calculate the sine of an angle by the logic described above. The value found is compared to your built-in sine routine. The two should be identical.

Lines 60 to 110 divide the argument by 2π and take the fractional part of the answer. This reduces the angle to less than one revolution. Lines 120 to 220 reduce the angle to between -90^0 and $+90^0$. The reduced argument is stored in A4 while its square is stored in A8. Lines 260 to 350 add up the terms of the series expansion. The number of terms added is controlled by the variable B1.

With some sacrifice in accuracy, the sine routine can be quickened by computing fewer terms in the series. In the BASIC program this is done by changing the loop counter from "4" to "3" and deleting the next piece of data (39.7109). The loop counter can be decreased to 2 and then to 1 with further loss of accuracy. Table 1 was generated using from one to five terms in the sine equation.

Note that the worst loss in accuracy is at the largest value of X. Even the three-term approximation of sine is accurate to better than 1%. For most games and even plotting high-resolution patterns, this accuracy is sufficient. However, you will not increase the speed of your program by using this BASIC program to calculate sines. If you understand the logic of the machine language sine routine, you can relocate it into RAM and change the loop counter to increase speed or accuracy as needed.

Table 1

(X)	Number of Terms in Equation: 5	4	3	2	1
.1	.09983	.09983	.09983	.09983	.1000
.4	.38942	.38942	.38942	.38933	.4000
.8	.71736	.71736	.71740	.71467	.8000
1.2	.93204	.93203	.93274	.91200	1.200
1.5	.99749	.99740	1.00078	.9375	1.500

The BASIC program follows exactly the same logic as the machine sine routine in OSI ROM BASIC. The variables A1 and A2 correspond to the primary and secondary floating point accumulators. Data must be moved to these registers before any mathematical operations can be done. Thus the BASIC program is written in a rather strange fashion to simulate the machine code.

(Continued on page 28)

Not Just Software But The Company Behind It. . .

SDS™

INDUSTRY PIONEERS

Southwestern Data Systems has been a pioneer in producing innovative software for the Apple II since its inception in the late 1970's when they entered the software marketplace with the introduction of Roger Wagner's APPLE-DOC and THE CORRESPONDENT. Mounting sales and interest contributed to the company's expansion including a truly professional staff, combining quality and reliability with both product and support perpetuating their special insight into realizing and satisfying consumer needs.

From the outset, they established a MONEY-BACK GUARANTEE as evidence of their dedication to supplying only the finest quality products available. Software of such uncompromising quality became the cornerstone for their success and is the reason for their recognition as an industry leader today.

SOFTWARE SPECIALISTS

Minimal scrutiny will reveal SDS software to be the ripest, juiciest fruit on the proverbial apple software tree. Succulent appetizers like A.C.E., Z-TERM, APPLE-DOC, ASCII EXPRESS and LISTMASTER have already established SDS as the UTILITY AND COMMUNICATIONS SOFTWARE SPECIALISTS! But that's only the beginning! SDS is bursting at the seams with a bumper crop of quality software just recently made available! They're sensational! Have a byte of:

ASCII EXPRESS — The Professional: The advanced version of the classic communications package.

THE ROUTINE MACHINE: Allows anyone to put machine language in their Applesoft programs.

DOUBLETIME PRINTER: Print files while **simultaneously** running another program.

MUNCH-A-BUG: Easily de-bug machine language programs.

MERLIN: A full featured 6502 macro assembler.

UNIVERSAL GRAPHICS: Now you can print any Hi-Res image on almost any printer.

FINANCIAL MANAGEMENT SYSTEM II: The best home and small business accounting package available.

NORAD: A fast moving Hi-Res arcade game.

These recent releases not only include superior utility and communications software but illustrate the SDS advance into the business and education computer game fields. The full line of SDS products are professionally and attractively packaged and should be available to the retail customer for inspection through your local dealer, or write to SDS for your complementary product guide. Dealers should contact Joanne Johnson regarding promotional materials and software samplers.

THE PEOPLE

SDS is more than just a list of products though. It's the people that make the difference. Together, the entire staff at SDS will continue to offer you, the software user, the highest quality product available.

ROGER WAGNER, founder and president of Southwestern Data Systems, is a person of integrity and reputation, and has established himself as an authority in his field. In addition to the successful programs he has to his credit, he is also author of the highly acclaimed ASSEMBLY LINES articles in Softalk, now available in convenient book form. A firm believer in quality assurance, Roger stakes his reputation on every product he endorses. Unless he is confident of its performance and certain that it is the best product of its kind available, it will NOT carry the SDS name, shielding both his company and its customers from the tedium involved with marketing second class merchandise.

A.

A. ROGER WAGNER
SDS Founder & President, Author.

B. SDS products are handsomely packaged, supported by extensiv advertising and (C.) are availabl from retail software outlets nationwide.

D. SDS personnel explore every aspe of a potential product's marketin prior to making any publication commitments.

E. Office Manager, Tom Burns revie a software package for accuracy.

F. Vice President Joanne Johnson provides the personal touch, kee customers & dealers up to date o the latest SDS projects.

It's the total involvement in every aspect of the software industry th makes SDS unique.

E

southwestern

DESIGN: DOUG WESTERKAMP/PHOTOGRAPHY: IAN CUMMINGS

(Continued from page 25)

The variables A4, A8 and B1 represent page zero addresses. The machine code is stored in ROM starting at $BC03, and the data table begins at $BC7C. The data is stored in four-byte floating point format (except for the loop counter "4"). The data table can be read by using the following trick: Cold start BASIC and in immediate mode enter AA = 1. Then jump to the monitor and look at the hex data stored at $0303 and beyond. You will find

$0303	41	= "A"
$0304	41	= "A"
$0305	81	This is the floating
$0306	00	point expression
$0307	00	for 1
$0308	00	

The first four bytes in the sine data table at $BC7C are 83 49 OF DB. Put this hex data into memory starting at $0305. Then warm start BASIC and PRINT AA. The value of AA is now 6.28319 or the first value in the data table of the BASIC program. The remainder of the data table can be decoded in a similar fashion. Be careful of the single byte "4" at $BC84. Those of you with sharp eyes will note the value for "E" from the theoretical equation does not exactly agree with the value in OSI's data table (42.0 *vs.* 39.7).

MICRO

```
10  REM  SINE ROUTINE
15  REM
20  REM  BY EARL MORRIS
25  REM
30  CLEAR
40  INPUT "ARGUMENT FOR SINE";A1
50  PRINT  SIN (A1);: REM  USE INTERNAL SINE
60 A2 = A1
70  READ A1: REM  GET 2*PI
80 A1 = A2 / A1: REM  DIVIDE BY 2 PI
90 A2 = A1
100 A1 =  INT (A1)
110 A1 = A2 - A1: REM  TAKE FRACTIONAL PART
120  READ A2: REM  GET .25
130 A1 = A2 - A1
140  IF A1 >  = 0 THEN FLAG = 1
150  IF FL = 1 THEN 180
160 A1 = A1 + .5
170  IF A1 < 0 THEN 190
180 A1 =  - A1
190  RESTORE : READ A2: READ A2: REM  USE .25 AGAIN
200 A1 = A1 + A2
210  IF FL = 1 THEN 230
220 A1 =  - A1
230 A4 = A1: REM  FIRST QUADRANT ARGUMENT
240 A1 = A1 * A4
250 A8 = A1: REM  ARGUMENT SQUARED
260  READ B1: REM  TERMS IN SERIES EXPANSION
270  READ A2: REM  GET COEFFICIENT
280 A1 = A1 * A2
290  READ A2: REM  GET COEFFICIENT
300 A1 = A1 + A2
310 A2 = A8: REM  GET ARG SQUARED
320 B1 = B1 - 1
330  IF B1 <  > 0 THEN 280
340 A2 = A4: REM  GET ARG
350 A1 = A1 * A2
360  PRINT A1: REM  PRINT CALCULATED SINE
370  GOTO 10
380  DATA 6.283185,.25,4,39.7109,-76.575,81.6022
390  DATA -41.3417,6.283185
```

for fast development of fast, tight programs...

step beyond FORTH, to

RPL

High speed, low memory requirements, and user-friendly development tools are no longer mutually exclusive. **R**everse **P**olish **L**anguage, a FORTH-like language now available for the PET and CBM computers, is faster than FORTH, easier to debug than BASIC, and more space-efficient than any other language known, including assembly language. Here's what **Loren Wright**, MICRO magazine's PET Vet, says about it:

> **"RPL is generally faster and more conservative of memory than FORTH . . . RPL will serve well the need for a language that is faster than BASIC yet easier to program than assembly language. The package is well-thought-out and well-documented."**

RPL uses the ordinary Commodore BASIC screen editor for program entry and editing. And the full power of BASIC, in both immediate and program modes, remains available to the user throughout a development session. The RPL Compiler and Symbolic Debugger reside in the top 8K of memory, ready to be invoked at any time, directly from BASIC, via the commands "compile" and "debug". RPL source code is saved to disk or cassette just like BASIC source, and is compiled memory-to-memory for quick compilation turnaround and instant source accessibility. RPL supports separate compilation of program modules through the use of the compiler's "global symbol" features, which also permit the development of true "subroutine libraries".

The language itself is concise and straightforward, making it much easier to learn and master than most other computer languages. A total of only 47 special keywords and symbols provide the following capabilities:

- Nestable, multi-line IF . . . THEN . . . ELSE constructs.
- Nestable FOR . . . NEXT loops.
- Named subroutines and functions of arbitrary length.
- Compile-time constants and code ORGability.
- Full 16-bit integer arithmetic and logical manipulations.
- Built-in character-string handling.
- Stack-management directives including n-index, n-rotate.
- GET, INPUT, and PRINT operators
- Forward and backward symbolic references, including GOTO.
- Easy access to machine language.
- Predefined arrays with numeric and/or string contents.
- Local and global symbols.

. . . and *much* more. The 60-page RPL manual is clear and well-organized, making the language easy to learn and easy to use: **Loren Wright** says that **"the documentation is about the best I have ever seen."**

The Samurai RPL Symbolic Debugger is a screen-oriented, object-level debug facility using a soft-key-driven command syntax for ultra-ease of use. Features included are:

- Full visibility into both stacks at all times.
- Single-stepping, with source-level next-step display.
- Breakpointing in both auto-single-step and "go" modes.
- Address specification using expressions with symbols.
- Stack-edit capability on both stacks.
- Debugger video usage is transparent to target program.
- Extra run-time error-checking during debugging only.

. . . and, of course, much more. Here's what **Robert Baker,** author of the PET-pourri column in Kilobaud Microcomputing, says about it:

> **"RPL offers an unbeatable combination of speed, memory space efficiency, and ease of use. It is well-designed, well-implemented, and well-documented, and it deserves the serious consideration of every PET/CBM programmer. The Samurai RPL Symbolic Debugger, in particular, must be seen to be believed."**

The compiler includes a special option making it very easy for you to create "execute-only" object modules from which all development-utility software and memory allocations have been excluded. The price you pay for the compiler also includes an unlimited license to resell the RPL "run-time library" (*not* the compiler) in conjunction with "execute-only" application object modules of your own.

The Samurai RPL Compiler is now available at the special introductory price of $49.95, which includes the manual in a nice 3-ring binder and First Class postage within the continental U.S. Media supplied is of top quality, and is *not* copy-protected (this permits you to make backups for yourself without hassles). Compiler and debugger together are **$80.91, complete.** Manuals are available separately at $10.00 and $4.00, respectively, and will be credited toward software purchase. Please specify machine type, memory size, ROM version, and media type (cassette, 4040, or 8050 diskette) when ordering.

Order anytime, day or night, 7 days a week

Outside Florida:

800-327-8965

(ask for ext. 2)

Within Florida: **305-782-9985**

VISA and Master Charge accepted

All orders shipped within 2 days of receipt

(For technical inquiries, please phone 305-782-9985)

For more information, or to order by check or money order, please write:

SAMURAI SOFTWARE
P.O. Box 2902
Pompano Beach, FL 33062

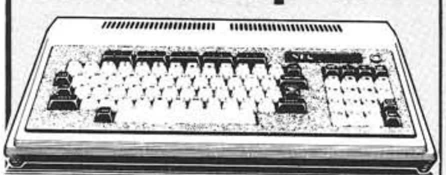
ORDER TOLL-FREE

consumer computers Mail Order DISCOUNTS

FREE SHIPPING*

*on all pre-paid cash orders

NEC Microcomputer



PC-8001A 32K Computer	CALL
PC-8012A I/O Unit w/32K RAM	CALL
PC-8013A Dual Mini-Disk Drive Unit	CALL
PC-8001 Multi Cardware (FDI/O & 32K)	CALL
CP/M 2.2 Operating System for NEC	129
WordStar configured for NEC	299
SuperCalc configured for NEC	279
NEC Wordprocessor & Accounting Software	CALL

Many more software packages and languages; (Pascal, Fortran, Cobol, etc) are available configured for the NEC 8001A Computer.
Please call or write for a product price list.

ATARI

ATARI 800 16K

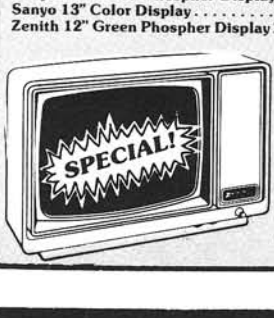
CALL FOR BEST PRICE

Atari 400 w/16K	349
410 Program Recorder	65
810 Disk Drive	449
825 80 col. 7x8 Dot matrix impact printer	699
822 40 col. Quiet Thermal Printer	349
850 Interface Module	159
Atari 16K Ram Module	69
Axlon Ramcram 32K Module	189
Asteroids, Missile Command and Star Raiders	35 ea.

SPECIAL ATARI CATALOG AVAILABLE

Video Monitors

Amdek/Leedex Video 100 12" B&W	149
Amdek/Leedex Video 100G 12" Green Phospher	179
Amdek (Hitachi) 13" Color w/audio output	389
NEC 12" Green Phospher Display JB-1201M	CALL
NEC 12" Lo-Res Color Display	CALL
NEC 12" Hi-Res RGB Color Display	CALL
Sanyo 9" B&W Display	185
Sanyo 9" Green Phospher Display	CALL
Sanyo 12" B&W Display	269
Sanyo 12" Green Phospher Display	285
Sanyo 13" Color Display	449
Zenith 12" Green Phospher Display ZVM-121	149



ZENITH 12" GREEN $149

S-100

California Computer Systems

Floppy Disk Controller	**$369**
64K Dynamic Ram Board, 200ns	**$499**
Z-80 CPU board w/monitor ROM	**$269**
16K Static memory board, 200ns	369
32K Static memory board, 200ns	599
S-100 12 Slot Mainframe	475
4-Port Serial Interface	299
2-Port Serial/2-Port Parallel Interface	299
4-Port Parallel Interface	229

commodore VIC20™

Personal Computer

Color · Sound · Graphics
Call or write for more info.
Disk drives available soon!

$259

Printers

Silentype
w/Apple II interface
$349

Epson MX-80 or MX-80 FT CALL

Anadex 9501 w/2K Buffer	1349
C. Itoh Starwriter 25 CPS daisywheel	1449
C. Itoh Starwriter 45 CPS daisywheel	1649
Epson MX-70	CALL
Epson MX-80 & MX-80 F/T	CALL
Epson MX-100	CALL
NEC 8023 Impact Dot Matrix	695
NEC Spinwriters (Latest models)	CALL
Paper Tiger IDS-445G w/graphics	699
Paper Tiger IDS-460G w/graphics	949
Paper Tiger IDS-560G w/graphics	1249
Silentype Printer w/Apple interface	349
Qume Sprint Daisywheels (Latest models)	CALL

ORDER TOLL FREE

800-854-6654

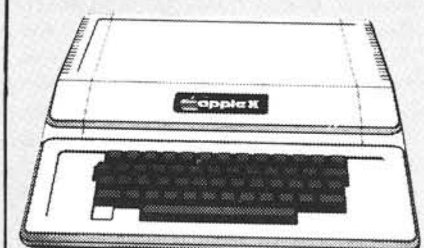
In California and outside continental U.S.

(714) 698-8088

Telex 695-000 Beta CCMO

apple computer

Authorized Dealer



APPLE II PLUS

16K's, 48K's, 64k's*

*48K Apple with 16K RamBoard

CALL FOR BEST PRICES

APPLE DISK DRIVES

DRIVE ONLY OR W/CONT & DOS 3.3

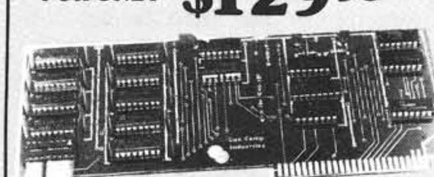
CALL FOR PRICES

SPECIAL APPLE CATALOG AVAILABLE

16K RAMBOARD *by ConComp*

for Apple II Computers

FOR ONLY **$129.95**



AVAILABLE NOW

Ordering information: Phone orders using VISA, MASTERCARD, AMERICAN EXPRESS, DINER'S CLUB, CARTE BLANCHE, bank wire transfer, cashier's or certified check, money order, or personal check (allow ten days to clear). Unless prepaid with cash, please add 5% for shipping, handling and insurance (minimum 5.00). California residents add 6% sales tax. We accept CODs. OEM's, Institutions and corporations please send for a written quotation. All equipment is subject to price change and availability without notice. All equipment is new and complete with manufacturer's warranty (usually 90 days). Showroom prices may differ from mail order prices.

Send Orders To:

consumer computers Mail Order

8314 Parkway Drive
La Mesa, Calif. 92041

MICRO

From Here to Atari

By James Capparell

More on the Disk

Last month we looked at the floppy disk and the structure that constitutes DOS II. This month I will discuss one problem that plagues some early disk drives and describe an inexpensive remedy. I will also include an assembly language program that lists disk directory files.

Disk drives purchased prior to fall 1981 are susceptible to a few problems — including frequent errors 144 and 138. These errors are due in part to the use of the Western Digital disk controller chip, 1771. This chip, when used without a Western Digital Data Separator (D.S.) independent of the chip, provides marginal performance.

The D.S. functions to separate clock pulses from data pulses. As a drive is used, mechanical parts begin to wear, heads get out of alignment, speed varies, and errors occur, especially on the inside tracks, sector 600 and above. Recently added programs will not reload without generating the 144 error. Atari has corrected this problem in drives that were shipped after fall '81. Built into the new drives are the necessary D.S. as well as power supply improvements and the new ROM formatter (discussed last month). You can determine that a drive is new by looking for the circular stickers with DS and C printed on them. There has been some indication that there will be a retrofit available to those with old drives.

Another solution is available — the Data Separator board from the Percom Company. Initially this board was designed for Radio Shack disk drives suffering from the same problems due to use of the same 1771 chip. The board is available for $30 from the Percom Company, 211 N. Kirby, Garland, Texas 75042. It is relatively easy to install if you are familiar with a soldering iron and are not afraid of integrated circuits. Since tinkering with your drives will void your three-month warranty, I'd advise you to wait three months after purchase before trying this procedure.

```
10 .TITLE " GET DISC DIRECTORY"
20 .PAGE " DIR.ASM"
30 ; THIS PROGRAM ACCESSES THE DISC DIRECTORY
40 ; AND PRINTS IT TO SCREEN
50 ; IT RESIDES IN PAGE 6 TO MAKE IT AVAILABLE TO BASIC
60 ; USE L OPTION IN DOS MENU TO LOAD FILE
65 ; CALL FROM BASIC X=USR(1536)
70 ;
80 OPEN=$03  OPEN COMMAND
90 CLOSE=$0C CLOSE COMMAND
0100 GETREC=$05       GET RECORD COMMAND
0110 PUTREC=$09       PUT RECORD COMMAND
0120 IOCB0=$00 INDEX FOR IOCB 0 ASSIGNED TO E:
0130 IOCB5=$50 INDEX FOR IOCB 5 ASSIGNED TO DISC
0140 CIOV=$E456       CIO ENTRY VECTOR
0150 EOF=$88  END OF FILE STATUS VALUE
0160 ICHID=$340       HANDLER I.D. SET BY CIO
0170 ICDNO=ICHID+1    DEVICE # SET BY CIO
0180 ICCOM=ICDNO+1    COMMAND BYTE
0190 ICSTA=ICCOM+1    STATUS BYTE SET BY CIO
0200 ICBAL=ICSTA+1    BUFFR ADR LOW
0210 ICBAH=ICBAL+1    BUFFR ADR HI
0220 ICPTL=ICBAH+1
0230 ICPTH=ICPTL+1
0240 ICBLL=ICPTH+1    BUFFR LEN LO
0250 ICBLH=ICBLL+1    BUFFR LEN HI
0260 ICAX1=ICBLH+1    AUX1
0270 ICAX2=ICAX1+1    AUX2
0280 ;
0290 ;
0300 *=$0600          PAGE 6
0310 PLA    CLEAR NULL VAL FROM BASIC USR FUNCTION
0320 LDX #IOCB5
0330 LDA #OPEN        OPEN FILE OR DIRECTORY
0340 STA ICCOM,X      COMMAND BYTE
0350 LDA #NAME&$FF    SET UP BUFFER POINTER
0360 STA ICBAL,X      TO POINT TO DIR SEARCH
0370 LDA #NAME/256    COMMAND D:*.*
0380 STA ICBAH,X
0390 LDA #$06         SETUP FOR INPUT
0400 STA ICAX1,X
0410 LDA #0
0420 STA ICAX2,X
0430 JSR CIOV         GO OPEN FILE
0440 BPL A05          EVERTHING OK
0450 BMI EXIT         ERR ON OPEN FILE
0460 A05 LDA #GETREC
0470 STA ICCOM,X
0480 LDA #PUTREC
0490 STA ICCOM        SETUP IOCB 0
0500 LDA #$6E
0510 STA ICBAL,X      BUFFER LOW
0520 STA ICBAL
0530 LDA #$6
0540 STA ICBAH,X
0550 STA ICBAH
0560 A10 LDA #$14     SET MAX, RECORD SIZE
0570 STA ICBLL,X
0580 STA ICBLL
0590 LDA #0
0600 STA ICBLH,X
0610 STA ICBLH
0620 JSR CIOV         READ ONE DIR RECORD
0630 BMI A20          EITHER EOF OR ERROR
0640 LDX #0           SETUP IOCB 0
0650 JSR CIOV         GO WRITE RECORD TO E:
0660 LDX #$50         RESET IOCB TO 5
0670 BNE A10          GO GET NEXT RECORD
0680 A20 CPY #EOF     DONE ?
0690 BNE EXIT         NO THIS WAS AN ERROR
0700 LDA #CLOSE       SHUTDOWN FILE
0710 STA ICCOM,X
0720 JMP CIOV         USE JMP HERE SO THAT RTS IN CIO
0730 ;                EILL RETURN TO BASIC
0740 EXIT RTS
0750 ;
0760 ;
0770 NAME .BYTE "D:*.*"
0780 .END
```

At the top corners of your disk drive you will locate the Phillips-head screws. Pry off the concealing tabs, loosen the screws, and lift off the plastic top. As you view your drive from the front and top you will see a long board on your left. It is mounted vertically and there is a sheet-metal box covering part of the circuitry. The 1771 chip is socketed under this metal box along with some other chips such as the ROM formatter. (The metal box is included for RFI shielding.)

Carefully disassemble the board from the motherboard, which lies flat to the rear of the drive. Mark all wires as you unplug them. Pay attention to the front-rear, and top-bottom orientation of plugs to be assured of correct reassembly. After the long board is unplugged from the motherboard and the metal box has been pried loose, locate the 40-pin integrated circuit marked 1771.

Now find the crystal which sits about two inches forward of the 1771. This crystal must be moved to make room for the Percom D.S. Unsolder the crystal and solder on longer leads. Return the crystal to its original location, but this time bend it forward. (The longer leads should allow this.) Carefully pry the 1771 out of its socket and insert it into the Percom D.S. circuit board in the orientation described in the board's instructions. Make sure every chip is properly seated again.

Following Percom's instructions, insert the new circuit board in place of the 1771. This board can really only be inserted logically in one direction, extending toward the front of the drive covering the crystal. Reassemble all boards and loose wires, taking care that orientations are correct. The metal box will not fit in its original location without cutting a notch for the newly moved crystal.

I recommend testing the disk before putting the cover on. You may need to reseat the chips again; I played with mine a couple of times before everything worked. Prior to installing this board I could not consistently read any sectors above about 600, the inner tracks. After installation everything worked like new.

Product Reviews

I have received the following products for review. They will be handled in more detail in MICRO's new "Reviews in Brief" department, which will begin in April.

Eastern House Software
3239 Linda Drive
Winston-Salem, NC 27106

1. Macro Assembler
2. Monkey Wrench utility ROM used in right-hand cartridge with BASIC

LJK Enterprises, Inc.
P.O. Box 10827
St. Louis, MO 63129

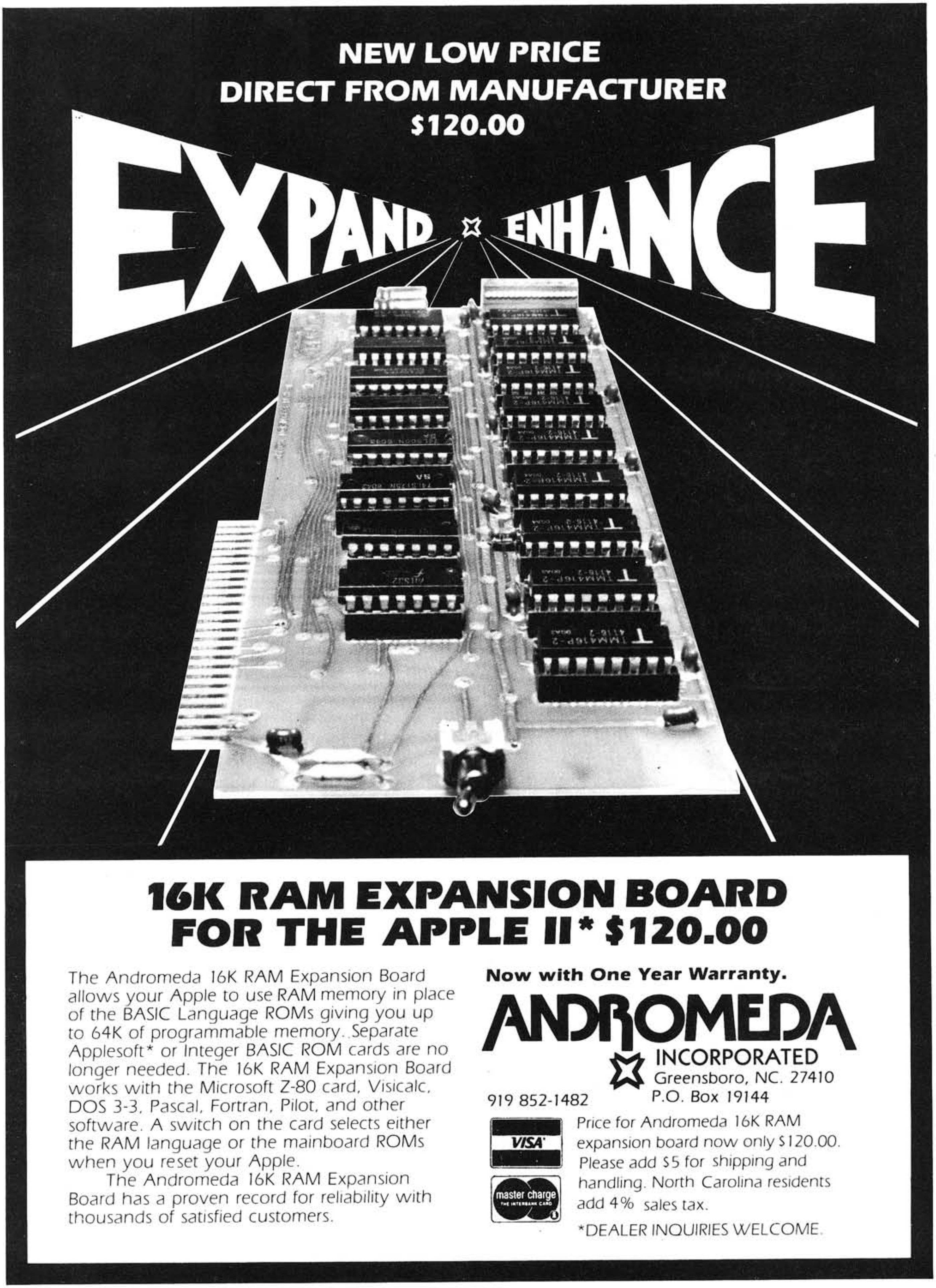
1. Letter Perfect word processor

Optimized System Software, Inc.
10379 C Lansdale Ave.
Cupertino, CA 95014

1. BASIC A+ Enhanced BASIC
2. OS/A+ Enhanced DOS

Please address all correspondence to the author at 297 Missouri, San Francisco, CA 94107.

MICRO


NEW LOW PRICE
DIRECT FROM MANUFACTURER
$120.00
EXPAND ENHANCE
16K RAM EXPANSION BOARD
FOR THE APPLE II* $120.00
The Andromeda 16K RAM Expansion Board allows your Apple to use RAM memory in place of the BASIC Language ROMs giving you up to 64K of programmable memory. Separate Applesoft* or Integer BASIC ROM cards are no longer needed. The 16K RAM Expansion Board works with the Microsoft Z-80 card, Visicalc, DOS 3-3, Pascal, Fortran, Pilot, and other software. A switch on the card selects either the RAM language or the mainboard ROMs when you reset your Apple.
The Andromeda 16K RAM Expansion Board has a proven record for reliability with thousands of satisfied customers.
Now with One Year Warranty.
ANDROMEDA
INCORPORATED
Greensboro, NC. 27410
P.O. Box 19144
919 852-1482
VISA
master charge
Price for Andromeda 16K RAM expansion board now only $120.00. Please add $5 for shipping and handling. North Carolina residents add 4% sales tax.
*DEALER INQUIRIES WELCOME.

ROCKWELL Microcomputers from Excert, Inc.

THE AIM 65/40
Single Board or Smorgasbord

- A full size terminal style keyboard w/8 special function keys
- A smart, 40 character display with its own microprocessor
- A 40 column printer w/text and graphic output
- Up to 64K of on-board RAM and ROM
- On-board interfaces include RS232, dual audio cassette and 2 user I/O R6522 devices
- Firmware includes interactive monitor and text editor w/options of Assembler, BASIC, FORTH and PL/65

THE AIM 65
Take-Out Order

- A full size terminal style keyboard w/3 special function keys
- A 20 character display
- A 20 column printer w/text and graphic output capability
- Up to 4K RAM and 20K ROM on-board
- On-board interfaces include 20MA TTY, dual audio cassette and 1 user I/O R6522 device
- Firmware includes interactive monitor and text editor w/options of Assembler, BASIC, FORTH, PASCAL, & PL/65

And if the above isn't enough,
Try the RM65 — a product line filled with embellishments including:

32K DRAM Board	ACIA Board	Prototype cards
CRT Controller	IEEE-488Board	Adaptor Buffer Modules
Floppy Disk Controller	CPU/SBC Board	General Purpose I/O Board
PROM Programmer	4-16 Slot Card Cages	PROM/ROM Board

NEW LOWER PRICES AND A CASH DISCOUNT* TO BOOT!

A65/40-16 (16K RAM)	$1225
A65/40-32 (32K RAM)	$1295
A65/40-A (Assembler)	$ 85
A65/40-B (BASIC)	$ 65

A65-1 (1K RAM)	$420
A65-4 (4K RAM)	$445
A65-4B (4K RAM w/BASIC)	$495
A65-PS (PASCAL)	$100
A65-F (FORTH)	$ 65
A65-A (Assembler)	$ 35

Mail Order to:

Educational Computer Division
EXCERT INCORPORATED

- SALES
- SERVICE
- INSTALLATION
- CONSULTING

P.O Box 8600
White Bear Lake, MN 55110
(612) 426-4114

Higher quantities quoted upon request, COD's accepted, shipping will be added. *Deduct 5% cash discount on prepaid orders. Minnesota residents add 5% sales tax. Prices subject to change without notice.

MICRO Short Subjects

Hybrid Program Storage — *by Chris Williams*

A Bug in Apple's RENUMBER — *by Robert C. Leedom*

Hybrid Program Storage

Chris Williams, 5676 S. Meadow La., #101, Ogden, Utah 84403

Usually for reasons involving speed, many Applesoft programs contain assembly language subroutines as an integral part. This "hybrid" form conventionally requires two SAVEs (one for the assembly and one for the Applesoft) and, of course, two LOADs. An excellent example of this is Richard Suitor's hybrid *LIFE* program included in *MICRO on the Apple*, Volume 1, page 168. The two LOADs can be irritating after using the program several times.

You can reduce this procedure to a single SAVE and LOAD through judicious use of the start- and end-of-program pointers located at $67-$68 and $AF-$B0 respectively. If your assembly language subroutines are located above the Applesoft program (i.e., higher memory) then all you need to do is put the final location of the assembly language subroutines into the end-of-program pointer. A single BASIC SAVE will now save everything, and a single BASIC LOAD will bring it all back. This is particularly convenient if you happen to be working with tape.

If your assembly language subroutines are located below the Applesoft program you must be a bit trickier. Set the start-of-program pointer to two locations prior to the start of the assembly language. Next, do your BASIC save, and then whenever you wish to load it be sure to set the start-of-program pointer to this same value. And before running, in this instance, the start-of-program pointer must be reset to the start of the Applesoft. Additionally, the byte immediately prior to the Applesoft must be set to 0 to avoid a SYNTAX ERROR message.

These same methods can be used to enable an Applesoft machine (Apple II Plus) to run Integer BASIC hybrid programs. For this application, in addition to the measures described above, you must take care of another potential problem. Again, I direct you to Richard Suitor's *LIFE* as an illustration.

Mr. Suitor placed his assembly language routines at $800 and used Integer BASIC. Since, as it turns out, all of his Integer BASIC commands are executable in Applesoft, it *should* run. However, Applesoft defaults to $800 for the start of program storage.

To overcome this problem, simply set the start-of-program pointer to a point similar to $1000 before you type in the BASIC. When you're finished, reset the pointer to $7FE (i.e., two locations prior to the start of the assembly language). Set $FFF to 0 — the byte just before the Applesoft — and execute a BASIC SAVE. Now whenever you want to LOAD, set the start-of-program pointer to $7FE and perform a standard BASIC LOAD.

A Bug in Apple's RENUMBER

Robert C. Leedom, 14069 Stevens Valley Ct., Glenwood, Maryland 21738

The RENUMBER utility program supplied with Apple DOS 3.2 and 3.3 has an insidious bug. However, after you use RENUMBER, your program may appear to run perfectly, so you may not even notice that your program's operations have been altered! RENUMBER will correctly change all line number references to agree with the new line numbers. Unfortunately, it also may alter any number in an arithmetic expression which follows an asterisk (the multiply operator), and has the same value as a pre-RENUMBER line number.

I obtained the corrections for the DOS 3.2 version from the Apple Hotline in May of 1980, but I recently discovered that the problem still exists in the DOS 3.3 version.

The fixes for the DOS 3.2 and DOS 3.3 versions of the program are similar: they involve swapping two data values in the program, as shown in table 1.

To permanently correct the RENUMBER program you must

1. LOAD RENUMBER
2. EXECUTE the two POKEs for your version of DOS
3. SAVE RENUMBER

For your future reference, Apple dealers have a loose-leaf notebook which answers commonly-asked questions including "What's wrong with RENUMBER?" Also, *The Apple Orchard* indicates that the two locations to be POKEd for RAM Applesoft RENUMBER are 14342 and 14343.

Table 1

	DOS 3.2	DOS 3.3
From BASIC	POKE 4815,172 : POKE 4816, 171	POKE 4789,172 : POKE 4790,171
or From monitor	* 12CF: AC AB	* 12B5: AC AB

MICRO

OSI COMPATIBLE PRODUCTS

56K 2-MHz Ultra Low Power CMOS Static Memory Board MEM-56K $850

Partially Populated Boards (Specify address locations required) **MEM-48K $750**
MEM Board uses the new 2K-Byte Wide Static RAM chips which are **MEM-32K $550**
2716 EPROM compatible. Any 2K byte memory segment can be **MEM-24K $450**
populated with RAM or EPROM (or left empty for use of Address Space **MEM-16K $350**
by another board). Fully expandable to any memory size you will ever **MEM- 8K $250**
need. No special addressing requirements, just solder in extra sockets **MEM- 4K $200**
and add memory. Also has space for a 1.75K Monitor ROM at $F800 (FC).

Extra 2K RAM Memory Chip	**$24**
Optional Parallel Printer Port	**-P $120**
Optional Calendar/Clock (Software available in EPROM)	**-T $ 25**
Both options (Disk software mods provided for use of 6522 VIA on printer).	**-PT $125**

EXAMPLE USES:

C4P & C8P: Expansion to **40K RAM** of Basic workspace.
Parallel Printer Port — Reserve Serial Port for MODEM
Calendar/Clock Displaying on unused portion of screen.
Space for 5.75K of **Enhanced System Monitor EPROMS.**

All of this on 1 Board, using only one of your precious slots. Software for Enhanced System Monitor capabilities is continuously being developed and improved. As new EPROM Monitors are available, you may upgrade to them for any price differential plus a nominal $10 exchange fee. Another possibility is to fill any portion of the memory with Basic Programs in EPROM for **Power-on Instant Action.** This custom EPROM programming service is available at $25 per 2716 (includes EPROM). Extra copies at $15 for each EPROM.

C4P-MF & C8P-DF: Memory expansion to **48K.**
Add 4K**Memory** at $E000 for special software requirements.
Parallel Printer Interface and/or **Displaying Calendar/Clock.**
Add 1.75K **Enhanced System Monitor ROM.**

C3: Up to **56K of Memory** Expansion — can be addressed for **Multiuser.** (Optionally, each user can have his own **Dedicated Printer Port**). Add **Enhanced Monitor ROM** with Calendar/Clock software, warm start and Hard Disk Boot.

IEEE-488 INTERFACES AND SOFTWARE:

The General Purpose Instrumentation Bus (GPIB) Controller interface is available for all OSI Computers. Machine code GPIB Drivers are linked to Basic to provide easy control of IEEE-488 instruments which is equal to the best of Hewlett-Packard Controllers and far superior to most others. Basic Commands for Serial Poll, Parallel Poll, IFC Clear, full Local/Remote Control, Respond to SRQ Interrupts, Send Trigger, do Formatted Input/Output, Direct Memory Input/Output and MORE. Interface includes IEEE-488 Ribbon Cable/Connector.

GPIB Controller Interface for C2, C3, C4 and C8 Systems .... **GPIB 4-488 $395**
GPIB Software for **OS-65D** (Add -8 for 8" or -5 for 5") .... **GPIB 488-D $ 70**
GPIB Software for **OS-65U** .... **GPIB 488-U $100**
GPIB Software on two 2716 EPROMS for **ROM** based systems ..... **GPIB 488-R $100**
Add Optional **Parallel Printer Interface** to GPIB 4-488 .... **-P $120**
Add Optional **Calendar/Clock** to GPIB 4-488 .... **-T $ 25**
Add **2K RAM** to GPIB 4-488 (Specify location, $4000-$BFFF & $D000-$EFFF available) **-M $ 25**
Software for **EPROM Programming, Reading, Verifying,** and **Erased Check;** fully integrated with **Assembler, Editor** and **Extended Monitor.** Can be used with many types of EPROM up to 8K. Requires Optimal Technology Model Ep-2A-79 EPROM Programmer and the GPIB 4-488 Board. Specify 8" or 5" Disk .... **EPROM MI-EP $180**
GPIB Controller for C1P, Includes Software, Clock & space for 6K EPROM .... **GPIB 6-488R $395**
Add Optional **Parallel Printer Interface** to GPIB 6-488R........................ **-P $120**

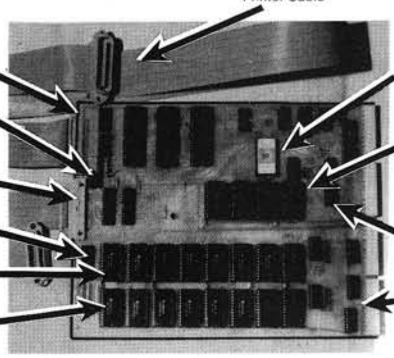
EPROMS: (Check with your Dealer for newest EPROM Products).

C1P ROM with 48 Col Display, Smart Terminal, Edit & More for Series II .... **ROM-TERM II $59.95**
C1P ROM with 24 Col Display, Other ROM-TERM II Features & Disk Boot .... **ROM-TERM $59.95**
Just flip switch for Serial or Video System with Corrected Keyboard .... **SYNKEY $39.95**

ENHANCED MONITOR ROMS FOR USE ON GPIB 4-488 & MEM BOARDS:

Expanded Support for **C4P & C8P** Featuring Calendar/Clock, Line Edit, Smart Terminal, Memory Files, Parallel Printer Control & More **MI48P1 $59.95**
Disk Support with Calendar/Clock, Warm Start and Corrected Keyboard **MI48D1 $59.95**
Expanded C3 Monitor with Calendar/Clock Software, Hard Disk Boot, and Warm Start **MIC3-1 $59.95**
C1-P Series II Computer with ROM-TERM II Smart Terminal Monitor (Order Direct) ... **$549.00**

IEEE-488 CONTROLLER INTERFACE



THE GPIB 4-448 INTERFACE BOARD CONVERTS ANY OSI COMPUTER INTO AN IEEE-488 INSTRUMENT BUS CONTROLLER!

BENEFITS-- Provides a Sophisticated Instrumentation Controller at very low cost (often saving thousands of Dollars). The combination of IEEE-488 Instrumentation Controller and High Capacity Hard Disk file storage available on OSI Computer systems is available at a fraction of the cost required by the nearest competitor. The IEEE-488 Bus, also known as the GPIB, HP-IB or IEC-625 is the most popular International Standard for connecting instrumentation systems. This 16-line bus is designed to interconnect and control up to 15 instruments at a time. Currently, over 1400 different instruments are available to work on this bus. They include: Plotters, Digitizers, Printers, Graphic Displays, Recorders and a multitude of specialized Test/Measurement/Control Equipment.

COMPATABILITY-- The fast machine code GPIB drivers are linked to OSI BASIC to provide easy programmable control of standard instrumentation which is equal to the best HP-IB Controllers and far superior to all others. These GPIB Drivers are accessed from BASIC with a new command "BUS" followed by simple control letters to do total command of IEEE-488 Instrumentation. You can do direct Input/Output of formatted String and Numerical variables as simply as with the "INPUT" and "PRINT" statements. Utilizes the MC68488 IC designed by Motorola to provide automatic Handshaking and Addressing Protocol as required by the IEEE-488/78 Standard. This combination of Hardware/Software makes instrumentation control programs extremely easy to write and modify.

EPROM-ABLE -- Can be used with a C4-P to create a dedicated IEEE-488 controller which has the capability of running your unique program at Power-on (No loading of program tapes required). You simply develop your program for your special control function and send a copy of tape to us. We will then transfer your program to EPROM and you are on line with a customized IEEE-488 Instrumentation System.

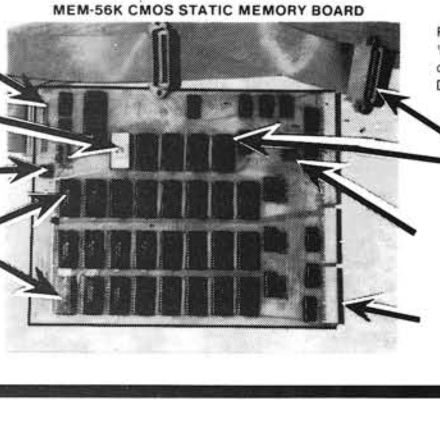
C2-D MULTIPLE USER SYSTEMS

SAVE -- 2 and 3 user Time Sharing Systems are available on the C2-D Winchester Disk Computer at a considerable cost savings from C3 Multiple User Systems. The 3 user C2-D System can be expanded to include a word processing printer, 4 other parallel printers and 3 serial printer interfaces.

COMPATABLE -- The special C2-D Multi-User Executive Program is 100% compatable with OS-65U V1.2. The Multi-User Real Time Clock, Memory Partition Control and IRQ Interrupt Management are done on the Micro Interface Memory Board. Thus, the CPU board is not modified and remains in factory condition.

CONVERSIONS - The Up-Grade of your existing C2-D Computer to Multiple User Configuration is also available. Call for details.

MEM-56K CMOS STATIC MEMORY BOARD



ULTRA-LOW POWER — By using CMOS Static RAM Memory, the total power consumption is about ½ Amp at 5 Volts when populated for 48K. In fact, most of power is used by the Address Line Buffers and the Data Transceivers.

MULTI-USER — Can be addressed for any of the 16 multi-user memory partitions. The low power and single memory board/partition simplify installation and provide a typical $1400 saving for a 3-user system.

MICRO-INTERFACE
3111 SO. VALLEY VIEW BLVD., SUITE I-101
LAS VEGAS, NEVADA 89102
Telephone: (702) 871-3263

Check with your local Dealer or Order Direct.
Phone orders accepted.
TERMS: Check/Money Order/Master Charge/VISA
Sent POSTPAID ON PREPAID ORDERS.
Foreign Orders: Prepaid only.
Add 5% for handling/shipping.

Cross Reference Generator for OSI BASIC-in-ROM

Program development and debugging often depend on keeping track of references and variables. The following article describes a cross reference generator for OSI ROM BASIC which will help you find any variable or line number within a BASIC program.

John Krout
5108 N. 23rd Road
Arlington, Virginia 22207

It is almost inevitable that when you develop a large program in BASIC, you'll need to find all the references to some aspect of the program. If you decide to delete a particular line, it is important to locate all the GOTOs, THENs, and GOSUBs mentioning that line. If you want to conserve memory by merging two string variables into one, you must find all the appearances of the string variable names. A cross-reference generator program is extremely useful at times like these, for it can find references within your program much faster and more accurately than the traditional visual search.

A cross-reference generator is most often needed, however, when free memory is a scarce commodity. In this article we'll develop a cross-reference generator which requires less than 1K of RAM and will find references to variable names, constants, literals, line numbers, and any word in the vocabulary of BASIC.

When you type a line of BASIC program text, OSI BASIC-in-ROM stores that text in a condensed or "tokenized" format in RAM. Listing 1 is a program which takes a look at itself in RAM, and table 1 shows that program's output.

Table 1

```
768   0          787 ( 40        806 1 49        825 ( 40
769   25         788 1 49        807   3         826 X 88
770   3          789 2 50        808 ■ 206       827 ) 41
771 j 106        790 4 52        809   234       828 ; 59
772   234        791 ) 41        810 X 88        829 X 88
773 T 84         792   0         811 ■ 171       830   0
774 ■ 171        793 & 38        812 ■ 187       831 H 72
775 ■ 187        794   3         813 ( 40        832   3
776 ( 40         795 ■ 196       814 I 73        833
777 1 49         796   234       815 ) 41        10
778 2 50         797   129       816   0         834   235
779 3 51         798 I 73        817 ? 63        835   130
780 ) 41         799 ■ 171       818   3         836 I 73
781 ■ 163        800 7 55        819   0         837 : 58
782 2 50         801 6 54        820   235       838   128
783 5 53         802 8 56        821   151       839   0
784 6 54         803   157       822 I 73        840   0
785 ■ 165        804 T 84        823 ; 59        841   0
786 ■ 187        805   0         824   192       842   0
                                                 843 T 84
```

In listing 1, variable T points to the beginning address of numeric variable storage in RAM, which is also the end of your BASIC program text. The beginning of BASIC text is address 768. (See MICRO 31:61 for more information on text and variable storage area pointers.) To look at the RAM storing BASIC text, the FOR-NEXT loop examines all addresses from 768 to T. Line 60160 prints the address, the graphic corresponding to the data at the address, and the data at the address in decimal.

Although the printer used to create table 1 does not use OSI's entire graphics code, a comparison of listing 1 to its tokenized version in table 1 is very informative. First of all, we can see that the variable names, constants, and some BASIC symbols are stored in their ASCII code form, just as if they were strings of characters. Most BASIC keywords and symbols, however, are stored as single characters called "tokens," and all of the tokens have values greater than 127.

The line number of each line is also stored. While each reference to a line number (GOTOs, GOSUBs, THENs) is stored as a string following the appropriate token, the line number of each tokenized line is stored at the beginning of the line in low-high format. For instance, line number 60010 begins at address 771:

$$PEEK(771) + 256*PEEK(772) = 60010$$

Moreover, each line of tokenized text is terminated with a zero.

Listing 1

```
60010 T=PEEK(123)+256*PEEK
      (124)
60100 FORI=768TOT
60110 X=PEEK(I)
60160 PRINTI;CHR$(X);X
60170 NEXTI:END
```

Listing 2

```
60120 IFX=0GOTO60500
60500 REM NEW LINE
60510 LINE=PEEK(I+3)+256*PEEK
      (I+4)
60520 PRINTLINE
60530 I=I+5
60540 GOTO60110
```

There are two other bytes of data between each terminating zero and the bytes representing the number of the following line. These are a pointer, also in low-high format, to the next line. For instance, before the beginning of line 60010 in RAM:

PEEK(769) + 256*PEEK(770) = 793

At address 792 we see the zero terminating line 60010, and at address 795 and 796 the number of the second program line is stored. Therefore, the next-line pointer for each line points to the next-line pointer for the following line.

Listing 2 is a modification to be added to listing 1 which decodes and prints the number of each tokenized line. The program spots each terminating zero in line 60120 and branches to the line decoder. An interesting point about FOR-NEXT loops is utilized in line 60530: you can change the value of the loop variable while the loop is running. This enhances execution speed slightly by skipping the next-line pointers.

It stands to reason that, if BASIC can translate new text lines to tokens and, during a LIST, *vice versa*, then there should be a dictionary of BASIC vocabulary and corresponding tokens somewhere in ROM. In fact, the dictionary resides in addresses 41092 through 41314 (see MICRO 24:25, 23:65). Listing 3 takes a look at the dictionary, and the results of listing 3 appear in table 2.

The items are placed in the dictionary in the numerical order of their corresponding tokens. The last character of each item has its most significant digit set to 1, to tell BASIC that the end of the item has been reached. In listing 3, X represents a byte of data in the dictionary, and is used in line 61040 to build a string, B$, of consecutive bytes. Line 61050 branches to avoid incrementing the token number, variable TK, and printing and clearing B$, if the item is not yet complete; i.e., if the most significant bit of X is cleared. While assembling B$, we use Boolean logic in line 61040 to clear the most significant bit of every character, not just the last one. This may be overkill, but it is also compact code and serves our need to conserve RAM.

We can now knit together listings 1 through 3. This will enable us to search for any string, or token corresponding to a dictionary item, that we need to find.

Listing 3

```
61000 REM LOOKUP TOKEN
61010 TK=127:B$=""
61020 FORI=41092TO41314
61030 X=PEEK(I)
61040 B$=B$+CHR$(XAND127)
61050 IFX<128GOTO61100
61060 TK=TK+1
61070 PRINTTK;B$
61080 B$=""
61100 NEXT
```

Listing 4

```
60050 INPUT"WHICH STRING";A$:PRINT
60070 L=LEN(A$):B$=""
60130 B$=B$+CHR$(X)
60160 IFA$=RIGHT$(B$,L)THENPRINTLINE;
60170 NEXTI:PRINT:GOTO60050
60520 B$=""
```

Listing 5

```
60030 INPUT"KEYWORD OR STRING";A$:PRINT
60040 IFASC(A$)=75GOTO61000
60170 NEXTI:PRINT:GOTO60030
61005 INPUT"WHICH KEYWORD";A$:PRINT
61015 L=LEN(A$)
61070 IFA$=LEFT$(B$,L)THENA$=CHR$(TK):GOTO60070
61200 PRINTA$;" NOT FOUND":PRINT:GOTO60030
```

Table 2

128	END	145	NULL	162	STEP	179	SQR
129	FOR	146	WAIT	163	+	180	RND
130	NEXT	147	LOAD	164	-	181	LOG
131	DATA	148	SAVE	165	*	182	EXP
132	INPUT	149	DEF	166	/	183	COS
133	DIM	150	POKE	167	^	184	SIN
134	READ	151	PRINT	168	AND	185	TAN
135	LET	152	CONT	169	OR	186	ATN
136	GOTO	153	LIST	170	>	187	PEEK
137	RUN	154	CLEAR	171	=	188	LEN
138	IF	155	NEW	172	<	189	STR$
139	RESTORE	156	TAB(	173	SGN	190	VAL
140	GOSUB	157	TO	174	INT	191	ASC
141	RETURN	158	FN	175	ABS	192	CHR$
142	REM	159	SPC(	176	USR	193	LEFT$
143	STOP	160	THEN	177	FRE	194	RIGHT$
144	ON	161	NOT	178	POS	195	MID$

Listing 4 modifies listings 1 and 2 to find a string, represented by the variable A$, in any tokenized text line. A$ can therefore be a variable name, constant, line reference, or literal in a print statement, data statement, string computation or remark. The variable B$ here represents the tokenized text, and is built byte by byte in line 60130. If the contents of A$ resides anywhere within B$, then sooner or later A$ will equal the rightmost L characters of B$, where L represents the length of A$. When this match occurs, line 60160 prints the line number of the current line represented by B$. The previous unconditional print of each byte and line number has been replaced, and B$ is cleared in line 60520 whenever a new line number is decoded.

If you have entered listings 1 through 4 in sequence, then listing 5 adds the capability of converting a keyword to its token by searching the dictionary, and finding all references to the token. Line 61070 converts the numeric token TK to a 1-byte string A$, and then uses the string search routine of listing 4 to locate matches for A$.

As is, the cross-reference generator will now find all that you seek, but it finds a few extra items as well. As an example, direct the program to examine its own text for references to the numeral 7. It prints the line numbers in which the constants 75, 768, and 127 as well as line reference 60070 appear. Ask it to find references to the numeric variable A (there are none), and it prints references to A$. If references to T are sought, two of the input prompts and one of the remark literals are found, as well as all references to T and TK. Some fine tuning is definitely in order to eliminate, or at least cut down on, the unwanted reference reports.

The problem of distinguishing a constant from a line reference is very complex, partly because line references can be surrounded by commas in an ON/GOTO or ON/GOSUB context, while constants can also be surrounded by commas in a multiple-argument function or command. In my programs, I've found line references to be far more common than constants, and far more likely to end with the numeral 0. I have seen other cross-reference generators which can do the job, but they are larger than this one and not as versatile. Since our purpose is compactness, versatility is useful, and since the chances of confusion appear to be minimal, I can live with the constant/line reference problem.

The problem of distinguishing subscripted, string and numeric variables is much easier to solve. If references to a numeric variable are sought, the program should reject any it finds which are followed by either a (or a $. If references to a string variable are sought, the program should ignore any followed by a (character. These suffix rejection rules for numeric and string variables suggest that we can eliminate erroneous references embedded in larger strings (illustrated above by the searches for 7 and T) by implementing a set of suffix and prefix rejection rules.

The prefix rule for all strings is rejection of references preceded by a numeric or upper-case alphabetic character. The suffix rule for constants, line references and numeric variables is as stated above for numeric variables, with the additional rejection of numeric and upper-case alphabetic suffixes.

Listing 6 incorporates these rules into the cross-reference generator, utilizing three defined Boolean functions in a single IF/GOTO statement. The functions are defined in lines 60005 through 60007. The argument in

Listing 6

```
60005 DEFFNA(X)=(X>47ANDX<58)OR(X>64ANDX<91)
60006 DEFFNB(X)=X<>36ANDX<>40
60007 DEFFNC(X)=NOTFNB(X)ORFNA(X)
60070 L=LEN(A$):B$="":A=ASC(A$)
60080 IFA>127GOTO60100
60090 B=ASC(RIGHT$(A$,1))
60135 IFA>127GOTO60160
60140 IFA$<>RIGHT$(B$,L)GOTO60170
60145 Y=PEEK(I+1):IFLEN(B$)>LTHENW=ASC(RIGHT$(B$,L+1))
60150 IFFNA(W)OR(B=36ANDY=40)OR(FNB(B)ANDFNC(Y))GOTO60170
60535 W=0
```

each is the ASCII value of a character. FNA returns a true value if the character is numeric or upper-case alpha. FNB returns true if the character is neither (nor $. FNC, utilizing FNA and FNB in its definition, returns true if the character is either numeric, upper-case alpha, (or $.

Line 60070 is modified to set new variable A equal to the ASCII value of the first byte of A$. Lines 60080 and 60135 skip over the rules implementation if A indicates that A$ represents a token. Line 60090 sets new variable B equal to the ASCII value of the last byte of B$, to decide later if the string to be found is a subscripted or string variable.

Since the program doesn't need the rules unless a potential reference is located, line 60140 jumps past the rules until that condition is met. In line 60145, Y is the ASCII value for the reference suffix and, if the reference is not the first item in the text line, then W is the ASCII value of the reference prefix. Line 60535 sets W to zero whenever a new line number is decoded.

Line 60150 skips the line number printing statement if any of the prefix or suffix rejection rules are met when a potential reference is found. This is one easy way to read the line:

> IF the prefix W in the text is numeric or upper-case alpha,
>
> OR the item sought ends with a $ and the text suffix is a (,
>
> OR the item ends with neither (nor $ and the text suffix is either numeric, upper-case alpha, $ or (,
>
> GOTO 60170.

The first clause implements the prefix rule, the second the string variable suffix rule, and the third the suffix rule for numeric variables, constants, and line references.

Listing 7 is the result of all these developments. It does indeed run in less than 1K of RAM, with about 200 bytes to spare for a few instructions inserted between lines 60010 and 60030, if desired. That might be a good place to remind yourself that the symbols +, −, *, /, ∧ , > , =, and < are treated as keywords, not strings. (See table 2.)

Listing 7

```
0 GOTO60000
60000 REM XREFGEN
60002 CLEAR
60005 DEFFNA(X)=(X>47ANDX<58)OR(X>64ANDX<91)
60006 DEFFNB(X)=X<>36ANDX<>40
60007 DEFFNC(X)=NOTFNB(X)ORFNA(X)
60010 T=PEEK(123)+256*PEEK(124)
60030 INPUT"KEYWORD OR STRING";A$:PRINT
60035 Y=FRE(1)
60040 IFASC(A$)=75GOTO61000
60050 INPUT"WHICH STRING";A$:PRINT
60070 L=LEN(A$):B$="":A=ASC(A$)
60080 IFA>127GOTO60100
60090 B=ASC(RIGHT$(A$,1))
60100 FORI=768TOT
60110 X=PEEK(I)
60120 IFX=0GOTO60500
60130 B$=B$+CHR$(X)
60135 IFA>127GOTO60160
60140 IFA$<>RIGHT$(B$,L)GOTO60170
60145 Y=PEEK(I+1):IFLEN(B$)>LTHENW=ASC(RIGHT$(B$,L+1))
60150 IFFNA(W)OR(B=36ANDY=40)OR(FNB(B)ANDFNC(Y))GOTO60170
60160 IFA$=RIGHT$(B$,L)THENPRINTLINE;
60170 NEXTI:PRINT:GOTO60030
60500 REM NEW LINE
60510 LINE=PEEK(I+3)+256*PEEK(I+4)
60515 IFLINE>59999THENPRINT:GOTO60030
60520 B$=""
60530 I=I+5
60535.W=0
60540 GOTO60110
61000 REM LOOKUP TOKEN
61005 INPUT"WHICH KEYWORD";A$:PRINT
61010 TK=127:B$=""
61015 L=LEN(A$)
61020 FORI=41092TO41314
61030 X=PEEK(I)
61040 B$=B$+CHR$(XAND127)
61050 IFX<128GOTO61100
61060 TK=TK+1
61070 IFA$=LEFT$(B$,L)THENA$=CHR$(TK):GOTO60070
61080 B$=""
61100 NEXT
61200 PRINTA$;" NOT FOUND":PRINT:GOTO60030
```

A few extra lines in listing 7 are useful options. Line 0 is simply a jump to the start of the program, so you can load it from tape on top of your main program already in RAM, and simply type RUN to begin cross referencing. Since modification of a program erases the tables of variables in upper RAM, you'll need the CLEAR statement in line 60002 only if you test your own program and then enter the cross-reference generator by typing GOTO 60000. The FRE function in line 60035 allows the garbage collection routine to conserve memory in the string storage space whenever a new A$ is input in line 60030. Rest assured that garbage collect will not crash the system (MICRO 35:43) unless your own program uses subscripted string variables and their values are preserved by avoiding both program modification and the CLEAR statement. Line 60515 ends the search when the program's own line numbers are reached.

You can conserve even more memory by deleting the remark statements and altering the references to those lines accordingly, as well as by combining unreferenced lines into multiple statements. Obviously, this latter step saves the four-byte header for each of the lines eliminated, and can add up to a critical saving.

Have you been wondering about the need for the next-line pointers? They are essential to BASIC's execution of branching statements. An understanding of this process will help you improve execution speed of your own programs as well as the cross-reference generator.

When a branch token such as a GOTO is executed, BASIC first translates the string of digits following the token into the low-high line number format. The speed of this operation clearly depends on the length of the string, so it always helps to utilize small line numbers, even though this may be impractical in large programs. If line references were stored in low-high format when tokenized, it would save memory and speed things up. I suspect Microsoft shares my conclusion that it is difficult to distinguish constants and line references.

Once the line number is ready, BASIC looks at each tokenized line header in turn, starting with the first program line in RAM, until a line number match is found. If the current header doesn't match, BASIC uses the next-line pointer to skip to the next header.

You can maximize the speed of this skip-compare process by minimizing the number of lines and lengthening each line with multiple statements. You should also put your most frequently-called routines in the lowest line numbers, where BASIC will find them first, and put the initialization code in the highest line numbers, so BASIC won't have to skip through it on the way to the more important material. The cross-reference generator has a very significant execution speed problem in this regard, because not only its own initialization in lines 60000-60090, but also the entire tokenized text data base, sits below the main processing loop routine in RAM!

However, you can modify the cross-reference generator to use next-line pointers in two ways to improve execution speed. Once a reference is found in a line, there is no need to search the remaining portion of the line, so use the pointer to increment the loop variable I to the beginning of the next line. More helpful is an input specifying the range of line numbers in your program through which the cross-reference generator should search. It can use the next-line pointers to skip to the first line number you specify, and then quit when it finds the last line number you specify. If you're looking for references to a block of code in your own program about to be moved or eliminated, you can reduce the number of searches required by adding a search for references to a specified range of line numbers. I suggest that you create a defined Boolean function of your own to help implement the rules for these extra features.

John Krout is a patent attorney with the firm of Gipple & Hale in Arlington, Virginia. He teaches an adult education course entitled Introduction to Computer Programming, at Open University in Washington, DC, He has computerized Open University's walk-in registration process on a Challenger 1P, and has performed trendspotting election analysis on Election Night 1981 for the Virginia Radio Network using a Challenger 8P DF.

MICRO

More Hooks into OSI BASIC

This article shows you how to add your own keywords to BASIC under OS-65D V3.2.

Richard L. Trethewey
5405 Cumberland Road
Minneapolis, Minnesota 55410

In the September 1980 issue of *The Small Systems Journal* (MICRO 28:42), Ohio Scientific published a method by which the user could add his own new keywords to BASIC. As soon as I saw it, I knew I wanted to try it, and I did. It worked, but I didn't want to stop at the two keywords that OSI had provided. I wanted much more.

Unfortunately, the way OSI wrote their code involved the use of look-up tables which made adding new keywords a little cumbersome. Having the look-up tables imbedded in the code also meant a small memory overhead. I rewrote the code to use simple direct comparisons instead of comparisons and subsequent jumps from tables. This makes the code shorter, a little faster, and easier to follow.

Even though my program takes up 1K of memory, in just 2½ pages of RAM I have implemented commands for the following: screen clear, color background select, screen state select, BASIC reset, disk directory printer, hex-dec and dec-hex converter, and a screen-to-printer dump. It's well worth the overhead, even on 24K systems.

To implement new keywords, OSI replaced the regular instructions in BASIC that interpreted the keyword "LET" with a JSR to the code that interpreted the new keywords. OSI used an asterisk as a character in both of their keywords. This is because when BASIC saves text, it tokenizes certain

Trethewey Listing

```
BC00                    *=$BC00
00A5=                   TOKEN=$A5                    ; TOKEN FOR ASTERISK
00C7=                   TXTPTR=$C7                   ; POINTER TO TEXT
BC00 A001               LDY #$01
BC02 B1C7               LDA (TXTPTR),Y               ; FETCH 2ND CHARACTER
BC04 C9A5               CMP #TOKEN                   ; IS IT AN ASTERISK?
BC06 D023               BNE BACK                     ; NO! GO BACK TO BASIC
BC08 88                 DEY                          ; YES! NOW GET 1ST CHAR.
BC09 B1C7               LDA (TXTPTR),Y
                        ;
                        ; NOW GO TO APPROPRIATE CODE
                        ;
BC0B C943               CMP #'C
BC0D F038               BEQ CCODE
BC0F C942               CMP #'B
BC11 F04E               BEQ BCODE
BC13 C953               CMP #'S
BC15 F058               BEQ SCODE
BC17 C951               CMP #'Q
BC19 F05F               BEQ QCODE
BC1B C952               CMP #'R
BC1D F06C               BEQ RCODE
BC1F C944               CMP #'D
BC21 F07A               BEQ DCODE
BC23 C948               CMP #'H
BC25 F00C               BEQ HCODE
BC27 C950               CMP #'P
BC29 F00B               BEQ PCODE
                        ;
                        ; RETURN TO BASIC AND CONTINUE
                        ;
BC2B 202E0F   BACK      JSR $0F2E
BC2E 8596               STA $96
BC30 8497               STY $97
BC32 60                 RTS
                        ;
                        ; JUMPS TO CODE BEYOND PAGE BOUNDARIES
                        ;
BC33 4C5EBD   HCODE     JMP H0
BC36 4C59BE   PCODE     JMP P1
                        ;
                        ; RESETS POINTERS AND RETURNS TO BASIC
                        ;
BC39 A5C7     UPDATE    LDA TXTPTR
BC3B 18                 CLC
BC3C 6902               ADC #$02
BC3E 85C7               STA TXTPTR
BC40 9002               BCC UP1
BC42 E6C8               INC TXTPTR+1
BC44 68       UP1       PLA
BC45 68                 PLA
BC46 60                 RTS
                        ;
                        ; ROUTINE TO CLEAR SCREEN
                        ;
BC47 A920     CCODE     LDA #'                       ; LOAD ASCII BLANK
BC49 A208               LDX #$08                     ; LOAD PAGE COUNTER
BC4B A000               LDY #$00                     ; INIZ POINTER
BC4D 9900D0   C1        STA $D000,Y                  ; SAVE TO SCREEN
BC50 C8                 INY                          ; BUMP POINTER
BC51 D0FA               BNE C1                       ; LOOP FOR PAGE
BC53 EE4FBC             INC C1+2                     ; BUMP PAGE ADDRESS
BC56 CA                 DEX                          ; DECREMENT PAGE COUNT
BC57 D0F4               BNE C1                       ; LOOP 'TIL DONE
BC59 A9D0               LDA #$D0                     ; RESET PAGE ADDRESS
```

(Continued on next page)

words and characters into one byte. The asterisk, which is the operator symbol for multiplication, is one of those characters. This way, the BASIC interpreter can't confuse your new keyword with a variable name. This may be overly cautious since we hook into BASIC at "LET" — one of the last things to be interpreted. If you use reasonable caution, you should be able to choose your own keywords freely. (Note that "LET" is still valid.)

Before we are able to use the new keywords, we must do a little housekeeping in the interpreter. The following is what you will need to incorporate into your BEXEC* in order to use the hooks (I have used OSI's code):

```
10 POKE 133, ADRH-1
20 FOR N=2470TO2476: READ
   A : POKE N,A : NEXT
30 DATA  32,ADRL,ADRH,234,
   234,234,234
40 DISK! "CA ADDR=TT,S"
```

ADRH and ADRL are the high and low bytes of the address where your hooks reside in memory. ADDR is the hex address of the same. TT is the track number and S is the sector where you have saved the hooks' code on disk. The POKE to 133 protects your hooks from getting overwritten by BASIC strings.

I have a 48K system, so ADRH = 188 and my code resides at $BC00. 24K systems may want to use ADRH = 92 and $5C00. My system has 8-inch disks so I put my hooks on track 8, sector 5, which is unused. (Note: OS-65D V3.3 users will have to put their codes elsewhere.)

Another option is to alter BASIC permanently on disk to eliminate lines 20 and 30. After you have done the POKEs from these lines, just enter the following command: DISK! "SA 02,1 = 0200/B" for 8-inch systems. On minis, just change the last "B" to an "8". This will save the adjustments to BASIC and you can eliminate the code in your BEXEC*. Also, if you have enabled LIST, NEW, and <CTRL> 'C', these commands will be enabled whenever you invoke BASIC. While writing this code, I was constantly going from the assembler to BASIC. Not having to do those POKEs was very convenient.

The commands I have added are very simple to use: "C*" clears the screen, "B*x" changes the background

Trethewey Listing *(Continued)*

```
BC5B 8D4FBC          STA C1+2
BC5E 4C39BC          JMP UPDATE                 ; RETURN TO BASIC
                     ;
                     ; ROUTINE TO SET BACKROUND COLOR
                     ;
BC61 A9E0      BCODE LDA #$E0                   ; LOAD CLR PAGE #
BC63 8D4FBC          STA C1+2                   ; SAVE IN CCODE
BC66 A002            LDY #$02
BC68 B1C7            LDA (TXTPTR),Y             ; FETCH COLOR #
BC6A E6C7            INC TXTPTR                 ; ADJUST BASIC
BC6C 4C49BC          JMP CCODE+2                ; AND ENTER CCODE
                     ;
                     ; ROUTINE TO TOGGLE SCREEN SCROLL
                     ;
BC6F AD2A26    SCODE LDA $262A                  ; TOGGLES SCROLL
BC72 4940            EOR #$40                   ; ON AND OFF
BC74 8D2A26          STA $262A
BC77 4C39BC          JMP UPDATE
                     ;
                     ; ROUTINE TO SET DISPLAY STATE
                     ;
BC7A A002      QCODE LDY #$02
BC7C B1C7            LDA (TXTPTR),Y             ; FETCH NUMBER
BC7E 8D00DE          STA $DE00                  ; SEND IT
BC81 A5C7            LDA TXTPTR                 ; AND ADJUST BASIC
BC83 18              CLC
BC84 6903            ADC #$03
BC86 85C7            STA TXTPTR
BC88 4C44BC          JMP UP1
                     ;
                     ; ROUTINE TO RESET NEW, LIST, AND <CTRL> C
                     ;
BC8B A94C      RCODE LDA #76                    ; THESE ARE THE NUMBER
BC8D 8DE502          STA 741                    ; YOU'RE USED TO
BC90 A94E            LDA #78                    ; SEEING
BC92 8DEE02          STA 750
BC95 A9AD            LDA #173
BC97 8D1908          STA 2073
BC9A 4C39BC          JMP UPDATE                 ; GO BACK TO BASIC
                     ;
                     ; ROUTINE TO PRINT DISK DIRECTORY
                     ;
265E=                SCTN=$265E
265F=                PAGES =$265F
2660=                ADRLX=$2660
2661=                ADRHX=$2661
2662=                TRAKX=$2662
26A6=                SEEKX=$26A6
2754=                LOAD =$2754
2761=                UNLOAD=$2761
295D=                CALLX=$295D
2D6A=                CRLF=$2D6A
2CF7=                SWAP=$2CF7
2D92=                PRBYTE=$2D92
2343=                CHROUT=$2343
2D73=                STROUT=$2D73
2E79=                DIRBUF=$2E79
BC9D 20F72C    DCODE JSR SWAP
BCA0 20A9BC          JSR D
BCA3 20F72C          JSR SWAP
BCA6 4C39BC          JMP UPDATE                 ; RETURN TO BASIC
                     ;
                     ; NOTE: BY CHANGING THE FOLLOWING LOCATIONS
                     ; IN THE OS, THE 'D*' COMMAND WILL ALSO BE
                     ; AVAILABLE FROM THE OS KERNEL.
                     ; CHANGE AS FOLLOWS:
                     ; $2E3D=$2A, $2E3E=$A8, $2E3F=$BC
                     ;
BCA9 20732D    D     JSR STROUT                 ; PRINT MESSAGE
BCAC 2A              .BYTE '* DIRECTORY *',$D,$A,$A,0
BCAD 20
BCAE 44
BCAF 49
BCB0 52
BCB1 45
BCB2 43
BCB3 54
BCB4 4F
BCB5 52
BCB6 59
BCB7 20
BCB8 2A
BCB9 0D
BCBA 0A
BCBB 0A
BCBC 00
```

```
BCBD A901            LDA #$01             ; LOAD SECTOR NUMBER
BCBF 8DB3BE          STA COUNT            ; SAVE IT
BCC2 20D2BC          JSR DIRIN            ; READ IN SECTOR
BCC5 20F4BC          JSR D1               ; PRINT OUT CONTENTS
BCC8 EEB3BE          INC COUNT            ; BUMP SECTOR NUMBER
BCCB 20D2BC          JSR DIRIN            ; REPEAT PROCESS
BCCE 20F4BC          JSR D1               ; FOR 2ND SECTOR
BCD1 60              RTS
                     ;
                     ; ROUTINE TO READ IN A SECTOR OF DIRTK
                     ;
BCD2 A979     DIRIN  LDA #$79             ; LOAD LOW BYTE
BCD4 8D6026          STA ADRLX            ; SAVE IT
BCD7 A92E            LDA #$2E             ; LOAD HIGH BYTE
BCD9 8D6126          STA ADRHX            ; SAVE IT TOO
BCDC ADB3BE          LDA COUNT            ; LOAD SECTOR # TO RE
BCDF 8D5E26          STA SCTN             ; SAVE IT
BCE2 A908            LDA #$08             ; LOAD DIR TK #
BCE4 8D6226          STA TRAKX            ; SAVE IT
BCE7 20A626          JSR SEEKX            ; MOVE HEAD TO TRACK
BCEA 205427          JSR LOAD             ; LOAD HEAD
BCED 205D29          JSR CALLX            ; READ SECTOR
BCF0 206127          JSR UNLOAD           ; UNLOAD HEAD
BCF3 60              RTS                  ; AND GO BACK
                     ;
                     ; ROUTINE TO PRINT CONTENTS OF DIRBUF
                     ;
BCF4 A000     D1     LDY #$00             ; INIZ POINTERS
BCF6 8CB2BE          STY FIFTH
BCF9 A200            LDX #$00
BCFB B9792E   D2     LDA DIRBUF,Y         ; FETCH A BYTE
BCFE E006            CPX #$06             ; IS IT START TK #?
BD00 F01A            BEQ TK1              ; YES! PRINT IT
BD02 E007            CPX #$07             ; END TRACK # ?
BD04 F021            BEQ TK2              ; YES! PRINT IT
BD06 C923            CMP #'#              ; IS ENTRY A NULL?
BD08 F008            BEQ DO               ; YES! SKIP IT
BD0A 204323          JSR CHROUT           ; NO! PRINT IT
BD0D E8       D3     INX                  ; BUMP ENTRY COUNTER
BD0E C8       D4     INY                  ; BUMP BUFFER POINTER
BD0F D0EA            BNE D2               ; LOOP 'TIL DONE
BD11 88              DEY
BD12 98       DO     TYA                  ; X-FER POINTER
BD13 18              CLC                  ; TO ACCUMULATOR
BD14 6908            ADC #$08             ; BUMP TO NEXT ENTRY
BD16 A8              TAY                  ; X-FER IT BACK
BD17 B041            BCS QUIT             ; BRANCH IF DONE
BD19 4CF9BC          JMP D2-2             ; LOOP IF NOT
BD1C A920     TK1    LDA #'               ; LOAD BLANK
BD1E 204323          JSR CHROUT           ; PRINT IT
BD21 2053BD          JSR TKOUT            ; PRINT START TK#
BD24 4C0DBD          JMP D3               ; AND LOOP
BD27 A92D     TK2    LDA #'-              ; LOAD "-"
BD29 204323          JSR CHROUT           ; PRINT IT
BD2C 2053BD          JSR TKOUT            ; PRINT END TK#
BD2F EEB2BE          INC FIFTH
BD32 ADB2BE          LDA FIFTH
BD35 C904            CMP #$04
BD37 D00B            BNE TK3
BD39 206A2D          JSR CRLF
BD3C A200            LDX #$00
BD3E 8EB2BE          STX FIFTH
BD41 4C0EBD          JMP D4
BD44 98       TK3    TYA
BD45 48              PHA
BD46 20732D          JSR STROUT
BD49 20              .BYTE '  ',$00
BD4A 20
BD4B 00
BD4C A200            LDX #$00
BD4E 68              PLA
BD4F A8              TAY
BD50 4C0EBD          JMP D4
BD53 B9792E   TKOUT  LDA DIRBUF,Y         ; LOAD TRACK #
BD56 20922D          JSR PRBYTE           ; PRINT IT
BD59 60              RTS
BD5A 206A2D   QUIT   JSR CRLF
BD5D 60              RTS
                     ;
                     ; ROUTINE TO CONVERT DECIMAL TO HEX
                     ;
BD5E A002     HO     LDY #$02
BD60 B1C7            LDA (TXTPTR),Y
BD62 C924            CMP #'$              ; HEX TO DEC ?
BD64 F068            BEQ CONV             ; YES! DO HEX-DEC
BD66 A200            LDX #$00             ; INIZ X
```

(Continued)

color to the ASCII value of x, "S*" toggles the screen scrolling on and off, "Q*x" is like POKEing 56832 with 'x' for the screen/sound select, "R*" resets NEW, LIST, and < CTRL > 'C', "D*" prints a directory of the disk in the currently selected drive, "H*xxxxx" converts decimal "xxxxx" to hex, "H*$xxxx" converts from hex to decimal, and "P*" dumps text from the screen to the serial printer port. These commands may be used in either the program or the immediate mode, which means they can be entered from the keyboard or embedded in your programs. The disk directory can be handy if you are trying to find where a certain program is on disk; the hex-dec converter can save a lot of headaches when writing USR(X) routines. Note that the "H*" command doesn't expect a specific format.

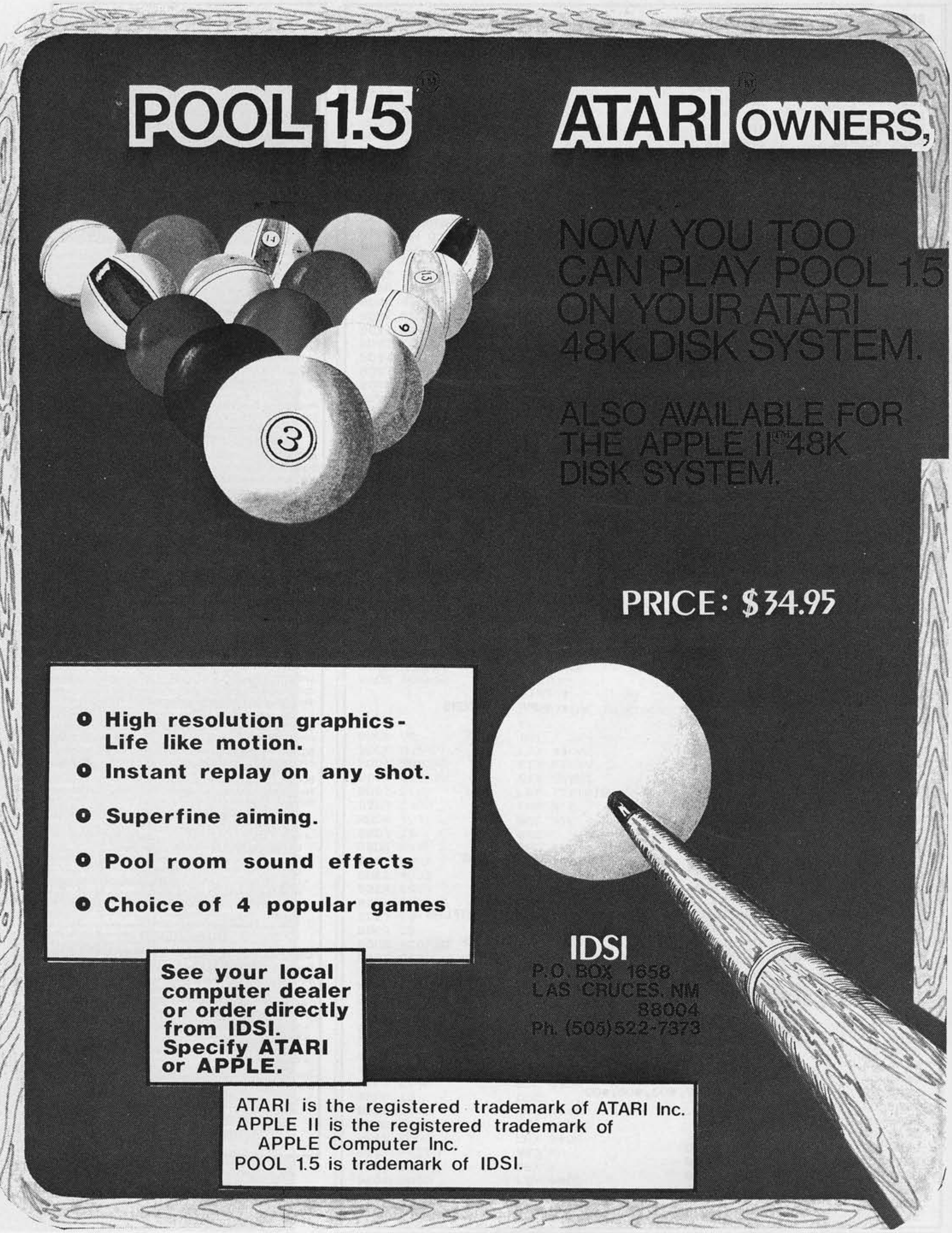
In order to save room I used existing code in the operating system where possible. OS-65D V3.3 owners may want to alter the output routines to format the output of the directory a little.

My thanks to OSI for the original idea. They deserve a lot of credit for making this information available and workable on so many different systems.

Trethewey Listing *(Continued)*

```
BD68 8EB1BE          STX RESHI          ; INIZ RESHI
BD6B B1C7     H1     LDA (TXTPTR),Y     ; GET CHARCTER
BD6D C930            CMP #$30           ; IS IT A # ?
BD6F 900F            BCC H2             ; NO! CALCULATE
BD71 38              SEC
BD72 E930            SBC #$30           ; STRIP OFF ASCII
BD74 9DB5BE          STA INBUF,X        ; SAVE IT
BD77 C8              INY
BD78 E8              INX                ; BUMP POINTERS
BD79 E006            CPX #$06           ; TOO MANY DIGITS?
BD7B F003            BEQ H2             ; CALCULATE ANYWAY
BD7D 4C6BBD          JMP H1             ; NO! ==> LOOP
BD80 98       H2     TYA
BD81 18              CLC
BD82 65C7            ADC TXTPTR
BD84 85C7            STA TXTPTR         ; UPDATE BASIC
BD86 CA              DEX                ; BUMP X BACK 1
BD87 BDB5BE          LDA INBUF,X        ; GET ONES
BD8A 8DB0BE          STA RESLO          ; ALWAYS SAVE IT
BD8D E000            CPX #$00           ; SINGLE DIGIT ?
BD8F F029            BEQ PRINT          ; YES! PRINT!
BD91 A003            LDY #$03           ; SET POINTER TO TBLS
BD93 CA              DEX                ; BUMP X BACK ONE
BD94 8EB3BE   H3     STX COUNT          ; SAVE X
BD97 BDB5BE          LDA INBUF,X        ; GET NUMBER
BD9A F01A            BEQ H5
BD9C AA              TAX                ; X-FER FOR COUNTER
BD9D ADB0BE   H4     LDA RESLO          ; LOAD LOW BYTE
BDA0 18              CLC
BDA1 79ABBE          ADC TBL1,Y         ; ADD FROM TABLE
BDA4 8DB0BE          STA RESLO          ; SAVE IT
BDA7 ADB1BE          LDA RESHI          ; LOAD HIGH BYTE
BDAA 79ACBE          ADC TBL2,Y         ; ADD FROM TABLE
BDAD 8DB1BE          STA RESHI          ; SAVE IT
BDB0 CA              DEX                ; DECREMENT COUNTER
BDB1 D0EA            BNE H4             ; LOOP TIL DONE
BDB3 AEB3BE          LDX COUNT          ; FETCH # OF DIGITS
BDB6 88       H5     DEY
BDB7 CA              DEX                ; BUMP DIGIT COUNTER
BDB8 10DA            BPL H3             ; LOOP 'TIL DONE
BDBA ADB1BE   PRINT  LDA RESHI          ; FETCH HIGH BYTE
BDBD F003            BEQ PR10           ; IF 0 SKIP IT
BDBF 20922D          JSR PRBYTE         ; NON-0, SO PRINT IT
BDC2 ADB0BE   PR10   LDA RESLO          ; FETCH LOW BYTE
BDC5 20922D          JSR PRBYTE         ; PRINT IT
BDC8 206A2D          JSR CRLF           ; DO A <CR> <LF>
BDCB 4C44BC          JMP UP1            ; RETURN TO BASIC
                     ;
                     ; ROUTINE TO CONVERT HEX TO DECIMAL
                     ;
BDCE C8       CONV   INY                ; BUMP POINTER 1
BDCF A200            LDX #$00           ; INIZ COUNTER
BDD1 8EB2BE          STX FIFTH
BDD4 8EB1BE          STX RESHI
BDD7 B1C7     H10    LDA (TXTPTR),Y     ; FETCH CHARACTER
BDD9 C930            CMP #'0            ; IS IT A NUMBER?
BDDB 9011            BCC H12            ; NO! CALCULATE
BDDD 38              SEC
BDDE E930            SBC #'0            ; STRIP OFF ASCII
BDE0 C90A            CMP #$0A
BDE2 9002            BCC H11
BDE4 E907            SBC #$07
BDE6 9DB5BE   H11    STA INBUF,X        ; SAVE VALUE
BDE9 E8              INX                ; BUMP COUNTERS
BDEA C8              INY
BDEB 4CD7BD          JMP H10            ; AND LOOP
BDEE CA       H12    DEX                ; BUMP BACK ONE
BDEF 98              TYA                ; X-FER # OF CHARS
BDF0 18              CLC
BDF1 65C7            ADC TXTPTR         ; AND UPDATE BASIC
BDF3 85C7            STA TXTPTR
BDF5 BDB5BE          LDA INBUF,X        ; LOAD ONE'S VALUE
BDF8 C90A            CMP #$0A           ; ADJUST FOR HEX
BDFA 3006            BMI H30
BDFC 38              SEC
BDFD E90A            SBC #$0A
BDFF 18              CLC
BE00 6910            ADC #$10
BE02 8DB0BE   H30    STA RESLO          ; ALWAYS SAVE IT
BE05 E000            CPX #$00           ; ARE WE DONE?
BE07 F021            BEQ PR             ; YES! BRANCH
BE09 CA              DEX                ; NO, BUMP COUNTER
BE0A A000            LDY #$00           ; INIZ COUNTER
BE0C 8EB4BE          STX POINT
BE0F AEB4BE   H13    LDX POINT          ; FETCH POINTER
BE12 BDB5BE          LDA INBUF,X        ; GET CHAR FROM BUFFR
```

```
BE15 F007           BEQ H15                 ; IF ZERO, SKIP IT
BE17 AA             TAX
BE18 203BBE   H14   JSR ADD                 ; DO ADDUP ROUTINE
BE1B CA             DEX
BE1C D0FA           BNE H14
BE1E ADB4BE   H15   LDA POINT
BE21 F007           BEQ PR                  ; BRANCH WHEN DONE
BE23 CEB4BE         DEC POINT               ; BUMP COUNTERS
BE26 C8             INY
BE27 4C0FBE         JMP H13                 ; AND LOOP
BE2A ADB2BE   PR    LDA FIFTH               ; FETCH 5TH DIGET
BE2D F006           BEQ PR1                 ; SKIP IF ZERO
BE2F 18             CLC
BE30 6930           ADC #$30                ; ADJUST TO ASCII
BE32 204323         JSR CHROUT              ; AND PRINT IT
BE35 4CBABD   PR1   JMP PRINT               ; FINISH ELSEWHERE
BE38 F8       ADD   SED                     ; WE'RE USING BCD HER
BE39 ADB0BE         LDA RESLO               ; FETCH LOW BYTE
BE3C 18             CLC
BE3D 79A2BE         ADC LOTBL,Y             ; ADD FROM TABLE
BE40 8DB0BE         STA RESLO               ; SAVE RESULT
BE43 9003           BCC ADD1                ; ADJUST FOR OVRFLO
BE45 EEB1BE         INC RESHI
BE48 ADB1BE   ADD1  LDA RESHI               ; FETCH HIGH BYTE
BE4B 18             CLC                     ; AND CONTINUE AS
BE4C 79A5BE         ADC HITBL,Y             ; BEFORE
BE4F 8DB1BE         STA RESHI
BE52 9003           BCC ADD2
BE54 EEB2BE         INC FIFTH
BE57 D8       ADD2  CLD                     ; RESET BCD FLAG
BE58 60             RTS                     ; AND GO BACK
                    ; ROUTINE TO DUMP SCREEN TO PRINTER
00F1=               LOC=$F1
BE59 A901     P1    LDA #$01                ; LOAD PRINTER DEVICE
BE5B 8D2223         STA 8994                ; SAVE IN OUTPUT FLAG
BE5E A900           LDA #$00                ; INIZ LOC TO SCREEN
BE60 85F1           STA LOC                 ; ADDRESS
BE62 A9D0           LDA #$D0
BE64 85F2           STA LOC+1
BE66 A000           LDY #$00                ; INIZ POINTER
BE68 B1F1     P2    LDA (LOC),Y             ; FETCH CHARACTER
BE6A C920           CMP #'                  ; CHECK FOR GRAPHICS
BE6C 1005           BPL P3                  ; CHARACTERS
BE6E A920           LDA #'                  ; AND PRINT A BLANK
BE70 4C7ABE         JMP P5                  ; INSTEAD
BE73 18       P3    CLC
BE74 C97F           CMP #$7F
BE76 3002           BMI P5
BE78 A920           LDA #'
BE7A 204323   P5    JSR CHROUT              ; PRINT!
BE7D C8             INY                     ; BUMP POINTERS
BE7E C040           CPY #$40
BE80 D0E6           BNE P2
BE82 A000           LDY #$00
BE84 A5F1           LDA LOC
BE86 18             CLC
BE87 6940           ADC #$40
BE89 85F1           STA LOC
BE8B A5F2           LDA LOC+1
BE8D 6900           ADC #0
BE8F 85F2           STA LOC+1
BE91 206A2D         JSR CRLF                ; END OF LINE
BE94 A5F2           LDA LOC+1
BE96 C9D8           CMP #$D8
BE98 D0CE           BNE P2
BE9A A902           LDA #$02                ; RESTORE OUTFLAG
BE9C 8D2223         STA 8994
BE9F 4C39BC         JMP UPDATE              ; AND GO BACK
BEA2 16       LOTBL .BYTE $16,$56,$96
BEA3 56
BEA4 96
BEA5 00       HITBL .BYTE $00,$02,$40
BEA6 02
BEA7 40
BEA8 10       TBL1  .BYTE $10,$E8,$64,$0A
BEA9 E8
BEAA 64
BEAB 0A
BEAC 27       TBL2  .BYTE $27,$03,$00,$00
BEAD 03
BEAE 00
BEAF 00
BEB0 00       RESLO .BYTE $00
BEB1 00       RESHI .BYTE $00
BEB2 00       FIFTH .BYTE $00
BEB3 00       COUNT .BYTE $00
BEB4 00       POINT .BYTE $00
BEB5=               INBUF=*
```


POOL 1.5™
ATARI™ OWNERS,
NOW YOU TOO
CAN PLAY POOL 1.5
ON YOUR ATARI
48K DISK SYSTEM.
ALSO AVAILABLE FOR
THE APPLE II™ 48K
DISK SYSTEM.
PRICE: $34.95
High resolution graphics - Life like motion.
Instant replay on any shot.
Superfine aiming.
Pool room sound effects
Choice of 4 popular games
See your local computer dealer or order directly from IDSI. Specify ATARI or APPLE.
IDSI
P.O. BOX 1658
LAS CRUCES, NM
88004
Ph. (505)522-7373
ATARI is the registered trademark of ATARI Inc.
APPLE II is the registered trademark of APPLE Computer Inc.
POOL 1.5 is trademark of IDSI.

Microbes and Updates

Roland E. Guilbault of Atkinson, New Hampshire, sent this note:

"Sorting" by William R. Reese (MICRO 39:29) is just what I needed to implement a record cataloging program that I am designing. Although the article is good, the listings have many typographical errors. The corrected lines follow:

```
LOAD SINGLE.SORT1

105 V$(I) = "4": FOR J = 1 TO 8:
    V$(I) = V$(I) + STR$ ( INT (10 *
    RND (1) )): NEXT J
112 REM SORT START HERE
115 V$(N + 1) = "999999999":
    SS%(N + 1) = N + 1
180 Q = SK(ST):P = SK(ST - 1)
200 FOR I = 0 TO N: PRINT I; TAB( 5);
    SS%(I); TAB( 10);V$(SS%(I)):
    NEXT
1145 VI = SS%(P):VH$ = V$(VI):I =
     P:J = K

LOAD DOUBLE.SORT

105 W$(I) = "4": FOR J = 1 TO
    8:W$(I) = W$(I) + STR$ ( INT (10
    * RND (1) )): NEXT J
130 IF P > = Q THEN 170
200 FOR I = 0 TO N: PRINT I; TAB( 5);
    SS%(I); TAB( 10);V$(SS%(I)); TAB(
    20 );W$(SS%(I)): NEXT
1160 J = J - 1: IF V$(SS%(J))< VH$
     THEN 1170
1161 IF V$(SS%(J)) > VH$ GOTO 1160
1162 IF W$(SS%(J)) < = W$(VI)
     GOTO 1170
1171 IF V$(SS%(I))< VH$ GOTO 1170
1190 GA = SS%(I):GB = SS%(J)
```

The program can be optimized by changing line 1190 and 1195 to the following:

```
1190 GA = SS%(I)
1195 SS%(I) = SS%(J) : SS%(J) = GA
     : GOTO 1160
```

Maurice Bernstein of Panorama City, California, wrote in with these modifications to "Othello" by Charles F. Taylor, Jr. (42:63):

Over the past couple of years I have found very few game programs modified for use with modems. I have not seen games for modem use in computer stores. Yet it is the very use of modem communication that could add another dimension to the recreational use of the microcomputer, expanding game competition to outside the household.

I have found that the game program "Othello" by Charles F. Taylor, Jr., in the November 1981 issue of MICRO, is a good example of a competitive Lo-Res game which can be easily modified for modem use.

The additions and minor modification noted below are based on the following assumptions:

1. Both Apple II computers using this Applesoft BASIC program have DOS.
2. Both computers use a Hayes Micromodem II with the card in slot #3, and are at the outset established in terminal-terminal half-duplex mode.
3. Both computers have the modified "Othello" program loaded in memory and the players have agreed who will move first.

First, each player types a CTRL A, CTRL X sequence to leave the terminal mode. Then each player types 'RUN', and when prompted, types his turn number. The players type the legal coordinates, in turn, as if running the program without a modem. At the game's end, lines 750 and 760 from the original program prompt whether or not to play again. These can be left out.

These routines could be used for other games where the controlling characters would be string expressions of paddle values (e.g., STR$(PDL(0))). These values would transmit and then recover the integer value on reception using the VAL command. If any random values are generated in the program, the random value as an ASCII character must come from one computer only and be transmitted to the other computer. Otherwise, you won't get synchronous graphics.

OTHELLO by Charles F.Taylor

Modifications for modem use by Maurice Bernstein, M.D., December 1981.

```
INITIALIZE DOS COMMAND
Add line 152 D$=CHR$(4)
INDICATE TURN SEQUENCE
Add line 792 PRINT" WHICH TURN DO
YOU WANT? 1 OR 2": INPUT TURN$: IF
TURN$< >'1" AND TURN$< >"2"
THEN HOME: GOTO 792
FIND WHETHER TO TRANSMIT AND IF
SO GO TO SUBROUTINE
Change line 1340 IF TURN=
INT(VAL(TURN$)) THEN GOSUB 2000:
GOTO 1350

RECEIVE ROUTINE
Add line 1345 PRINT D$; "PR #0"
Add line 1346 PRINT D$; "IN #3"
Add line 1348 INPUT MOVE$
TRANSMIT SUBROUTINE
Add line 2000 PRINT D$; "IN #0"
Add line 2002 INPUT MOVE$
Add line 2010 PRINT D$; "PR #3"
Add line 2020 PRINT MOVE$
Add line 2030 PRINT D$ ;"PR#0"
Add line 2040 RETURN
```

If you've discovered a microbe or developed an update, send your information to Microbes/Updates, MICRO, P.O. Box 6502, Chelmsford, MA 01824.

MICRO

SPEED POWER EFFICIENCY for OSI

65D3 SYSTEMS

FBASIC: BASIC Compiler $155/$10

- **FAST** efficient machine code written with the ease of BASIC.
- **SPEED**-optimized, native-code compiler.
- **INTEGER** subset of OSI-supplied BASIC.
- **DISK** based: No problem with size of source or object files.
- **EXTENSIONS** to BASIC:
  - Simple interface to system hardware and software.
  - Direct access to 6502 registers.
  - Array initialization.
  - Optional absolute array locations.
  - WHILE and other structures.
  - Simple technique for combining compiler and interpreter advantages.
- **FULL** system: utilities (plus source), manual, and many useful examples.

R-EDIT: Edit any program or text with ease! $40

- **FULL CURSOR** control. Edit anywhere on the screen.
- **INSERT**, replace, add, delete.
- **ONE KEY** stroke and you're editing.
- **BASIC**, assembler, etc. can all be edited without reloading editor.
- **RAM**-resident: Always ready!
- **SYSGEN** relocates R-EDIT and customizes.

SPUL-65: Printer Spooler $95/$10 Virtual Indirect File

- **STOP WAITING** for your printer!
- **PROCESS** words, write programs...all while printing!
- **QUEUE** lets you pile on print jobs.
- **MULTIPLE COPIES** printed with top and bottom page margins.
- **SYSGEN** relocates SPUL65 and gives extensive customization.
- **INDIRECT FILE** commands produce disk files giving you:
  - A virtually unlimited temporary file.
  - A link between incompatible files; for example, use WP-2 for extensive BASIC editing.
  - Ability to merge multiple program segments.

XREF: BASIC Cross Referencer $25

- **TABULATES:**
  - Referenced line numbers.
  - Variable names (numeric, string, array).
  - Defined functions.
- **FAST** machine language program.
- **DISK** based: Handles large BASIC source files on any drive.

CP/M to OSI Disk Translation

Frustrated by all those good CP/M disks that won't run on your OSI CP/M system? It's that special OSI disk format! And we can fix that. Just send us your disk, $15, and you'll soon have an OSI compatible disk.

drc Software

Data Resource Corporation
Suite 204
1040 Lunaai Street
Kailua, HI 96734 (808) 261-2012

Manual orders applied to software purchases. Programs supplied on 8-in, single-density, single-sided disks. Hawaii residents add 4% tax.

Microsoft BASIC-in-ROM Extensions

PRINT AT and CALL functions are added to OSI BASIC-in-ROM using Ed Carlson's technique. Minor modifications may be made to apply these functions to other machines.

Michael M. Mahoney
4136 NE 14th Street
Portland, Oregon 97211

This article is primarily intended for OSI BASIC-in-ROM systems such as the C2-4P, C4P or the C1P. However, with modifications the program should be adaptable to other Microsoft 6502 BASICs, such as PET BASIC or Applesoft. It can even be modified for use with OSI Disk BASIC.

The programs were written on an OSI C2-4P with 20K RAM, using BASIC-in-ROM and OSI's 65XX Assembler.

In the June 1980 issue of MICRO (25:15), Ed Carlson presented an article entitled "Put Your Hooks Into OSI BASIC" which explained a method of adding new commands (pseudo keywords) to OSI BASIC-in-ROM. His method consisted of altering the character parser, located in page zero, to recognize and process the additional commands. I thought Mr. Carlson's idea was excellent and searched the succeeding issues of MICRO, and other magazines, for additional articles or routines to use with his method. I never found one, so I wrote one myself.

This article adds two new pseudo keywords (PRINT AT and CALL) and fixes a minor problem I discovered in the original version. It also presents a more elegant solution to the dual keyword flag situation.

How it Works

OSI BASIC-in-ROM, and most other Microsoft 6502 BASICs, contain a page zero resident subroutine used by all the other routines in BASIC to fetch characters, one at a time, from the BASIC statement being executed. This routine is called a character parser and is usually referred to as 'CHRGET'.

The procedure Mr. Carlson and I used alters this routine to jump to some code of our own. It will then recognize and execute the pseudo keywords.

The only tricky part is that because there are two modes of operation in BASIC, (Immediate and Run), Mr. Carlson uses two separate keyword flags to allow the pseudo keywords to be entered into the text workspace. A percent sign (%) and a pound sign (#) were chosen as the keyword flags. When entering a BASIC statement containing a pseudo keyword, you would type a percent sign, and to immediately execute one you would type a pound sign. This is somewhat awkward.

When the altered parser sees a pound sign, it executes the appropriate routine immediately. When it sees a percent sign, the parser routine changes it to a pound sign without execution, allowing the command to be placed into the text area. Then the next time the line is passed through the parser, it recognizes the pound sign and executes the pseudo keyword.

The Problem

Since the parser changes the percent signs to pound signs, when SAVEing programs containing pseudo keywords to tape, the pound sign is the flag SAVEd. So, when LOADing the program back from tape, the parser sees the pound sign and executes the pseudo keyword immedately instead of storing it in the text area.

Depending on the pseudo keyword encountered, this may cause the computer to "hang" or miss several characters or lines of your program. Luckily, there are several methods to consider for preventing this. You can:

1. always LOAD your programs before LOADing and implementing the extensions;
2. check the LOAD FLAG ($0203) and execute pseudo keywords only when the LOAD flag is "OFF";
3. add special pseudo keywords to disable and enable the extensions;
4. add a special flag that can be set or reset from BASIC to control whether or not execution should be permitted.

Option 1 is not very practical since it would require resetting the computer and reLOADing the extensions every time you wished to run a different program.

Option 2 would work, except when the program LOADs data from tape. Any pseudo keywords between turning on and turning off LOAD not only would not execute, but would cause a syntax error.

Option 3 is a workable solution, but it would take a relatively large amount of code to implement.

This leaves Option 4 — the use of a flag — which is the method I chose. It has the advantage of requiring little extra code, and provides an easy way to enable or disable the extensions from BASIC.

By simply entering

POKE 250,1 < RETURN >

in the immediate mode, the extensions are disabled. And by entering

```
POKE 250,0 < RETURN >
```

they are enabled.

Using a flag also relieves you of the necessity of having two different keyword flags, thus saving some code and removing the awkwardness of remembering which flag to use. Now to enter pseudo keywords either from the keyboard or tape, you must first disable the extensions by POKEing the flag to "1". To RUN a program or to do an immediate mode pseudo keyword, you must enable the extensions by POKEing the flag to zero. I usually place the appropriate POKEs at the beginning and end of my programs containing pseudo keywords.

The CALL Pseudo Keyword

The "CALL" command is identical in function to the USR(X) command, as it is used to transfer control to a machine language routine and then return to BASIC at the next statement. To use the USR function you first need to set the USR vector at 11 and 12 decimal ($0B and $0C) to point to the entry point of the machine language routine before performing the USR. This results in a line of code such as

```
POKE 11,0:POKE 12,253:X=
  USR(X)
```

To do this, you must convert the address from hex to decimal, then convert it to the standard 6502 two-byte low, high format in decimal so that it can be POKEd in BASIC.

I created the CALL command to perform all that for me. With the CALL, a hexadecimal literal, a decimal address literal, or a numeric variable containing a decimal address, can be used as the argument. (Sorry, hexadecimal addresses cannot be assigned using a string variable.)

The format for the CALL command is

```
#U ADDRESS
```

The # is the pseudo keyword flag, the U is the CALL pseudo keyword, and ADDRESS is the entry point address of the machine language routine. ADDRESS may be a decimal number such as 64783, a numeric variable name such as N, a hexadecimal literal such as $FD00, or any valid numeric expression such as A(J+2) or 3*J.

Note that when using the hexadecimal notation option, the address *must* be preceded by a dollar sign ($).

***Listing 1:* CALL Command Examples**

```
100 REM                        CALL COMMAND EXAMPLES

110 :
120 POKE 250,0          : REM  ENABLE EXTENSIONS
130 :
140 X=64768:X(1)=X:X(2)=X-768
150 :
160 #U $FD00            : REM - HEXADECIMAL LITERAL

170 #U 64768            : REM - DECIMAL LITERAL

180 #U X                : REM - DECIMAL VARIABLE

190 #U X(1)             : REM - DECIMAL ARRAY ELEMENT

200 #U X(2)+768         : REM - DECIMAL EXPRESSION
210 :
220 POKE 250,1 : END
```

Listing 2: PRINT AT Command

```
300 REM                             PRINT AT COMMAND

310 :

320 REM     INPUT AT WITH SIZE OPTION

330 :

340 CUR$=CHR$(161) : REM CURSOR CHARACTER

350 PROMPT$="What is your name ?" : SIZE = 30
360 :

370 POKE 250,0 : REM -- ENABLE EXTENSIONS
380 :

390 #C : REM CLEAR SCREEN

400 #P(12,5) PROMPT$ : REM --PRINT PROMPT
410 I=1:NAME$=""
420 :
430 #P(12,25+I) CUR$    : REM--PRINT CURSOR CHAR

440 #U $FD00            : REM--POLL KEYBOARD

450 X=PEEK(531):X$=CHR$(X) : REM--GET KEY PRESSED
460 :
470 IF X<>13 THEN 490
480 #P(12,25+I) " ":GOTO 550  : REM--ERASE CURSOR
490 IF X<32 OR X>122 THEN 430 : REM IGNORE INVALID CHARS
500 :

510 #P(12,25+I) X$ : REM PRINT CHARACTER

520 NAME$=NAME$+X$ : REM CONCATENATE NAME
530 I=I+1 : IF I<SIZE THEN 430 : REM CHECK FOR SIZEERROR
540 :
550 NAME$="HELLO THERE "+NAME$+"!"

560 #P(20,5) NAME$
570 :

580 POKE 250,1 : END
```

Listing 3

```
; BASIC EXTENSIONS  VER 1.3    APR 1981
;
;           BY MICHAEL M. MAHONEY
;
; ADAPTED FROM 'PUT YOUR HOOKS INTO OSI BASIC'
;    BY ED CARLSON  IN MICRO #25:15
;
; PROCEDURE AND CLEAR SCREEN BY ED CARLSON
;
; CALL & PRINT AT COMMANDS, AND
; ENABLE/DISABLE FLAG  BY MIKE MAHONEY
;
;      EQUATES
;
PARSER  =  $00BC   ; GETCHR ROUTINE
GETCUR  =  $00C2   ; ENTRY - GET CURRENT CHAR
POINTR  =  $00C3   ; CODE POINTER FOR PARSER
;
; PAGE ZERO LOCATIONS USED BY EXTENSIONS
;
ROLBYT  =  $E7     ; USED FOR BIN CONVERS
D.FLAG  =  $FA     ; DISABLE FLAG (250 DECIMAL)
TEMP2   =  $FC
TEMP1   =  $FE
;
; BASIC ROUTINES AND POINTERS
;
VARTYP  =  $5F     ; VAR TYPE FLAG (>$7F = STRING
STRPTR  =  $71     ; BASIC STRING PTR
ERRPRT  =  $A256   ; PRINT ERROR MESSAGE
EXPR.1  =  $AAAD   ; EVAL EXPRESS WITH TM CHECK
EXPR.2  =  $AAC1   ; EVAL EXPRESS - NO TM CHECK
RPAREN  =  $ABFB   ; SN ERR IF NOT ')'
LPAREN  =  $ABFE   ; SN ERR IF NOT '('
COMMA   =  $AC01   ; SN ERR IF NOT ','
SETSTR  =  $B0AE   ; SCAN & SETUP STRING
SETPTR  =  $B2B6   ; GET PTRS AND LEN OF STRING
EXPR.8  =  $B3AE   ; 8 BIT EXPRESS EVALUATION
FIX     =  $B408   ; CONVERT FL ACC TO FIXED
FP.STR  =  $B96E   ; CONVERT FL ACCUN TO STRING
;
LEGAL   =  $FE93   ; CHK FOR LEGAL ASCII HEX
;
```

(Continued)

Otherwise the CALL routine will evaluate the address as either a numeric literal or variable, or report a syntax error. Also, the hex address *must* be four valid ASCII hex characters, so remember to pad with leading zeros when necessary. Otherwise you'll get a syntax error.

Listing 1 gives some examples of valid CALL command formats.

The PRINT AT Pseudo Keyword

"PRINT AT" allows you to specify exactly where on the screen to print any single value, without disturbing other areas, and without scrolling the screen. This is highly desirable, especially in any type of data processing input procedure, since formatted screens are possible. In games you can use PRINT AT to maneuver pre-defined figures around the screen easily and rapidly. Or, coupled with the CALL command, an INPUT AT with size option can be simulated as in listing 2.

The format for the PRINT AT command is

#P (L,C) VALUE

#P is the PRINT AT pseudo keyword, L is the line number of the screen (0 to 31), C is the column number (0 to 63), and VALUE is the value to print.

Both L and C may be numeric literals, numeric variables, or any valid numeric expression, within the specified ranges for each. VALUE may be any single numeric literal, string literal, numeric variable, string variable or valid numeric or string expression. Parentheses must surround the line and column specifications, and they must be separated by a comma. All spaces shown are optional; there may be more if desired.

Listing 2 shows examples of a number of valid PRINT AT formats as they might appear in a program.

The Program

The assembly language program is shown in listing 3. I'd like to point out that in addition to the PRINT AT and CALL discussed here, there is also a "CLEAR SCREEN" command (#C) available. Also, please note that I have

Listing 3 *(Continued)*

```
                     *  =  $1EDO
                    ;
                    ;
                    ; ALTERS THE PARSER TO JUMP TO
                    ; THE NEW ROUTINE
                    ;
1EDO A94C           ENTRY  LDA #$4C       ; 'JMP'
1ED2 85BC                  STA PARSER
1ED4 85FB                  STA $00FB      ; FOR 'CALL'
1ED6 A9EB                  LDA #STARLO    ; ADDR LO OF NEW RTNE
1ED8 85BD                  STA PARSER+1
1EDA A91E                  LDA #STARHI    ; ADDR HI OF NEW RTNE
1EDC 85BE                  STA PARSER+2
1EDE A9EA                  LDA #$EA       ; 'NOP'
1EE0 85BF                  STA PARSER+3
1EE2 85C0                  STA PARSER+4
1EE4 85C1                  STA PARSER+5
1EE6                ;
1EE6 A900                  LDA #$00
1EE8 85FA                  STA D.FLAG
1EEA                ;
1EEA 60                    RTS            ; BACK TO BASIC
1EEB                ;
1EEB                ; MAIN ROUTINE
1EEB                ;
1EEB E6C3           START  INC POINTR     ; INCREMENT CODE POINTER
1EED D002                  BNE MAIN.1
1EEF E6C4                  INC POINTR+1
1EF1                ;
1EF1 A5FA           MAIN.1 LDA D.FLAG     ; EXTENSIONS DISABLED?
1EF3 D008                  BNE MAIN.X     ; YES SO SKIP CHECK
1EF5                ;
1EF5 A000                  LDY #$00
1EF7 B1C3                  LDA (POINTR),Y ; GET CHARACTER
1EF9 C923                  CMP #'#        ; IS IT A POUND SIGN?
1EFB F003                  BEQ EXTEND
1EFD                ;
1EFD 4CC200         MAIN.X JMP GETCUR     ; BACK TO PARSER
1F00                ;
1F00                STARHI  =  START/256
1F00                TMP     =  STARHI*256
1F00                STARLO  =  START-TMP
1F00                ;
1F00                ; PSEUDO-KEYWORD DECODING
1F00                ;
1F00 20BC00         EXTEND JSR PARSER     ; GET NEXT CHAR
1F03 A000                  LDY #$00
1F05 A2FF                  LDX #$FF
1F07 E8             X.LOOP INX
1F08 BD1E1F                LDA EXTTBL,X   ; GET PSEUDO-KEYWORD
1F0B F0F0                  BEQ MAIN.X     ; END TBL SO EXIT
1F0D D1C3                  CMP (POINTR),Y ; CMP TO CHAR IN STMT
1F0F D0F6                  BNE X.LOOP     ; CHECK NEXT KEYWORD
1F11                ;
1F11                ; FOUND A MATCH SO GET ADDR
1F11                ;
1F11 BD261F                LDA ADRLO,X    ; TRANSFER ADDRESS OF
1F14 85FC                  STA TEMP2      ; OF KEYWORD RTNE
1F16 BD221F                LDA ADRHI,X    ; TO PG ZERO LOCATION
1F19 85FD                  STA TEMP2+1    ; FOR JUMP
1F1B                ;
1F1B 6CFC00                JMP (TEMP2)    ; GOTO KEYWORD RTNE
1F1E                ;
1F1E                ; PSUEDO-KEYWORD AND ADDRESS TABLES
1F1E                ;
1F1E                ; VALID SINGLE CHAR 'KEYWORDS'
1F1E                ; END TABLE WITH NULL  ($00)
1F1E                ;
1F1E 43             EXTTBL .BYTE 'CPU',$00
1F1F 50
1F20 55
1F21 00
1F22                ;
1F22                ; HIGH BYTE OF ROUTINE'S ENTRY ADDRESS
1F22                ;       END WITH NULL  ($00)
1F22                ;

1F22 1F             ADRHI  .BYTE C.HI,P.HI,U.HI
1F23 1F
1F24 1F
1F25 00                    .BYTE $00
1F26                ;
1F26                ; LOW BYTE OF ROUTINE'S ADDRESS
1F26                ;       END WITH NULL  ($00)
1F26                ;
1F26 2A             ADRLO  .BYTE C.LO,P.LO,U.LO
1F27 43
```

Listing 3 *(Continued)*

```
1F28 AB
1F29 00              .BYTE $00
1F2A                ;
1F2A                ; SCREEN CLEAR ROUTINE
1F2A                ;
1F2A A208    C.RTNE LDX #$08       ; # PAGES ($04 FOR C1P)
1F2C A9D0           LDA #$D0       ; SET POINTER TO START
1F2E 85FD           STA TEMP2+1    ; OF SCREEN
1F30 A000           LDY #$00
1F32 84FC           STY TEMP2
1F34 A920           LDA #$20        ; SPACE CHARACTER
1F36 91FC    C.LOOP STA (TEMP2),Y ; BLANK LOC ON SCREEN
1F38 C8             INY
1F39 D0FB           BNE C.LOOP
1F3B E6FD           INC TEMP2+1
1F3D CA             DEX
1F3E D0F6           BNE C.LOOP      ; CHECK IF DONE
1F40                ;
1F40 4CBC00         JMP PARSER      ; BACK TO BASIC
1F43                ;
1F43         C.HI    = C.RTNE/256
1F43         TMP     = C.HI*256
1F43         C.LO    = C.RTNE-TMP
1F43                ;
1F43                ; PRINT AT ROUTINE
1F43                ;
1F43 A9D0    P.RTNE LDA #$D0        ; SET UP PTR TO LINE #0
1F45 85FD           STA TEMP2+1    ;    $D000
1F47 A900           LDA #$00
1F49 85FC           STA TEMP2
1F4B                ;
1F4B 20BC00         JSR PARSER     ; GET NEXT CHAR
1F4E 20FEAB         JSR LPAREN     ; CHECK FOR OPEN PAREN
1F51 20AEB3         JSR EXPR.8     ; GET 8 BIT ARG IN X
1F54 E020           CPX #$20       ; CHECK LINE # <=31
1F56 B01B           BCS FN.ERR     ; NO- SO CAUSE FN ERROR
1F58                ;
1F58 E8             INX
1F59 CA      INCLIN DEX            ; INCREMENT PTR TO START
1F5A F00D           BEQ GETCOL     ; OF CORRECT LINE
1F5C 18             CLC
1F5D A940           LDA #$40       ; LINE SIZE ($20 FOR C1P)
1F5F 65FC           ADC TEMP2
1F61 85FC           STA TEMP2      ; INCREMNT PTR BY ONE LINE
1F63 90F4           BCC INCLIN
1F65 E6FD           INC TEMP2+1
1F67 D0F0           BNE INCLIN   ; FORCED BRANCH
1F69                ;
1F69 2001AC  GETCOL JSR COMMA    ; CHECK FOR COMMA
1F6C 20AEB3         JSR EXPR.8   ; GET 8 BIT ARG IN X
1F6F E040          CPX #$40     ; CHECK COL <=63 ($20 FOR C1P)
1F71 9005          BCC COLOK    ; IT'S OK
1F73               ;
1F73 A208   FN.ERR LDX #$08     ; FN MESSAGE OFFSET
1F75 4C56A2        JMP ERRPRT   ; ERROR PRINTER
1F78               ;
1F78 8A     COLOK  TXA
1F79 18            CLC
1F7A 65FC          ADC TEMP2  ; ADD COLUMN TO PTR
1F7C 85FC          STA TEMP2
1F7E 9002          BCC GETARG
1F80 E6FD          INC TEMP2+1
1F82               ;
1F82 20FBAB GETARG JSR RPAREN  ; CHECK FOR CLOSE PAREN
1F85               ;
1F85               ; NOW GET VALUE TO PRINT
1F85               ;
1F85 20C1AA VARNAM JSR EXPR.2  ; EXPRESSION HANDLER NO TM
1F88 245F          BIT VARTYP  ; STRING OR NUMERIC ?
1F8A 3006          BMI STRING
1F8C               ;
1F8C 206EB9 NUMERC JSR FP.STR  ; CONVERT TO ASCII STRING
1F8F 20AEB0        JSR SETSTR  ; SCAN AND SETUP STRING
1F92               ;
1F92 20B6B2 STRING JSR SETPTR  ; SET POINTERS & GET LENGTH
1F95 AA            TAX         ; PUT LENGTH OF STRING IN X
1F96 A000          LDY #$00
1F98 E8            INX
1F99               ;
1F99               ; NOW PRINT TO SCREEN
1F99               ;
1F99 CA     PRINT  DEX
1F9A F00C          BEQ P.EXIT      ; DONE WITH PRINTING
1F9C B171          LDA (STRPTR),Y ; GET CHAR TO PRINT
1F9E C90D          CMP #$0D       ; IGNORE CARRIAGE RETURNS
1FA0 F0F7          BEQ PRINT
```

(Continued)

made extensive use of existing routines in ROM BASIC, especially in the PRINT AT routine, and have tried to identify their functions in the listing. If you want to modify the program for other versions of Microsoft BASIC, you will need to replace these addresses with the corresponding ones for your machines.

Because the program is designed with no self-modifying sections, I can place it in ROM eventually. To accomplish this, certain page zero locations were used. These locations are not normally used by OSI ROM BASIC, but may be used on other machines. The locations used are $E7, and $FA through $FF.

Listing 4 contains a BASIC program that will load the extensions into the top of any size memory machine. It will also configure itself for either C1 or C2, alter the character parser, and set the ENABLE/DISABLE flag to 0 (the extensions are ENABLED). In addition, it will lower the top of memory, and then NEW itself.

MICRObits

Deadline for MICRObits: 20th of second month before publication; i.e., March 20th for May issue. Send typewritten copy with $25 per insertion. (Subscribers: first ad at $10.)

COMPendium

The only monthly guide to 20+ current microcomputer magazines. Every article, editorial, hardware/software evaluation described. Listed by computer type, application. Copiously indexed. Subscription includes *Annual Reference.* Helps pinpoint useful information; perfect if you can't afford 20 subscriptions. Sample ($1.50); subscription ($18).

Epicurious
P.O. Box 129
Lincolndale, NY 10540

(Continued on page 92)

Listing 3 *(Continued)*

```
2090 1FA2 91FC            STA (TEMP2),Y   ; PUT IT ON SCREEN
2100 1FA4 C8              INY
2110 1FA5 4C991F          JMP PRINT       ; BACK FOR NEXT CHAR
2120 1FA8                 ;
2130 1FA8 4CC200   P.EXIT JMP GETCUR      ; EXIT TO BASIC
2140 1FAB                 ;
2150 1FAB          P.HI   =   P.RTNE/256
2160 1FAB          TMP    =   P.HI*256
2170 1FAB          P.LO   =   P.RTNE-TMP
2180 1FAB                 ;
2190 1FAB                 ; CALL ROUTINE
2200 1FAB                 ;
2210 1FAB 20BC00   U.RTNE JSR PARSER      ; GET NEXT CHAR
2220 1FAE C924            CMP #'$         ; HEX ADDRESS ?
2230 1FB0 F014            BEQ HEXADR
2240 1FB2                 ;
2250 1FB2 20ADAA   DECADR JSR EXPR.1      ; 16 BIT EXPR - TM CHECK
2260 1FB5 2008B4          JSR FIX         ; CONVERT FL ACCUM TO FIX
2270 1FB8 A511            LDA $11         ; TRANSFER ADDRESS TO
2280 1FBA 85FC            STA TEMP2       ; PG ZERO TEMPORARY LOC
2290 1FBC A512            LDA $12
2300 1FBE 85FD            STA TEMP2+1
2310 1FC0 20FB00          JSR $00FB       ; DO 'CALL'
2320 1FC3                 ;
2330 1FC3 4CC200          JMP GETCUR      ; BACK TO BASIC FOR DECIMAL
2340 1FC6                 ;
2350 1FC6 20D61F   HEXADR JSR HEXIN        ; GET BYTES 1&2-CONVERT
2360 1FC9 85FD           STA TEMP2+1  ; TO BINARY - PUT IN TEMP HI
2370 1FCB 20D61F          JSR HEXIN   ; GET BYTE 3&4-CONVERT TO
2380 1FCE 85FC            STA TEMP2   ; BINARY - PUT IN TEMP LO
2390 1FD0 20FB00          JSR $00FB   ; DO 'CALL'
2400 1FD3 4CBC00          JMP PARSER   ; BACK TO BASIC
2410 1FD6                 ;
2420 1FD6 20D91F   HEXIN  JSR HEXNXT
2430 1FD9 20BC00   HEXNXT JSR PARSER
2440 1FDC 2093FE          JSR LEGAL       ; LEGAL ASCII HEX ?
2450 1FDF 300F            BMI SN.ERR      ; NO-SO CAUSE SYNTAX ERROR
2460 1FE1                 ;
2470 1FE1 0A              ASL A            ; CONVERT TO BINARY
2480 1FE2 0A              ASL A
2490 1FE3 0A              ASL A
2500 1FE4 0A              ASL A
2510 1FE5 A004            LDY #$04
2520 1FE7 2A       ROLLIT ROL A
2530 1FE8 26E7            ROL ROLBYT
2540 1FEA 88              DEY
2550 1FEB D0FA            BNE ROLLIT
2560 1FED A5E7            LDA ROLBYT
2570 1FEF 60              RTS
2580 1FF0                 ;
2590 1FF0          U.HI   =   U.RTNE/256
2600 1FF0          TMP    =   U.HI*256
2610 1FF0          U.LO   =   U.RTNE-TMP
2620 1FF0                 ;
2630 1FF0 A202     SN.ERR LDX #$02        ; SN MESSAGE OFFSET
2640 1FF2 4C56A2          JMP ERRPRT      ; ERROR PRINTER
2650 1FF5                 ;
2660 1FF5                 .END
```

Listing 4

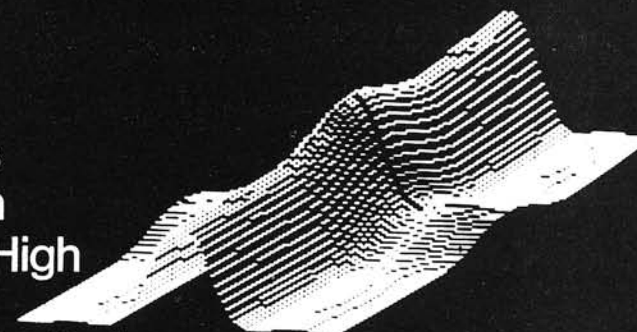
```
100 REM   BASIC EXTENSIONS VER 1.3
110 REM    AUTO CONFIGURATION  C1/C2 & MEM SIZE
120 :
130 REM     by MICHAEL M. MAHONEY   APRIL 1981
140 :
150 REM    ADAPTED FROM 'PUT YOUR HOOKS INTO OSI BASIC'
160 REM               by ED CARLSON IN MICRO #25
170 :
180 ME=PEEK(133)+256*PEEK(134):ME=ME-300
190 MH=INT(ME/256):ML=ME-(256*MH)
200 POKE 133,ML:POKE 134,MH:CLEAR: REM LOWER MEM & RESET PTRS
210 ME=PEEK(133)+256*PEEK(134)
220 :
230 FOR I=1 TO 293 : READ X : POKE ME+I,X : NEXT I
240 :
250 FOR I=1 TO 8 : READ X,Y : Y=Y+ME
260 YH=INT(Y/256) : YL=Y-(256*YH)
270 POKE ME+X,YL : IF X=8 THEN X=11
280 POKE ME+X+1,YH : NEXT I
281 :
282 FOR I=1 TO 3:READ X:Y=ME+X:YH=INT(Y/256):YL=Y-(256*YH)
283 POKE ME+82+I,YH:POKE ME+86+I,YL:NEXT I
290 :
300 T=20:X=PEEK(57088) : IF X<128 THEN 320: REM C2
310 POKE ME+92,4:POKE ME+143,32:POKE ME+161,32:T=5
320 MH=INT((ME+1)/256):ML=(ME+1)-(256*MH)
```

Listing 4 *(Continued)*

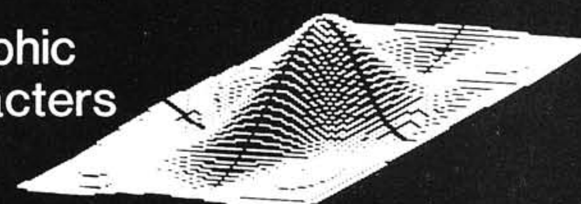
```
330 POKE 11,ML:POKE 12,MH : X=USR(X)
340 :
350 #C : #P(8,T)"BASIC EXTENSIONS   VER 1.3"
360 #P(9,T)"--------------------------"
370 #P(12,T)"  by Michael M. Mahoney"
380 #P(15,T)"EXTENSIONS NOW ENABLED"
390 #P(17,T)" TO DISABLE - POKE 250,1"
400 #P(19,T)" TO ENABLE  - POKE 250,0"
410 :
420 END
430 :
1010 DATA 169,76,133,188,133,251,169,235,133,189,169
1020 DATA 30,133,190,169,234,133,191,133,192,133,193
1030 DATA169,0,133,250,96,230,195,208,2,230,196
1040 DATA 165,250,208,8,160,0,177,195,201,35,240
1050 DATA 3,76,194,0,32,188,0,160,0,162,255
1060 DATA 232,189,30,31,240,240,209,195,208,246,189
1070 DATA 38,31,133,252,189,34,31,133,253,108,252
1080 DATA 0,67,80,85,0,31,31,31,0,42,67
1090 DATA 171,0,162,8,169,208,133,253,160,0,132
1100 DATA 252,169,32,145,252,200,208,251,230,253,202
1110 DATA 208,246,76,188,0,169,208,133,253,169,0
1120 DATA 133,252,32,188,0,32,254,171,32,174,179
1130 DATA 224,32,176,27,232,202,240,13,24,169,64
1140 DATA 101,252,133,252,144,244,230,253,208,240,32
1150 DATA 1,172,32,174,179,224,64,144,5,162,8
1160 DATA 76,86,162,138,24,101,252,133,252,144,2
1170 DATA 230,253,32,251,171,32,193,170,36,95,48
1180 DATA 6,32,110,185,32,174,176,32,182,178,170
1190 DATA 160,0,232,202,240,12,177,113,201,13,240
1200 DATA 247,145,252,200,76,153,31,76,194,0,32
1210 DATA 188,0,201,36,240,20,32,173,170,32,8
1220 DATA 180,165,17,133,252,165,18,133,253,32,251
1230 DATA 0,76,194,0,32,214,31,133,253,32,214
1240 DATA 31.133.252.32.251.0.76.188.0.32.217
1250 DATA 31,32,188,0,32,147,254,48,15,10,10
1260 DATA 10,10,160,4,42,38,231,136,208,250,165
1270 DATA 231,96,162,2,76,86,162
1280 :
1290 DATA 8,28,58,79,67,87,72,83,215,202,248,263
1300 DATA 253,263,264,266
1310 :
1320 DATA 91,116,220
```

MICRO

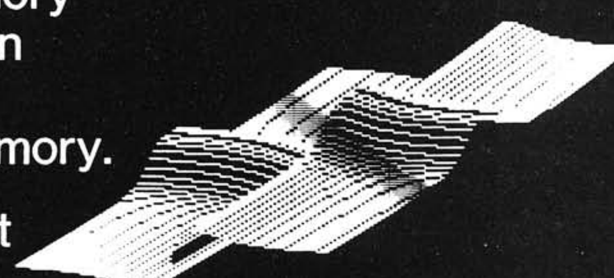
GRAPHICS
FOR OSI COMPUTERS



☆ You Can Produce The Images Shown Or Yours And Program Motion With Our 256 By 256 High Resolution Graphics Kit. Thats 65,536 Individually Controlled Points On Your TV Screen.



☆ Increase Column/Line Display. You Can Set Up Your Own Graphic Pixels Including Keyboard Characters And Unlimited Figures.



☆ This Kit Includes All Parts, Software And Assembly Instructions Required To Get Up And Running. The Included 8k Of 2114 Memory Is Automatically Available When Not Using The Graphics. Boot Up And See 8k More Memory.

☆ Adding The Kit Does Not Affect Your Existing OSI Graphics. Use Both At The Same Time Or Separately.

☆ Buy The Entire Kit, Including Memory, For $185.00 Or A Partial Kit For Less If You Have Parts. Board And Instructions $40.00. Instructions Include Software.

ME 01 10 ME

For This Kit Or A Catalog Of Other Kits, Software And Manuals Call Or Write:

MITTENDORF ENGINEERING
905 Villa Neuva Dr.
Litchfield Park, Az. 85340
(602)-935-9734

OSI TRS-80 COLOR-80 OSI

GALAXIAN - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95 OSI

LABYRINTH - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95. OSI

THE AARDVARK JOURNAL

FOR OSI USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.

2) Complete listings of two word processors for BASIC IN ROM machines.

3) Moving the Directory off track 12.

4) Listings for 20 game programs for the OSI.

5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues - $9.00

Vol. 2 (1981) 4 back issues and subscription for 2 additional issues - $9.00.

ADVENTURES!!!

For OSI, TRS-80, and COLOR-80. These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch".)

Adventures require 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 each.

ESCAPE FROM MARS (by Rodger Olsen)

This ADVENTURE takes place on the RED PLANT. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)

This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

TREK ADVENTURE (by Bob Retelle)

This one takes place aboard a familiar starship. The crew has left for good reasons - but they forgot to take you, and now you are in deep trouble.

DEATH SHIP (by Rodger Olsen)

Our first and original ADVENTURE, this one takes place aboard a cruise ship - but it ain't the Love Boat.

VAMPIRE CASTLE (by Mike Bassman)

This is a contest between you and old Drac - and it's getting a little dark outside. $14.95 each.

OSI NEW-NEW-NEW OSI TINY COMPILER

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -, =, *, /, , <,> , Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.

TINY COMPILER — $19.95 on tape or disk OSI

SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for —

SUPERDISK II $29.95 (5¼") OSI

BARE BOARDS FOR OSI C1P

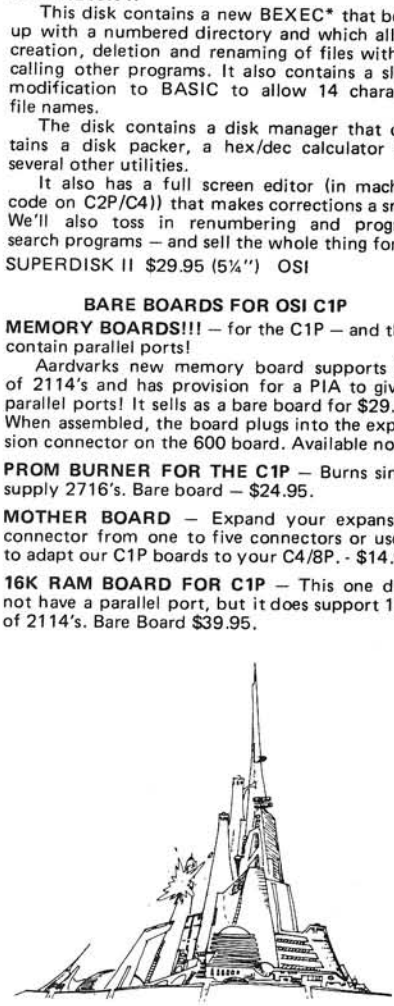
MEMORY BOARDS!!! — for the C1P — and they contain parallel ports!

Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available now!

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. - $14.95.

16K RAM BOARD FOR C1P — This one does not have a parallel port, but it does support 16K of 2114's. Bare Board $39.95.



WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabibilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (OS65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS — $39.95. Specify 5¼ or 8" disk.

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95

C1E/C2E similar to above but with extended machine code monitor. — $59.95 OSI

ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)

TIMETREK - A REAL TIME, REAL GRAPHICS STARTRECK. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! $14.95.

STARFIGHTER - This one man space war game pits you against spacecruisers, battlewagons, and one man fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Another real time goody. $9.95

BATTLEFLEET - This grown up version of Battleship is the toughest thinking game available on OSI or 80 computers. There is no luck involved as you seek out the computers hidden fleet. A topographical toughie. $9.95

QUEST - A NEW IDEA IN ADVENTURE GAMES! Different from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gather men and supplies by combbat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time.

16K COLOR-80 OR TRS-80 ONLY. $14.95

Please specify system on all orders

This is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSI systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.

OSI

AARDVARK TECHNICAL SERVICES, LTD.
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

COLOR-80

Machine Language to DATA Statement Generator

A convenient machine language program is presented to convert machine language routines to BASIC DATA statements. It can be applied to all OSI BASIC-in-ROM machines.

Yasuo Morishita
405 Lively Blvd.
Elk Grove Village, Illinois 60007

I find it tiresome to convert hex to decimal and to type everything in by hand (plus this may produce numerous typing errors). It would be convenient to have a short program to convert machine code routines to BASIC DATA statements. So, I wrote the following short program to do the work for me.

The command format is: ?USR(S)(E)(L)(I) and "Carriage Return," where:

- S is for the start address of the memory block
- E is for the end address of the memory block + 1, which you want to convert to DATA statements
- L is for the start of new line number for DATA statements
- I is for the increment of its new line number
- ? is short for the "PRINT" command in OSI BASIC; it can be "PRINT" or "Z=" — I have selected "?"

S, E, L and I should be input in decimal value.

To use this utility, you must:

1. RUN this program once after the BASIC COLD START, and it will set up USR(X) pointers and end-of-free-memory pointers.
2. You can LOAD or type in your own programs, if necessary, then type in ?USR(S)(E)(L)(I) to generate the required DATA statements.

Listing 1

```
10 REM DATA STATEMENT GENERATOR
20 REM
30 REM   VER. 3.0 ( SEP.1, 1981)
40 REM
50 REM    BY   YASUO MORISHITA
60 REM
70 REM FORMAT ?USR( START)(END)(NEW LINE #)(INC.)
80 REM
90 RESTORE:M=7858: REM USR(X) Start address=$1EB2
100 H=INT(M/256):L=M-256*H
110 POKE11,L:POKE12,H:POKE133,L:POKE134,H
120 N=163:FOR X=M TO M+N-1:READ J:POKE X,J:NEXT
130 A=41629:M=M+N:N=124:GOSUB160
140 A=41756:M=M+N:N=47 :GOSUB160
150 POKE M+15,96:NEW: REM Write "RTS"
160 FOR X=0 TO N-1:J=PEEK(A+X):POKE M+X,J:NEXT:RETURN
170 REM
180 REM Machine Language Data (163 Bytes)
190 REM
200 DATA 162,0,134,253,32,58,31,32,55,31,224,8,208,249,162
210 DATA 6,134,93,169,131,133,19,32,7,31,32,20,31,224,53,176
220 DATA 12,169,44,32,48,31,32,1,31,144,239,198,93,32,72,31
230 DATA 32,244,30,144,218,104,104,169,162,72,169,115,72,32
240 DATA 149,179,76,119,164,24,165,230,101,228,133,228,165
250 DATA 231,101,229,133,229,230,224,208,2,230,225,164,224
260 DATA 165,225,72,196,226,229,227,104,76,25,180,32,33,180
270 DATA 32,110,185,32,174,176,32,182,178,133,253,160,1,198
280 DATA 253,240,14,177,113,32,48,31,200,208,244,166,93,149
290 DATA 14,230,93,96,32,173,170,32,8,180,166,253,148,224
300 DATA 232,149,224,232,134,253,96,166,93,160,0,148,14,164
310 DATA 228,165,229,32,25,180
```

Listing 2

```
                   ; DATA STATEMENT GENERATOR
                   ;
                   ;  VER.3.0 ( SEP.1, 1981)
                   ;  BY  YASUO MORISHITA
                   ;
                    *=$1EB2
                   ;
                    DTABF = $E0
                   ;
1EB2 A2 00         DSG0    LDX #0       ;Get data from line
1EB4 86 FD                 STX $FD
1EB6 20 3A 1F              JSR GETDTA
1EB9 20 37 1F LDSG00 JSR GETDTB
1EBC E0 08                 CPX #8       ;Expects 4 data
1EBE D0 F9                 BNE LDSG00
1EC0 A2 06         LDSG02  LDX #6       ;Set input buffer ptr.
1EC2 86 5D                 STX $5D
1EC4 A9 83                 LDA #$83     ;"DATA" token
1EC6 85 13                 STA $13
1EC8 20 07 1F              JSR ENDDTA ;Get data from memory
1ECB 20 14 1F LDSG01 JSR FIXASC ;Write dec. data in buffer
1ECE E0 35                 CPX #$35     ;Line length limit
```

(Continued)

This program can even generate DATA statements with line numbers smaller than those of the existing BASIC program. It will insert the new line without any problem. If the new line number is same as the existing one, it will replace the old one with the new one.

Example: To convert memory block $0000 - $0010 with the starting line number as 10000 and an increment of 10, the input command will be:

```
?USR(0)(17)(10000)(10)
```

The result will be:

```
10000 DATA76,116,162,76,195,
      168,5,174,193,175,76,
      178,30,0,0
10010 DATA72,56
```

Even if you had any program prior to the execution of the above command, with line number such as 10, 20000, etc., the above two lines will be inserted correctly.

I have tried to convert 4096 bytes of memory into DATA statements with my C4P (BASIC-in-ROM) running at 2MHz system clock. It took about 20 seconds, produced 262 lines of DATA statements, and occupied about 14K bytes of memory. (Of course, I had to relocate the program to the safe location to do this experiment!)

Please note that this program uses quite a few subroutines out of BASIC ROM Version 1.0 Rev 3.2 of Ohio Scientific. It should work with ROM versions of the C4P, C1P, C2-4P and Superboard. It uses nine page 0 registers such as $E0 - $E7 and $FD. If you want to relocate the program, change the following subroutine addresses accordingly.

NAME of subroutines:
NXTLNO, NXTDTA, ENDDTA, FIXASC, WRTLBF, GETDTB, GETDTA, INSERT

It is also necessary to change line number 90 in the BASIC program to set up M as a new USR(X) start address in decimal.

If you are using a disk-based computer, or another manufacturer's computer, you will have to find out which register and subroutine will be equivalent to that used in this program.

Listing 2 *(Continued)*

```
1EDØ BØ ØC              BCS NDSGØ1
1ED2 A9 2C              LDA #$2C     ;","
1ED4 2Ø 3Ø 1F           JSR WRTLBF
1ED7 2Ø Ø1 1F           JSR NXTDTA   ;Set data ptr for next
1EDA 9Ø EF              BCC LDSGØ1
1EDC C6 5D              DEC $5D
1EDE 2Ø 48 1F NDSGØ1    JSR INSERT   ;Insert new line into text
1EE1 2Ø F4 1E           JSR NXTLNO   ;Update new line # & test
1EE4                                 ; if all data are done.
1EE4 9Ø DA              BCC LDSGØ2   ;Not yet all done.
1EE6 68                 PLA          ;Prepare to exit to BASIC
1EE7 68                 PLA
1EE8 A9 A2              LDA #$A2     ;Trick to JMP back to
1EEA 48                 PHA          ; BASIC warm start ($A274)
1EEB A9 73              LDA #$73
1EED 48                 PHA
1EEE 2Ø 95 B3           JSR $B395    ;Reset flag to numeric
1EF1 4C 77 A4           JMP $A477    ;Reset ptrs & exit to BASIC
1EF4              ;
1EF4 18           NXTLNO CLC         ;UPDATE NEW LINE #
1EF5 A5 E6              LDA DTABF+6  ;L. increment
1EF7 65 E4              ADC DTABF+4  ;L. current line #
1EF9 85 E4              STA DTABF+4  ;L. next line #
1EFB A5 E7              LDA DTABF+7  ;H. increment
1EFD 65 E5              ADC DTABF+5  ;H. current line #
1EFF 85 E5              STA DTABF+5  ;H. next line #
1FØ1 E6 EØ        NXTDTA INC DTABF   ;UPDATE DATA PTR.
1FØ3 DØ Ø2              BNE ENDDTA
1FØ5 E6 E1              INC DTABF+1
1FØ7 A4 EØ        ENDDTA LDY DTABF   ;Get current data ptr
1FØ9 A5 E1              LDA DTABF+1
1FØB 48                 PHA
1FØC C4 E2              CPY DTABF+2
1FØE E5 E3              SBC DTABF+3
1F1Ø 68                 PLA
1F11 4C 19 B4           JMP $B419    ;Set $11,12 with data ptr.
1F14                                 ;WRITE DEC. DATA IN BUFFER
1F14 2Ø 21 B4 FIXASC    JSR $B421    ;Get data from memory &
1F17                                 ; convert it to floating.
1F17 2Ø 6E B9           JSR $B96E    ;Floating-->ASCII string
1F1A 2Ø AE BØ           JSR $BØAE    ;Scan, set up string.
1F1D 2Ø B6 B2           JSR $B2B6    ;Discard unwanted string
1F2Ø 85 FD              STA $FD      ;ASCII string length
1F22 AØ Ø1              LDY #1
1F24 C6 FD        LFXASC DEC $FD
1F26 FØ ØE              BEQ RTNWLB   ;All done!
1F28 B1 71              LDA ($71),Y  ;Write dec.ASCII string
1F2A 2Ø 3Ø 1F           JSR WRTLBF   ; into input line buffer
1F2D C8                 INY
1F2E DØ F4              BNE LFXASC   ;=JMP
1F3Ø                                 ;WRITE CHR. IN INPUT BUF.
1F3Ø A6 5D        WRTLBF LDX $5D     ;Chr. ptr
1F32 95 ØE              STA $ØE,X
1F34 E6 5D              INC $5D
1F36 6Ø           RTNWLB RTS
1F37                                 ;GET DATA FROM LINE
1F37 2Ø AD AA GETDTB    JSR $AAAD    ;Evaluate expression
1F3A 2Ø Ø8 B4 GETDTA    JSR $B4Ø8    ;Get value in (Y/A)=(L/H)
1F3D A6 FD              LDX $FD      ;data counter
1F3F 94 EØ              STY DTABF,X  ;Save L. data
1F41 E8                 INX
1F42 95 EØ              STA DTABF,X  ;Save H. data
1F44 E8                 INX
1F45 86 FD              STX $FD
1F47 6Ø                 RTS
1F48                                 ;INSERT NEW LINE IN TEXT
1F48 A6 5D        INSERT LDX $5D     ;Chr. ptr.
1F4A AØ ØØ              LDY #Ø       ;Terminate input line
1F4C 94 ØE              STY $ØE,X    ; buffer
1F4E A4 E4              LDY DTABF+4  ;Get L. new line #
1F5Ø A5 E5              LDA DTABF+5  ;Get H. new line #
1F52 2Ø 19 B4           JSR $B419    ;Set new line # in $11,12
1F55              ;
1F55              ;Follows Insert program copied from BASIC ROM
                  ;
                  End
```

MICRO

NOW TWO LOCATIONS

SAVE TIME • SAVE SHIPPING

ATARI

Computers for people.™

A Warner Communications Company

800™ $679

410 Recorder	$76.00
810 Disc Drive	$449.00
822 Printer	$229.00
825 Printer	$629.00
830 Modem	$159.00
820 Printer	$269.00
850 Interface	$169.00
New DOS 2 System	$21.00
CX30 Paddle	$18.00
CX40 Joy Stick	$18.00
CX853 16K RAM	$89.00
Microtek 16K RAM	$75.00
Microtek 32K RAM	$159.00
Ramcram (128K)	$539.00
One year extended warranty	**$50.00**

ATARI 400

16K.... $329
32K.... $478
48K.... $555

ATARI SOFTWARE

CX404 Word Processor	$119.00
CX405 PILOT (educational)	$105.00
CX413 Microsoft Basic	$68.00
CX4101 Invitation To Programing I	$17.00
CX4102 Kingdom	$13.00
CX4103 Statistics	$17.00
CX4104 Mialing List	$17.00
CX4105 Blackjack	$13.00
CX4106 Invitation to Programing 2	$20.00
CX4107 Biorythm	$13.00
CX4108 Hangman	$13.00
CX4109 Graph It	$17.00
CX4110 Touch Typing	$20.00
CX4112 States & Capitals	$13.00
CX4114 European Countries & Capitals	$13.00
CX4115 Mortgage & Loan Analysis	$13.00
CX4116 Personal Fitness Program	$59.00
CX4117 Invitation To Programing 3	$20.00
CX4118-20 Conversational Languages (ea.)	$45.00
CX4121 Energy Czar	$13.00
CXL4001 Educational Master	$21.00
CX6001-17 Talk & Teach Series (ea.)	$23.00
CX8106 Bond Analysis	$20.00
CX8107 Stock Analysis	$20.00
CX8101 Stock Charting	$20.00
CXL4002 Basic Computing Language	$46.00
CXL4003 Assembler Editor	$46.00
CXL4004 Basketball	$24.00
CXL4005 Video Easel	$24.00
CXL4006 Super Breakout	$30.00
CXL4007 Music Composer	$45.00
CXL4008 SPACE INVADERS	$32.00
CXL4009 Chess	$30.00
CXL4010 3-D Tic-Tac-Toe	$24.00
CLS4011 STAR RAIDERS	$39.00
CXL4012 MISSLE COMMAND	$32.00
CXL4013 ASTEROIDS	$32.00
CXL4015 TeleLink	$20.00
Visicalc	$149.00
Letter Perfect (Word Processor)	$109.00
Source	$89.00
CX481 The Entertainer	$75.00
CX482 The Educator	$130.00
CX483 The Programmer	$54.00
CX 484 The Communicator	$329.00

Texas Instruments

TI-99/4A $359

PHP 1600 Telephone Coupler	$169.00
PHP 1700 RS-232 Accessories Interface	$169.00
PHP 1800 Disk Drive Controller	$239.00
PHP 1850 Disk Memory Drive	$389.00
PHP 2200 Memory Expansion (32K RAM)	$319.00
PHA 2100 R.F. Modulator	$43.00
PHP 1100 Wired Remote Controllers(Pair)	$31.00
32K Expansion	$329.00
PHP Printer Solid State	$319.00

CALL FOR SOFTWARE SELECTION AND PRICES

XEROX 820

Xerox 820	
System I 5¼"	$2450.00
System II 8"	$2950.00
CPM 5¼"	$169.00
Word Processing	$429.00
Super Calc	$269.00

CALL FOR MORE INFORMATION

PRINTERS

Centronics 739-1	$649.00
Diablo 630 Special	$1799.00
Epson	
MX70	$359.00
MX80	$469.00
MX80FT	Call
MX100	Call
NEC	
8023	$639.00
7730	Call
7720	Call
7710	Call
Okidata	
82A	$499.00
83A	$769.00
84	$1129.00
Citoh Starwriter	
25 CPS-P	$1329.00
45 CPS-P	$1699.00
Paper Tiger	
445G	$699.00
460G	$899.00
560G	$1129.00
Talley	
8024-7	$1399.00
8024-L	$1629.00

Terminals

Televideo	
910	$579.00
912C	$699.00
920C	$749.00
950	$939.00
Call for computers	
Zenith Z19	$749.00
Adds	$549.00

Modems

Novation Auto	$239.00
D Cat	$169.00
Cat	$159.00
Hayes	
Smart	$239.00

HOW TO ORDER: Phone orders invited or send check or money order and receive free shipping in the continental United States. PA and NEV. add sales tax.

computer mail order west

800-648-3351

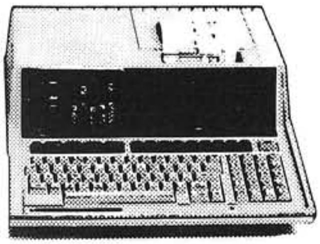
IN NEVADA, CALL (702) 588-5654

P.O. BOX 6689, STATE LINE, NEVADA 89449

TO SAVE YOU MORE!

COSTS • SAVE SALES TAX

HEWLETT PACKARD



HP•85 $1999

80 Column Printer	$799.00
NEW! HP•125	$2999.00
HP•83	$1699.00
HP•85 16K Memory Module	$249.00
5¼" Dual Master Disc Drive	$2129.00
Graphics Plotter (7225B)	$2079.00

NOW IN STOCK!

The new HP41CV Calculator

$259

41 C	$189.00
11 C	$119.00
12 C	$129.00
34 C	$117.00
38 C	$119.00
HP•41 Printer	$340.00
Card Reader	$164.00
Optical Wand	$99.00

HPIL CALCULATOR PERIPHERALS

IL Modual	$104.00
Digital Cassette	$449.00
Printer/Plotter	$419.00

CALL FOR SOFTWARE INFORMATION

Monitors

Amdex 12" B&W	$149.00
12" Green	$169.00
13" Color	$349.00
Sanyo 12" B&W	$259.00
12" Green	$269.00
13" Color	$449.00
TI 10" Color	$349.00

Electronics

Pioneer Lazer Disk	$599.00
BSR X-10 Systems	
PK 500	$84.00
LM 501	$16.00
AM611	$17.00
AM286	$17.00

commodore

CBM 8032

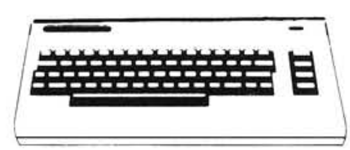
$1039

4032	$969.00
4016	$769.00
8096	$1569.00
Super Pet	$1599.00
2031	$529.00
8050	$1299.00
4022	$599.00
4040	$949.00
8300 (Letter Quality)	$1799.00
8023	$769.00
Pet to IEEE Cable	$37.00
IEEE to IEEE Cable	$46.00
Tractor Feed for 8300	$240.00
8010 Modem	$229.00

SOFTWARE

WordPro3 Plus	$199.00
WordPro4 Plus	$299.00
Commodore Tax Package	$399.00
Visicalc	$149.00
BPI General Ledger	$329.00
OZZ Information System	$289.00
Dow Jones Portfolio	$129.00
Pascal	$239.00
Legal Time Accounting	$449.00
Word Craft 80	$289.00
Power	$79.00
Socket-2-Me	$20.00
Jinsam	$Call
MAGIS	$ Call
The Manager	$209.00
Softrom	$129.00
Real Estate Package	$799.00
BPI Inventory Control	$319.00
BPI Job Costing	$319.00
BPI Payroll	$319.00

VIC 20 $259



COMPLETE

Vic 6 Pack Program	$44.00
VIC1530 Commodore Datassette	$69.00
VIC1540 Disk Drive	$499.00
VIC1515 VIC Graphic Printer	$339.00
VIC1210 3K Memory Expander	$32.00
VIC1110 8K Memory Expander	$53.00
VIC1011 RS232C Terminal Interface	$43.00
VIC1112 VIC IEEE-488 Interface	$86.00
VIC1211 VIC 20 Super Expander	$53.00
VIC1212 Programmers Aid Cartridge	$45.00
VIC1213 VICMON Machine Language Monitor	$45.00
VIC1901 VIC AVENGERS	$23.00
VIC1904 SUPERSLOT	$23.00
VIC1906 SUPER ALIEN	$19.00
VIC1907 SUPER LANDER	$23.00
VIC1908 DRAW POKER	$23.00
VIC1909 MIDNIGHT DRIVE	$23.00
VT106A Recreation Pack A	$44.00
VT107A Home Calculation Pack A	$44.00
VT164 Programmable Character/Gramegraphics	$12.00
VT232 VICTerm I Terminal Emulator	$9.00

New VIC Software

Household Finance	$27.00
VIC Games	$19.00
VIC Home Inventory	$13.00
VIC Rec/Ed II	$13.00
I SAM	$79.00
General Ledger	$229.00
Accounts Receivable	$229.00
Inventory	$229.00
Terminal	$13.00
Un Word	$13.00
Grafix Menagerie	$11.00
VIC PICS	$15.00
Ticker Tape	$13.00
Banner Headliner	$13.00
RS 232	$39.00

Above are cash prices, add 3% for Master Card and Visa purchases.

computer mail order east

800-233-8950

IN PA. CALL (717) 327-9575

501 E. THIRD ST., WILLIAMSPORT, PA 17701

Autonumber Plus for Cursor Control

These enhancements to the Cursor Control program (MICRO 36:75) include automatic line numbering, PRINT AT, and better BASIC access to such functions as window setting.

Kerry Lourash
1220 North Dennis
Decatur, Illinois 62522

This short, machine language utility frees C1P owners from the drudgery of typing line numbers and doubles as a fast line deleter.

When the Autonumber (AN) program (listing 1) is patched into Cursor Control, a number can be called up by hitting the LINE FEED key. A number will appear on the screen, indented one space and followed by a space, just as line numbers appear when they are LISTed. Only the number is stored in the buffer; this lets you use the limited buffer length to the fullest. By hitting the LINE FEED and RETURN keys alternately, you can delete lines quickly.

The counter for the Autonumber is located in $F1, F2 (decimal 241 and 242). It can be set directly with POKEs, or zeroed by doing a warm start. The counter can also be zeroed by POKEing $206 (decimal 518) to zero.

Autonumber is patched into the Cursor Control by setting CC's PATCH jump to the starting address of Autonumber:

Change $1E10 ($12) to $22
$1E11 ($1E) to $02

The line increment can be altered by changing location $024C (decimal 588).

The AN uses a BASIC-in-ROM subroutine whose normal function is printing line numbers for the LIST routine and EEROR IN XXXX messages. This subroutine converts the contents of the A and X registers to an ASCII string stored in $0100-010C. Next, it prints the string on the screen. The space after the line number is printed by another BASIC-in-ROM routine.

The AN program can be relocated, but $1E10 and $1E11 must point to the new starting address. If you've relocated the Cursor Control program, adjust AN's JMP $1E12 accordingly.

Because of memory space limitations, I was not able to make the Cursor Control as modular as I would have liked. Several useful routines are impossible to access directly from BASIC. Also, I noticed that I seldom used the window feature because the windows are hard to set. The following routines (listing 2) should correct these weaknesses.

First, I designed the USR GO routine to make machine language subroutines easier to access. This routine eliminates the need to POKE different USR vectors when multiple machine language routines are called in a BASIC program. The vector ($11-12) only needs to be set once, to the start of the USR GO routine. When you call a machine language subroutine, type X-USR (DDDDD). The D's represent the decimal address of the subroutine. You can use a number, variable, or even an expression inside the parentheses. For example, (2*256+6*16+4) would be accepted. To set USR GO, POKE 11,100:POKE12,2.

USR GO also allows five special subroutines to be called with a single digit (1-5). USR GO checks the high byte of the calling address in the USR parentheses before going to that address. If the high byte is zero (address less than 255), USR GO selects one of the five routines. If the number is not 1-5, a "function error" message is printed. With a little examination of the USR GO logic you can add over 200 of your own often-used subroutines. Here's a hint: $B408 returns with the low byte of the address in the Y register.

Now that multiple machine language routines are easy to access, it's possible to tap three useful Cursor Control subroutines:

ESC - Switch windows (1)
RUB - Erase current window (2)
HOM - Home cursor (3)

There is also a PRIN AT function that moves the cursor location to any address in screen memory:

PRINAT - Print at (4)

The command format is X= USR(4) offset. The offset should be 1-1000 and can be expressed as a number, variable, or formula. The offset is added to $D000 (upper left corner of the screen) and the cursor is moved to that location. A handy way to set cursor location is X= USR(4)A*32+B.

To make window setting easier, I developed:

WINSET - Set window boundaries (5)

The command format is X = USR(5)top boundary, bottom boundary. The boundaries are expressed as line numbers: 1 = top to 32 = bottom.

See figure 2 in the Cursor Control article for a map of the window lines. A typical command would be: X = USR(5)24,30. This command would set the alternate window to the bottom quarter of the screen. To use the window, call the ESC routine: X = USR(1).

CLR Subroutine

Notice that PRINAT uses one variable to the right of the USR parentheses, while WINSET uses two. CLR allows the use of a command form: X = USR(A),B,C for both routines. CLR finds the end of the statement, either colon or null, and sets the parser pointer ($C3,C4) past the end of the line. Otherwise, BASIC would print an error message.

After trying out the Autonumber Plus, you may wish to relocate it to leave the block of RAM at $0222 free. Cursor control could be moved down one or two pages and the AN relocated to the top of memory. Cursor Control will protect them from being overwritten. Warmstart vector $0001 and $0002 would have to be adjusted, of course.

Once again, I invite persons interested in CC or BASIC-in-ROM to drop me a line. I would particularly like to compliment the OSI Users Group-Northwest on their ROM BASIC memory map. Also, I thank A. Penaloza for his article in the August issue of *PEEK(65)* that made it possible to adapt the Cursor Control to C2P/C4P computers.

Listing 1

```
                    ;AUTONUMBER FOR CC
                    COUNTL=$F1
                    COUNTH=$F2
                      FLAG=$206
                         *=$222
                   ;
0222 C90A           AUTONM CMP #$A        ;LINE FEED KEY?
0224 D03B                  BNE  QUIT      ;NO, BACK TO CC
0226 A900                  LDA #0
0228 AE0602                LDX  FLAG      ;FLAG=0 ?
022B D004                  BNE  ZERO      ;NO, DON'T
022D 85F1                  STA  COUNTL    ;RESET COUNTER
022F 85F2                  STA  COUNTH
0231 8D0602         ZERO   STA  FLAG
0234 A6F1                  LDX  COUNTL
0236 A5F2                  LDA  COUNTH
0238 205EB9                JSR  $B95E     ;PRINT LINE #
023B 20E0A8                JSR  $A8E0     ;PRINT SPACE
023E A2FF                  LDX #$FF
0240 E8             LOOP   INX
0241 BD0101                LDA  $101,X    ;GET DIGIT
0244 F004                  BEQ  INCRMT    ;BRANCH IF NULL
0246 9513                  STA  $13,X     ;DIGIT IN BUFFER
0248 D0F6                  BNE  LOOP
024A                ;
024A 18             INCRMT CLC            ;INCREMENT COUNTER
024B A905                  LDA #5
024D 65F1                  ADC  COUNTL
024F 85F1                  STA  COUNTL
0251 9002                  BCC  DONE
0253 E6F2                  INC  COUNTH
0255 8E0602         DONE   STX  FLAG      ;SET FLAG
0258 68                    PLA           ;PULL BUFFER INDEX (X)
0259 A8                    TAY           ;FROM STACK AND REPLACE
025A 68                    PLA           ;WITH DIGIT COUNT
025B 8A                    TXA
025C 48                    PHA
025D 98                    TYA
025E 48                    PHA
025F A901                  LDA #1         ;NON-PRINTING CHAR
0261 4C121E         QUIT   JMP  $1E12     ;BACK TO CC
```

Listing 2

```
                 ;BASIC ACCESS TO
                 ;CURSOR CONTROL
                 CURSOR=$E0
                 ALTWIN=$E6
                  PATCH=$1E0F
                 ESCAPE=$1E5C
                   HOME=$1E72
                 RUBOUT=$1E80
                 PCURSR=$1F14
                  PRINT=$1F1F
                     *=$0264
0264 2008B4      USRGO  JSR  $B408     ;CONVERT TO 2-BYTE No.
0267 C900               CMP  #0        ;IS HI BYTE=0?
0269 F010               BEQ  ESC       ;YES, TO CC SUBS
026B 6C1100             JMP  ($0011)   ;JUMP TO ADDRESS
026E             ;
026E 201AA7      CLR    JSR  $A71A     ;FIND END OF LINE
0271 C8                 INY            ;PLUS 1
0272 98                 TYA
0273 18                 CLC            ;UPDATE PARSER POINTER
0274 65C3               ADC  $C3
0276 9002               BCC  CL1
0278 E6C4               INC  $C4
027A 60          CL1    RTS
027B             ;
027B 88          ESC    DEY            ;SWITCH WINDOWS
027C D005               BNE  RUB
027E 48                 PHA
027F 48                 PHA
0280 4C601E             JMP  ESCAPE+4
0283             ;
0283 88          RUB    DEY            ;CLEAR WINDOW
0284 D005               BNE  HOM
0286 48                 PHA
0287 48                 PHA
0288 4C841E             JMP  RUBOUT+4
028B             ;
028B 88          HOM    DEY            ;HOME CURSOR
028C D005               BNE  PRINAT
028E 48                 PHA
028F 48                 PHA
0290 4C6F1E             JMP  HOME-3
0293             ;
0293 88          PRINAT DEY            ;PRINT AT
0294 D016               BNE  WINSET
0296 201F1F             JSR  PRINT     ;ERASE CURSOR
0299 20C1AA             JSR  $AAC1     ;GET OFFSET
029C 2008B4             JSR  $B408     ;CONVERT TO 2-BYTE No.
029F 84E0               STY  CURSOR    ;ADD OFFSET TO $D000
02A1 18                 CLC
02A2 69D0               ADC  #$D0
02A4 85E1               STA  CURSOR+1
02A6 20141F             JSR  PCURSR    ;PRINT CURSOR
02A9 4C6E02             JMP  CLR       ;GOTO END OF LINE
02AC             ;
02AC 88          WINSET DEY            ;SET ALT. WINDOW
02AD D032               BNE  ERR
02AF 20C302             JSR  WINGET+3  ;GET START OF WINDOW
02B2 20D502             JSR  STOR      ;STORE IT
02B5 20C002             JSR  WINGET    ;GET END OF WINDOW
02B8 A202               LDX  #2
02BA 20D502             JSR  STOR      ;STORE IT
02BD 4C6E02             JMP  CLR       ;TO END OF LINE
02C0             ;
02C0 2001AC      WINGET JSR  $AC01     ;FIND COMMA ELSE ERROR
02C3 20C1AA             JSR  $AAC1     ;GET VALUE
02C6 2005AE             JSR  $AE05     ;CONVERT TO 2-BYTE #
02C9 C6AF               DEC  $AF       ;MINUS 1
02CB A205               LDX  #5        ;#6 FOR 2K CONVERSIONS
02CD 06AF        W1     ASL  $AF       ;MULTIPLY BY 32
02CF 26AE               ROL  $AE
02D1 CA                 DEX
02D2 D0F9               BNE  W1
02D4 60                 RTS
02D5             ;
02D5 A5AF        STOR   LDA  $AF       ;STORE WINDOW VALUES
02D7 95E6               STA  ALTWIN,X
02D9 18                 CLC
02DA A9D0               LDA  #$D0
02DC 65AE               ADC  $AE
02DE 95E7               STA  ALTWIN+1,X
02E0 60                 RTS
02E1 4C88AE      ERR    JMP  $AE88     ;FUNCTION CALL ERR
```

MICRO

PROGRAMMING THE APPLE II JUST GOT EASIER!

ANIX V1.0

A UNIX-like DOS 3.3 Compatible Operating System for the Apple II

ANIX 1.0 is a "Tiny-UNIX™" that is completely file structure upwards compatible with DOS 3.3. ANIX incorporates a healthy subset of UNIX'S™ capabilities and a superset of CP/M's™ capabilities. In addition to the ANIX operating system, Lazer provides over 30 utility programs on the ANIX disk. Included are: ADU (ANIX disk utility), DUPDSK, EXPUNGE, DREN, STAT, TYPE, PUTDOS, FREE2, UNDEL, MAKE, SIZE, MON, NOMON, WC (word count), LC (line count), PIE.PATCH, PRINT, BUILD, FIND, BTOT (binary to text conversion), INIT, MAKEAUTO, PAG3, ED, COPY, SETCLK, SETPRTR, SETLOWER, SETINVMODE, TIME, and more! Many of the programs found on the ANIX disk are selling alone for between $30 and $60. Also included are the source listings for several of the utilties. For advanced machine language programmers, the source listing of the entire operating system is available separately. ANIX supports powerful screen editing commands and will automatically time and date stamp files saved to disk (if a clock card is installed). For a complete description of ANIX's capabilities, call or write Lazer.

ANIX is the start of a complete line of system software tools available from Lazer MicroSystems, Inc. All new languages and applications programs available from Lazer will run under the ANIX operating system. Lazer Pascal is available now. Other languages and systems are in the works. Productive programmers are already using ANIX, are you?

Lazer Pascal

The Elegance of Pascal-- The Power of "C"

Lazer Pascal is a unique systems programming language for the Apple II. It combines features found in Pascal and is extended to include several features found in the "C" programming language. The Lazer Pascal compiler is very fast (1500-2000 lines/minute) making the system very easy to use. No longer will the systems programmer or game programmer be forced to use assembly language, Lazer Pascal is here!

Lazer Pascal supports BYTE, CHAR, BOOLEAN, INTEGER, LONG, pointer, string, array, static, and dynamic data types. Lazer Pascal was created to replace 6502 machine language as the choice of systems and game programmers. Included with Lazer Pascal are several sample programs demonstrating the use of Lazer Pascal including: ANIX.P, TSTPARMS.P, LINECOUNT.P, WORDCOUNT.P, CHARCNT.P, EXPAND.P, COMPRESS.P, CRYPT.P, and TRANSLIT.P. Better yet, the source listings for the compiler, P-code interpreter, and other utilities are included.

D/ASM

A High-Powered 6502 Disassembler for the Apple II

DISASM/65 produces a 6502 assembly language source listing from machine code and a set of input commands. Only DISASM/65 supports all the commonly used data types found in machine language programs. We used DISASM/65 to disassemble DOS 3.3 for our popular DOSOURCE 3.3 product-- that should describe DISASM/65's power! DISASM/65 is provided with our popular LISA V2.5 assembler. Several users, however, have reported considerable success using DISASM/65 with the Toolkit assembler, the SC Assembler, TED, and others; so we are offering DISASM/65 separately for these users.

p-SOURCE

The Internals of the Apple P-code Interpreter Explained

for Apple Pascal users

p-SOURCE is a technical manual that describes the internal operation of the Apple Pascal P-code interpreter. Included are descriptions of programming techniques used within the interpreter, hints on how to speed up the interpreter, add your own routines to it, and incorporate hardware floating point. p-SOURCE is absolutely essential to the Pascal programmer.

ANIX, Lazer Pascal, p-SOURCE and DISASM/65 were all written by Randy Hyde, the author of "USING 6502 ASSEMBLY LANGUAGE", LISA, SPEED/ASM, DOSOURCE 3.3, and other fine software products. Additional information on Lazer's software products can be obtained by calling or writing Lazer MicroSystems, Inc.

Lazer Software Products:

ANIX:	$49.95
LAZER PASCAL:	$39.95
DISASM/65:	$29.95
p-SOURCE (book):	$19.95

Other Fine Lazer Products:

Lower Case +Plus II :	$29.95
Lower Case +Plus :	$64.95
Keyboard +Plus :	$99.95
Character Set +Plus :	$24.95

UNIX™ Bell Labs.
CP/M™ Digital Research, Inc.
APPLE II™ Apple Computer, Inc.
ANIX™ Lazer MicroSystems, Inc.

Lazer MICRO SYSTEMS INC.
1791-G Capital
Corona, CA 91720
(714)735-1041

MICRO

The Single Life

By Brad Rinehart

Two years ago, Hudson Digital Electronics Inc. (HDE) decided to add an improved BASIC interpreter to their expanding line of development software. I was able to witness the creation of this interpreter, HDE Disk BASIC. Here's the inside story.

HDE designed their BASIC primarily for demanding industrial users, such as General Electric. Therefore, a comprehensive, sophisticated package was a necessity. However, since many hobbyists, businessmen, and professionals use HDE products, the new BASIC also had to be easy-to-use.

HDE wisely chose Microsoft's popular BASIC as the foundation for its new Disk BASIC. In the beginning there were the standard reserved words: PRINT, INPUT, TAN, POKE, etc., and only a few, very limited, disk commands. Today there are 127 reserved words, three types of disk files, and several library functions. Quite a piece of software!

HDE Disk BASIC is compatible with the AIM, SYM, and KIM disk systems. Customized versions are available by special request. I'll discuss only the KIM-based version here (the other versions are similar).

I want to start by describing some of HDE Disk BASIC's more outstanding features. In the KIM-based version, HDE Disk BASIC resides in memory from \$2000 to \$5000 and also uses memory from \$E000 to \$ECFF. In a 56K system, this leaves approximately 36,600 bytes of user memory! Room for plenty of code here.

HDE Disk BASIC is very user-friendly. For example, it relieves the user of the burden of manipulating the machine through PEEKs and POKEs. Instead, HDE Disk BASIC provides such handy features as 'ERN' and 'ERL', reserved words that return the 'ERror Number' and 'ERror Line' when an error occurs. The programmer can call this feature *via* an 'ON ERROR GOTO' instruction.

The following example demonstrates the use of ERL and ERN to determine that an OUT OF DATA error occurred during the READing of the data statements into the string variables.

```
 10 ON ERROR GOTO 100
 20 READ A$, B$, C$
 30 DATA THESE, ARE, STRINGS
 40 END
100 IF ERN = 4 AND ERL =
     20 THEN RESTORE: RESUME
110 PRINT "ERROR NUMBER";
     ERN; "OCCURRED IN LINE";
     ERL:STOP
```

Note that the use of ERL allows you to clarify not only the type of error, but also in which line it occurred.

HDE Disk BASIC also provides you with a line editor similar to the one found in HDE's TED (TExt eDitor). This feature alone will save many hours of program development time on the screen.

The editor functions include:

APP allows you to append or add statements or comments to the end of one or more lines.

AUTO provides automatic line numbering. You may specify the line number to start with, as well as the line increment value (1-10).

COPY lets you copy one line to one or more new or existing lines. If the target line (the one being copied to) already exists, the entire line will be replaced with the source line.

DEL deletes from line to line. You may also specify DEL REM which will delete all the REM or remark statements from the program and leave the rest of the program untouched.

EDIT in HDE Disk BASIC is almost identical to the EDT statement in TED. By specifying the line to edit, (i.e. EDIT 200), you may insert, delete, or modify characters within the line.

FIND lets you find or locate reserved words or statements within the program.

MOVE is similar to COPY, except that the source line is removed from the text. For example, MOVE 100 200 will cause line 100 to be removed from the text and placed at line 200.

RENUM provides for renumbering of the program. It automatically adjusts all GOTO and GOSUB references to renumbered lines. You may specify the line to start with, the line number increment value, the number to assign to the first line, and the last line to renumber. This is an extremely useful function.

SET allows you to change a group of characters or words anywhere it occurs within the program. BASIC's SET command will display the line before making the change. You then have the ability to invoke the automatic change, or abort the change in individual lines or string occurrences.

You will appreciate these powerful edit functions. Instead of having to list programs or search for a particular statement, you can execute the FIND command and BASIC will display all lines containing the requested statement. And if you've ever spent hours removing the remark statements from the runtime version of a program, you'll especially appreciate the DEL REM function.

When comparing one machine with another, keep in mind that HDE's edit functions are an integral part of the interpreter, not an add-on package or one that must be 'hooked' or 'linked' into the interpreter by the user. When you buy the software, you get the editor!

Modular or single board systems allow you to interface more than one type of terminal to the system. This ability provides a great deal of flexibility for the system designer, but it also presents interesting problems to the software engineer. For example, many terminals recognize the escape (ESC) character as a 'lead in' character for control sequences (clear screen, position cursor, etc.). Still others may use control characters, such as 'control X' or 'control A' to invoke special functions, like self test. Another problem arises when a teletype is used as the terminal. Most teletypes only provide for a 72-character line, while CRT displays allow a minimum of 80 characters.

HDE Disk BASIC recognizes several characters which are used for specific functions. The escape (ESC) character is used to insert characters within a line when editing. Control X is used as a cancel character for several commands. To avoid conflicts, a personality module allows you to change the characters that are recognized for backspace, back-arrow, escape, control X, and control A. The personality module also allows you to define the line width and the character sequence transmitted for 'CLS' or clear screen.

In addition to interfacing BASIC to the system terminal, the personality module may be used to keep you out of trouble. For example, program development generally dictates that some means of stopping the program and/or looking at memory locations must exist. However, when the program is purchased by end users, they do not want the program to stop, and they do not care what is in which memory location. Therefore, we need a way to keep both sides of the industry happy. HDE's personality module satisfies this requirement. The normal functions, such as escape, control X, control A, etc., may be used while you develop the software. Then, after the package is complete, the personality module may be used to eliminate these functions. The process is simple: just 'tell' BASIC to recognize $FF for these functions. Since no terminal transmits this hex sequence, the functions are ignored.

If you have not yet ventured beyond the realms of the personal computer, you may need an explanation of the library function. Many years ago someone decided that there must be a way to invoke common routines from more than one program. Therefore, a method was devised to provide shared routines, similar to the way in which people share books from a public library. When you want it, you go get it. When you're through with it, someone else may use it. The only requirement is that the book or routine must be in the library when you need it. In the case of a disk-based library, the routine is read from the disk when needed, but it is never erased or removed.

This library allows you to expand the capabilities of the interpreter without expanding the size of the interpreter in memory. Currently, there are several useful routines that may be included in the library. I should mention that the user is required to pay a nominal fee for these routines as they are not part of the standard package. A sampling of the HDE Disk BASIC library routines follows:

DUP lets you duplicate a data disk. DUP copies the entire disk using the FREE area in BASIC as a large disk buffer.

JMP prints a cross-reference of all referenced line numbers and the lines that reference them. For example, in the statement 100 GOSUB 1000, line 1000 is the referenced line, 100 is the line that refers to it.

VAR provides a cross reference of all variables and the lines in which they are used.

MAP displays the current memory map defining the area used for program storage, array storage, simple variables, disk file buffers, strings, and the free (or FRE) area.

A unique feature of the library is the means by which routines are called from it. For example, to implement the 'VAR' routine, you only need to enter LIB "VAR". Hence, virtually any routine may be added to the library and invoked through the 'LIB' command. This means that user routines may be called directly from the library, saving an entry into the disk index! The library is currently limited to fifty sectors, which is about 6400 bytes. However, it may be expanded if necessary.

When you boot up your HDE disk system, control is passed to the File Oriented Disk System (FODS). From FODS, you enter the command 'BAS' and BASIC is loaded from the system disk (usually drive 1 or 0) and initialized. The following phrase appears on the screen:

MEMORY SIZE?

You have three options. If a carriage return only is entered, BASIC determines the amount of available memory. If you do not wish to allow BASIC to use all of memory, you enter the decimal value of the highest location to be allocated to BASIC, followed by a (RETURN). If you want to allocate memory above $CFFF for something other than BASIC, you enter 53247 (RETURN) to the MEMORY SIZE? question. If this is the first time BASIC is run, you may want to enter a 'P' followed by (RETURN). This invokes the personality module.

There is one more method for invoking BASIC. Even though little has been written about FODS, it does provide for an 'auto start' function. This function may be used to implement BASIC, or any other program from the system disk. First the command word, such as BAS, must be written into FODS. Then either the boot strap routine must be changed to jump four locations higher into FODS, or the first three locations in FODS must be changed to NOPs ($EA). I find it easier to load FODS, change the first three locations, and save it back to a blank disk. This disk is then used for all auto start functions.

The command word used to invoke the called program is written into FODS beginning 38 ($26) locations from the beginning. FODS will recognize the first three characters as a command. The command word must be terminated by a carriage return ($0D). When invoking BASIC, if the sequence 'BAS. $0D' is used, BASIC will then look for a program '@MENU' on the system drive and execute it.

This auto start function allows a user, as opposed to a programmer, the ability to use pre-packaged software.

Next month I will explain the reserved words and their uses. Those of you who are interested in bringing HDE Disk BASIC up on your machine may want to contact one of HDE's factory representatives about obtaining a copy of the package.

Please address all correspondence to the author at 1508 Stanton Street, York, Pennsylvania 17404.

MICRO

A feast of computing ideas.

If you work with a 6502/6809-based system, you're probably hungry for the facts and ideas that will help you understand the inner workings of your computer. You want to go beyond canned software—use your computer for more than games—learn the advanced programming techniques that enable you to get the most out of your 6502/6809 system.

MICRO, The 6502/6809 Journal, gives you page after page, month after month, of solid information to sink your teeth into. **MICRO** is the premier how-to magazine for serious users of the Apple, PET/CBM, OSI, Atari, AIM, SYM, KIM, and all 6809 based systems including the TRS-80 Color Computer. It's a resource journal internationally respected by professionals in business, industry, and education. Every issue of **MICRO** keeps you informed with up-to-the-minute data on new products and publications:

- **hardware catalog** with organized, concise description
- **software catalog** in an easy-to-use format
- **new publications** listed and annotated
- **reviews and evaluations** of significant products

And there's much more:

- **In-depth hardware tutorials** bring expert advice into your home or office.
- **Detailed discussions of programming languages** deepen and broaden your programming ability.
- **Complete program listings** enable you to increase your machine's capabilities.
- **Bibliography of 6502/6809 information** helps you to find pertinent articles in a timely manner.
- **Special monthly features** with in-depth treatment of one subject or system increase your knowledge of the field.
- **Balanced mix of machine-specific and general articles** for your everyday use as well as long-range reference needs.
- **Informative advertising** focused specifically on 6502/6809 machines keeps you abreast of latest developments.
- **Reader feedback** puts you in touch with other micro-computerists.

MICRO is the magazine you need to get the most from your own 6502/6809 system!

To order, send your check or international money order (payable to MICRO) and the order form at left, to:

Subscription Fulfillment
MICRO, Dept. MI
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Or, for your convenience, call our toll-free number:

1-800-227-1617, Ext. 564

(In California, 800-772-3545, Ext. 564)

and charge your subscription to your MasterCard or VISA. (All orders must be prepaid in U.S. dollars or charged to your MasterCard or VISA.)

SUBSCRIPTION RATES (U.S. dollars)
Yearly subscription (ISSN 027-9002) saves 20% off the single-issue price.

U.S. $24*

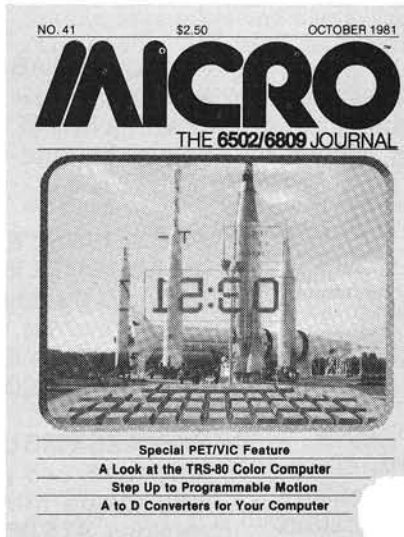
Canada $27

Europe $27 ($42 by air mail)

Mexico, Central America, Mideast, North and Central Africa $27 ($48 air)

South America, Far East, South Africa, Australasia $27 ($72 air)

*SPECIAL OFFER—U.S. ONLY:
Save even more—30% off single-issue price: 2 years, $42



You'll love every byte.

YES! I want to get more from my microcomputer. Please send me

___ year(s) of MICRO at $______/year.
(Outside U.S. and Canada, please indicate via ☐ surface or ☐ air mail.)

Name ______________________

Company ______________________

Street ______________________

City ________ State ________ Zip Code ________

☐ Check enclosed for $ ________
☐ Charge my credit card account
☐ VISA ☐ MasterCard

Signature ______________________

Card number ______________________

Expiration date ______________________

KIM Bouncy Keypad Cure

This 94-byte program eliminates the annoying keybounce and prolongs the life of your bouncy KIM-1 keypad. It can be used alone or as part of Jim Butterfield's BROWSE routine.

Jody Nelis
132 Autumn Drive
Trafford, Pennsylvania 15085

My KIM has a lot of miles on it. As it aged, the keybounce appeared and became progressively worse. After a frustrating attempt at repairing the keypad myself, I gave in and bought a replacement.

While this should have been the end of the story, it was not. Just as the original had developed a bounce, so did the replacement. The thought of buying a new keypad every year discouraged me, and I sought a solution.

A study of the KIM-1 schematic diagram eliminated the idea of putting capacitors across the contacts to debounce them. A multiplexed, scanned keypad that is shared with the LED display does not lend itself to a hardware fix.

The Cure

When I found that hardware wasn't practical, I turned to software. An analysis of the KIM-1 monitor listing led me to the source of the problem and suggested the cure.

Remember that KIM stands for Keyboard Input Monitor. Upon power up and after a little internal housekeeping, control is given to a routine in the ROM which does just that. It continually scans the keypad, patiently waiting for a key to be pushed. In its spare time, it keeps the LED display updated to reflect the latest input.

Unfortunately, this monitor routine does not have enough ability to differentiate between two separate and distinct keystrokes and one keystroke that bounced. The input routine is exceptionally fast. It will respond to your keystroke and update the display before the key you are pushing hits bottom.

Before this routine goes on to look for the next keystroke, it does a test to see if the key is still down. If it feels that it is still down, the routine just loops right there and continues to test until it decides that the key has been released. Then after a little additional internal housekeeping, it looks for the next keystroke.

A bouncing key is making and breaking contact. If it breaks contact long enough (only a few milliseconds), the test thinks that the original key has been released and a new one has been pushed. Thus, we can get a double (or triple) entry with just one keystroke. Some additional delay in the input routine is required.

Program Overview

Presented here is my mini-monitor called "KIM-1 Bounceless Data Entry." BOUNCELESS responds to all of the hex keys as well as the "AD," "DA," and "+" keys in the same fashion as the KIM monitor. It also adds the one feature missing in the KIM monitor. It includes an adjustable timed delay which is initiated when each keystroke is first detected. During this delay time, anything happening at the keypad is totally ignored. This effectively debounces even the worst of keypads.

BOUNCELESS is 94 bytes long. It fits nicely into the RAM area in page 17, but it is totally relocatable. You may put it in any other RAM location that suits you. Listing 1 is a commented disassembly listing of BOUNCELESS. Once you have played with it and set the delay to an interval that suits your finger and keypad condition, you can put it in EPROM if you wish.

While BOUNCELESS by itself is fine for short programs or data entry, anything over a few dozen bytes is best done using BROWSE. BROWSE is also a mini monitor utility that can be found in *The First Book of KIM*. Among other things, it can be used to aid in entering data or programs.

BROWSE has a debouncing scheme of sorts already written in it. While I have found it to be less susceptible to bounce than the KIM monitor, it is not adequate when the keypad deteriorates beyond a certain point.

Since I experienced keybounce using BROWSE, I modified it to include the time delay, thus making it bounceless also. I call my modified version DEBOUNCED BROWSE. It is 16 bytes longer than the original version, and remains totally relocatable.

Listing 2 is a hex dump of DEBOUNCED BROWSE produced by KIM. The commented disassembly of listing 3 shows only the modified portion.

Detailed Description

The KIM monitor is about 1K long. Fortunately not all of it is really the "Keyboard Input Monitor." In fact, most of it supports the TTY and cassette interfaces on the KIM-1 board.

BOUNCELESS duplicates only those portions of the monitor that actually handle the address pointers, hexadecimal input conversions, and display management during the inputting of programs or data. It utilizes monitor subroutines whenever possible.

Looking at listing 1, the coding in BOUNCELESS from $1780 to $17C7 and $17D8 to $17DD is a condensed version of the coding found in the KIM monitor from $1C4F to $1CE6. This portion, as you can determine from the comments, goes to the keyboard and gets the value of the key pushed. It then evaluates the key to see if it is a command or a hex key.

Once the decision is made, it goes to the KIM monitor to alter the address pointers, alter the data in the current address, or step to the next memory location.

The heart of BOUNCELESS is the 16-byte delay routine DEBNCE. It starts at $17C8 and goes to $17D7. Walking through the routine, you will see that the byte in $17C9 is stored in a location I labeled TIMER. This is the divide-by-1024 location for the KIM interval timer.

Once a value has been stored there, the timer starts counting down. It continues to count down, no matter what else the program may be doing. In this case though, there is nothing else for BOUNCELESS to attend to other than keep the display lit by repeated calls to SCAND while waiting for the timer to time out. It is during this waiting period that we get the debounce action. Since the program is occupied with watching the timer, it cannot get back to look at the keypad again rapidly enough to be fooled by a keybounce.

The end of the debounce routine has a forced branch. To keep the program fully relocatable, I couldn't use any JMP or JSR instructions, since they would have required absolute addresses in the arguments. So, a forced branch is achieved by clearing the carry flag at $17D5 and then using a Branch on Carry Clear at $17D6.

DEBOUNCED BROWSE is accomplished in the same manner as BOUNCELESS. The original BROWSE coding from $0100 to $018D remains the same except for the byte at 0124. My new routine, DEBNCE, follows from $018E to $019D. See listing 3. UP, a displaced routine from the original BROWSE ($018E to $0193) is now located from $019E to $01A3.

When relocating UP, the argument for the relative branch at $0123, which calls it, must be changed. Make $0124 $79 and it will point to the new location. Similarly, the argument for the forced branch at the end of UP had to be adjusted to get back to LP1.

Listing 1

```
                    ;KIM-1 BOUNCELESS DATA ENTRY
                    ;
                    ;BY JODY NELIS
                    ;
                    ;REGISTERS USED:
                    DELAY    EPZ $7F            ;MAY BE VARIED BY USER
                    POINTL   EPZ $FA            ;ADR POINTER LSB
                    POINTH   EPZ $FB            ;ADR POINTER MSB
                    TEMP     EPZ $FC            ;TEMPORARY STORAGE
                    MODE     EPZ $FF            ;AD OR DA MODE FLAG
                    ;
                    ;MONITOR EQUATES:
                    TIMER    EQU $1707          ;INTERVAL TIMER
                    INIT1    EQU $1E8C          ;INITIALIZE DISPLAY
                    SCAND    EQU $1F19          ;LIGHT LED DISPLAY
                    INCPT    EQU $1F63          ;INCREMENT MEMORY POINTERS
                    GETKEY   EQU $1F6A          ;INPUT KEY VALUE FROM KEYPAD
                    ;
                    ;INITIALIZE "ST" VECTOR TO ALLOW ENTERING PROGRAM
                    ; WITH THE "ST" KEY BY SETTING 17FA, 17FB EQUAL
                    ; TO THE (SAL),(SAH).
                    ;
                             ORG $1780
                    ;
1780 20 8C 1E       INITLZ   JSR INIT1          ;INITIALIZE DISPLAY
1783 20 19 1F       START    JSR SCAND
1786 F0 FB                   BEQ START          ;MAKE SURE NOT JUST
1788 20 19 1F                JSR SCAND          ; NOISE ON THE KEYS
178B F0 F6                   BEQ START
178D 20 6A 1F                JSR GETKEY         ;GET KEY VALUE
1790 C9 13                   CMP #$13           ;IF $13 OR GREATER IT IS
1792 10 EF                   BPL START          ; ILLEGAL OR NO KEY DOWN
1794 C9 10                   CMP #$10           ;IF 'AD' KEY DOWN, SET
1796 F0 28                   BEQ ADDRM          ; IN ADR MODE
1798 C9 11                   CMP #$11           ;IF 'DA' KEY DOWN, SET
179A F0 28                   BEQ DATAM          ; IN DATA MODE
179C C9 12                   CMP #$12           ;IF '+' KEY DOWN, GO TO
179E F0 38                   BEQ STEP           ; STEP ROUTINE
17A0                ;
17A0 0A             DATA     ASL
17A1 0A                      ASL                ;OTHERWISE MUST BE A
17A2 0A                      ASL                ; HEX KEY. SHIFT CHARACTER
17A3 0A                      ASL                ; INTO HIGH ORDER NIBBLE
17A4 85 FC                   STA TEMP           ; AND STORE IN TEMP.
17A6                ;
17A6 A2 04                   LDX #$04
17A8 A4 FF          DATA1    LDY MODE           ;TEST MODE FLAG (0=DATA,1=ADR)
17AA D0 0A                   BNE ADDR
17AC B1 FA                   LDA (POINTL),Y ;IF DATA, GET DATA,
17AE 06 FC                   ASL TEMP           ; SHIFT CHARACTER,
17B0 2A                      ROL                ;SHIFT DATA,
17B1 91 FA                   STA (POINTL),Y ; DATA TO DISPLAY,
17B3 18                      CLC                ; AND FORCE JUMP.
17B4 90 05                   BCC DATA2          ;ALWAYS!
17B6                ;
17B6 0A             ADDR     ASL                ;IF ADR, SHIFT CHAR,
17B7 26 FA                   ROL POINTL         ; SHIFT ADR LOW AND
17B9 26 FB                   ROL POINTH         ; SHIFT ADR HIGH
17BB                ;
17BB CA             DATA2    DEX
17BC D0 EA                   BNE DATA1          ;DO IT FOUR TIMES
17BE F0 08                   BEQ DEBNCE         ; THEN DEBOUNCE KEY
17C0                ;
17C0 A9 01          ADDRM    LDA #$01
17C2 D0 02                   BNE DATAM1         ;SET MODE FLAG TO ADDR
17C4                ;
17C4 A9 00          DATAM    LDA #$00
17C6 85 FF          DATAM1   STA MODE           ;SET MODE FLAG TO DATA
17C8                ;
17C8 A9 7F          DEBNCE   LDA #DELAY         ;SET DEBOUNCE TIME DELAY
17CA 8D 07 17                STA TIMER
17CD 20 19 1F       TIME     JSR SCAND          ;KEEP DISPLAY LIT
17D0 2C 07 17                BIT TIMER          ; IF TIME NOT UP YET
17D3 10 F8                   BPL TIME           ;KEEP CHECKING TIMER
17D5 18                      CLC                ;WHEN TIME IS UP,
17D6 90 AB                   BCC START          ; JUMP BACK TO START
17D8                ;
17D8 20 63 1F       STEP     JSR INCPT
17DB 18                      CLC                ;CALL ROUTINE TO EXECUTE
17DC 90 EA                   BCC DEBNCE         ; THE + COMMAND & DEBOUNCE
17DE                ;
17DE                         END
```

Listing 2: DEBOUNCED BROWSE for the KIM-1

Modified version of BROWSE to eliminate keybounce on the KIM-1 keypad.
(Original version of BROWSE by Jim Butterfield is in the first book of KIM . Modification by Jody Nelis - K3JZD.)

August, 1980

Hex Dump Produced by KIM

```
KIM
0000 AD G
0110    D8 A9 13 85 FE A9 00 85 FA 85 FB C6 F3 D0 0E A5
0120    FD F0 0A 10 79 A5 FA D0 02 C6 FB C6 FA 20 19 1F
0130    20 6A 1F C5 FE F0 E4 85 FE C9 15 F0 DE A2 00 86
0140    FD C9 10 90 1C 86 F4 C9 11 F0 01 E8 86 FF C9 12
0150    D0 02 E6 FD C9 14 D0 02 C6 FD C9 13 D0 CF 4C C8
0160    1D 0A 0A 0A 0A 85 FC A2 04 A4 FF D0 17 C6 F4 10
0170    07 20 63 1F E6 F4 E6 F4 B1 FA 06 FC 2A 91 FA CA
0180    D0 F8 F0 A9 0A 26 FA 26 FB CA D0 F8 F0 9F A9 7F
0190    8D 07 17 20 19 1F 2C 07 17 10 F8 18 90 8F 20 63
01A0    1F 18 90 89

KIM
01A4 EA
```

Running Instructions

BOUNCELESS is totally relocatable. Load it anywhere you wish. Use the object portion of the disassembly (listing 1) in the same way you woud use a hex dump. Save it at the beginning of your utility cassette since you will need it all of the time.

When you enter BOUNCELESS, nothing obvious will happen. This is normal. BOUNCELESS has taken command of all of the keyboard input now. It will respond to the AD, DA, + and #0 - #F hex keys in the normal fashion, but without the bounce. It will ignore the GO and PC keys.

When your data input is finished, return to the KIM monitor with the RS key. You can enter and exit BOUNCELESS at any time without upsetting anything else in user memory.

(Continued on next page)

DEBOUNCED BROWSE is also totally relocatable. Load it anywhere using listing 2. The operating instructions remain exactly the same as the original version. Refer to *The First Book of KIM* for all of BROWSE's features.

If you still get an occasional bounce when running either program, try increasing the value of the time delay byte. In BOUNCELESS this is $17C9 and in DEBOUNCED BROWSE it is $018F. Increasing the value will increase the delay.

Jody Nelis has been working with a KIM-1 since 1977. He bought it primarily to educate himself in the workings of small computers. Even though he also has an AIM-65 now, he still continues to work with the KIM-1 and highly recommends it to anyone desiring an excellent educational tool at a price that can't be beat.

MICRO

Listing 3

```
                    ;DEBOUNCED BROWSE
                    ;
                    ; MODIFICATION TO THE
                    ; ORIGINAL 'BROWSE'
                    ; BY JIM BUTTERFIELD
                    ; (THE MODIFIED PROGRAM
                    ; REMAINS COMPLETELY RELOCATABLE)
                    ;
                    ; MODIFIED BY JODY NELIS
                    ;
                    DELAY    EPZ $7F
                    ;CHANGE THE FOLLOWING LINE IN THE ORIGINAL 'BROWSE'
                             ORG $0123
                    ;
0123 10 79                   BPL UP           ;(NEW BRANCH OFFSET)
                    ;
                    ;ENTER NEW PROGRAMMING FROM $018E TO END
                    ;
                             ORG $018E
                    ;
018E A9 7F          DEBNCE   LDA #DELAY       ;SET DEBOUNCE TIME DELAY
0190 8D 07 17                STA TIMER        ; TO .13 SEC.
0193 20 19 1F TIME           JSR SCAND        ;KEEP DISPLAY LIT
0196 2C 07 17                BIT TIMER        ;IF TIME NOT UP YET
0199 10 F8                   BPL TIME         ;KEEP CHECKING TIMER
019B 18                      CLC              ;WHEN TIME IS UP
019C 90 8F                   BCC LP1          ;JUMP BACK TO LP1
019E                ;
019E 20 63 1F UP             JSR INCPT        ;CALL ROUTINE TO
01A1 18                      CLC              ; INCREMENT ADDR
01A2 90 89                   BCC LP1          ; FOR BROWSING
01A4                ;
01A4                         END
```

Binary Storage and Array Retrieval

A technique for extremely fast I/O of arrays on disk is presented. It is accompanied by a demonstration program and a discussion of the representation of arrays in Applesoft. The method may be extended to other floating-point BASIC languages.

Hank Blakely
122 6th Street, S.E.
Washington, D.C. 20003

The DOS textfile facility, although very useful for other purposes, is not an especially good method for storing and retrieving array data. In fact, compared to other DOS routines, particularly the binary save and load process, the textfile is extremely slow. An array of 30,000 elements requires two to three minutes to be saved to a textfile. An equivalent amount of data can be zipped onto the disk as a binary file in about thirty seconds.

Moreover, since textfiles are literal, rather than symbolic representations of data, they do not use space efficiently. An array value such as 123.123456 requires 11 bytes — one for each character — to be stored in a textfile. Compare this to the five bytes required for floating-point storage in core memory. Also, since textfiles are finicky about accepting such delimiters as "," and ":", they are not normally capable of saving and loading absolutely faithful images of string arrays.

Of course, textfiles do offer certain advantages over other forms of storage. For example, the APPEND command will link data to an existing file without needing to first read the file, add data, and then rewrite. The POSITION command, the B(yte) parameter specification, and the random access configuration allow the operating system to "reach into" the file and extract or replace specific records, fields or characters.

However, for those storage and retrieval applications that are not concerned with that level of manipulation, (particularly in cases where data is to be written read as an entire file), it is much faster and less cumbersome to BSAVE arrays directly to disk as binary files.

The following is a generalized method for implementing this process in Applesoft or, by extension, any similarly configured member of the Microsoft BASIC family. Although the techniques are comparatively simple, they need to be approached with a little forethought and some understanding of the principles involved.

How Applesoft Variables are Stored

As they are encountered within a program, all Applesoft variables are stored in a space that begins at the LOMEM setting (usually coincident with the end of the program itself, but occasionally set higher). The variable space is partitioned into two segments for simple and array variables.

The addresses for the first segment, simple variables, are stored in decimal locations 105/106 and 107/108. The addresses for the array variables in the topmost segment, are in locations 107/108 and 109/110.

These two segments contain all of the information necessary to allow an Applesoft program to find and manipulate all of the numeric variables. Strings, however, are another matter. Although the program and the numeric variables build from the bottom of user memory, strings start at the HIMEM setting and extend downward toward the top of the numeric value segment. In fact, it is the constantly narrowing gap between these two stacks that leads to the frequently sudden and annoying "garbage collection" process. The memory location of the last string stored is designated "start of string storage" and its address is at decimal locations 111/112.

The string storage area, however, contains only the literal elements of the strings. Applesoft locates these elements by referring to addresses located within the appropriate segment of the variable space that "point" to each string.

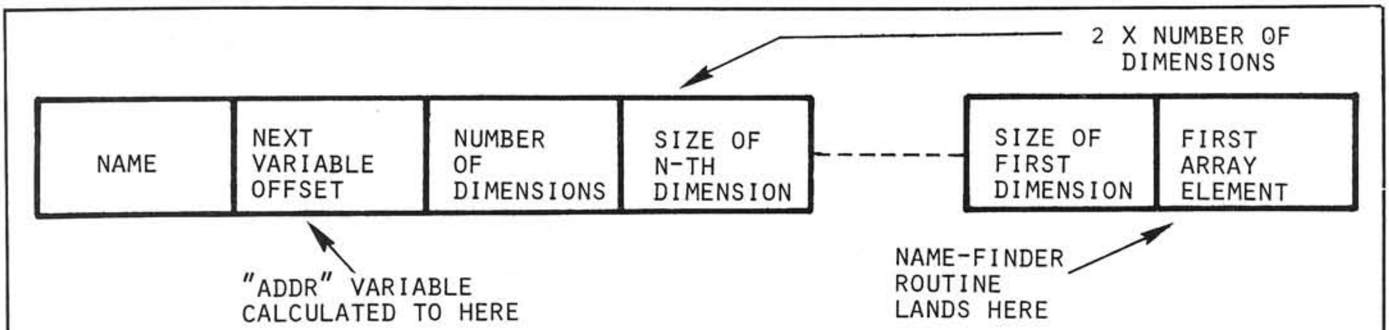


Figure 1: **Common Header for Variable Files**

Applesoft Array Storage

Each array within the array stack is defined by a variable file — a unique range of data that consists of two parts: an introductory header containing information relating to the nomenclature and structure of the array, and a trailer that contains the field of array values, or pointers to string array elements.

As shown in figure 1, the first seven bytes of the header are exactly the same for all three variable types, and are allocated as follows:

Two bytes for the first two characters in the variable's name. These are ASCII equivalents, and are uniquely coded by turning the high bit of each character on or off, to indicate the type of variable.

Two bytes for the value of an offset which, when added to the initial address of the current variable file, gives the address of the next variable in memory, and therefore implies the absolute length of the current file.

One byte for the number of dimensions in the array.

Two bytes for the size of the Nth array dimension.

From here the header adds two bytes for each dimension in the array. Since Applesoft limits the number of dimensions in an array to 88, the header length will always be between seven and 181 bytes.

As shown in figure 2, the length of the trailer for each file varies according to the number and type of the array elements. Each real variable requires five bytes to accommodate the floating-point format. Each integer value requires two bytes for reverse-order notation (most significant byte first). Each string pointer requires three bytes — one for the string length, and two for the address.

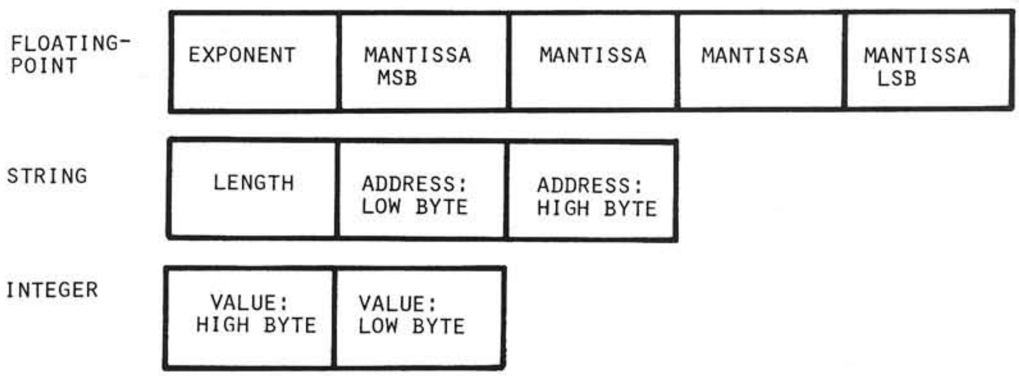


***Figure 2:* Structure for Each Array Element by Type**

Storing and Retrieving Arrays

Program B.MODE demonstrates the array filing method.

A choice of "W" from the selection menu creates three test arrays based on the squares and cubes of 10 elements (two 100-element integer and real types, and one 1000-element string type), and transfers control to the first address calculation routine "strings" at line 410.

Line 420 equates the zero-th element of the array to itself. This has the effect of invoking an Applesoft internal locator routine that first locates the array in the variable space, and then deposits its address in decimal locations 131 and 132. The values in these two registers are then transferred to two "safe" locations so they will not be lost when another variable is referenced. DMEN is then set to the number of dimensions, and control is transferred to the SAVE and LOAD section beginning at line 340.

The SAVE and LOAD routine calculates the location and length (ADDR and LGTH) of the variable file by starting at the address in the "safe" locations and counting back three bytes plus two times the number of dimensions to the next-variable offset. The resultant range of data is then BSAVEd to disk. The same calculations are performed for the "R" (for "Read") selection, but, of course, the length parameter is not used.

This basic process is the same for the storage and retrieval of all three array types. There is, however, one important difference in the process for string arrays. Since the actual strings are not located in the same area as their pointers, it is necessary to determine their locations and establish a separate file for them. Accordingly, line 440 resets ADDR to the location of the zero-th element, and sets LGTH equivalent to the difference between ADDR and STRG. This memory range is then either BSAVEd or BLOADed, depending on the menu selection.

```
100  REM        B.MODE

110 D$ =  CHR$ (4):FILE$ = "":SL$ = ""
120  DIM CHAR$(9,9),IGER%(9,9,9),REAL(9,9)
130 ADDR = 0:LGTH = 0:STRG = 0:J = 0:K = 0:DMEN = 0
140  REM

150  REM  ---------------------------------CONTROL
160  TEXT : HOME : VTAB 23: HTAB 1: INPUT "READ OR WRITE (R/W) "
     ;SL$: IF S  L$ <  > "R" AND SL$ <  > "W" THEN 160
170  IF SL$ = "R" THEN 410
180  IF SL$ = "W" THEN  GOSUB 200: GOTO 410
190  REM

200  REM --------------------------------TEST ARRAYS
210 STRG =  PEEK (111) +  PEEK (112) * 256
220  FOR J = 0 TO 9: FOR K = 0 TO 9
230 CHAR$(J,K) = "ROW " +  STR$ (J) + "," + "COL " +  STR$ (K)
240  NEXT : NEXT
250  FOR J = 0 TO 9: FOR K = 0 TO 9
260 REAL(J,K) = J * 10 + (K / 10)
270  NEXT : NEXT
280  FOR J = 0 TO 9: FOR K = 0 TO 9: FOR I = 0 TO 9
290 IGER%(J,K,I) = J * 100 + K * 10 + I
300  NEXT : NEXT : NEXT
310  PRINT  FRE (0); CHR$ (7)
320  TEXT : RETURN
330  REM

340  REM ---------------------------------SAVE & LOAD
350 ADDR = ( PEEK (254) +  PEEK (255) * 256) - 3 - 2 * DMEN
360 LGTH = ( PEEK (ADDR) +  PEEK (ADDR + 1) * 256)
370  IF SL$ = "W" THEN  PRINT D$"BSAVE "FILE$;",A";ADDR;",L"LGTH
380  IF SL$ = "R" THEN  PRINT D$"BLOAD "FILE$;",A";ADDR
390  RETURN
400  REM
410  REM  -------------------------------STRINGS
420 CHAR$(0,0) = CHAR$(0,0): POKE 254, PEEK (131): POKE 255,
     PEEK (132)
430 DMEN = 2:FILE$ = "CF.FILE": GOSUB 340
440 ADDR =  PEEK (ADDR + 4 + 2 * DMEN) +  PEEK (ADDR + 5 + 2 * DMEN)
     * 256
450 LGTH = STRG - ADDR
460  IF SL$ = "W" THEN  PRINT D$"BSAVE CS.FILE,A"ADDR",L"LGTH
470  IF SL$ = "R" THEN  PRINT D$"BLOAD CS.FILE,A"ADDR
480  REM --------------------------------REALS
490 REAL(0,0) = REAL(0,0): POKE 254, PEEK (131): POKE 255,
     PEEK (132)
500 DMEN = 2:FILE$ = "RL.FILE": GOSUB 340
510  REM --------------------------------INTEGERS
520 IGER%(0,0,0) = IGER%(0,0,0): POKE 254, PEEK (131): POKE 255,
     PEEK (132 )
530 DMEN = 3:FILE$ = "IG.FILE": GOSUB 340
540  REM

550  REM --------------------------------DISPLAY
560  PRINT  CHR$ (7)
570  TEXT : HOME
580  PRINT "STRINGS:": FOR J = 0 TO 9: FOR K = 0 TO 9:
     PRINT CHAR$(J,K)" ";: NEXT : PRINT : PRINT : NEXT
590  VTAB 23: HTAB 1: INPUT "NEXT ";SL$
600  TEXT : HOME
610  PRINT "REALS:": FOR J = 0 TO 9: FOR K = 0 TO 9: PRINT REAL
     (J,K)"  ";:
      NEXT : PRINT : PRINT : NEXT
620  VTAB 23: HTAB 1: INPUT "NEXT ";SL$
630  TEXT : HOME
640  PRINT "INTEGERS:": FOR J = 0 TO 9: FOR K = 0 TO 9:
     FOR I = 0 TO 9: PRINT
     IGER%(J,K,I)"  ";: NEXT : PRINT : PRINT : NEXT : PRINT : NEXT
```

Comparisons and Conditions

The differences, in terms of speed of execution and storage space, are impressive, and grow more so as the arrays increase in size. Figure 3 compares read and write timings and sector storage for text *vs.* binary files for the 10-element base test array. It also shows similar comparisons for the same array limited to five elements (25 numerics, and 125 strings), and expanded to 20 elements (400 numerics and 8000 strings).

Although the improvements in speed and space utilization depend on the size of the elements involved, it appears that most arrays can be saved and loaded in from four to 14 times faster than textfiles, and may take up only half the space for numeric arrays, while sacrificing only a few sectors for strings.

When using this technique, one or two conditions need to be observed. First, the address calculations for the

	10-ELEMENT BASE TEXT	10-ELEMENT BASE BMODE	20-ELEMENT BASE TEXT	20-ELEMENT BASE BMODE	5-ELEMENT BASE TEXT	5-ELEMENT BASE BMODE
WRITE TIME	0:57	0:14	6:44	0:37	0:13	0:11
READ TIME	0:51	0:10	6:31	0:28	0:08	0:07
SPACE; SECTS						
CHAR. - TEXT	6	6	22	20	3	3
CHAR. - PTRS	-	3	--	6	-	2
REALS	3	4	10	9	2	2
INTEGERS	17	9	143	64	3	3

***Figure 3:* Comparison of Timing and Sector Usage for Various-Sized Types of Arrays.**

string elements are based on the position of the zero-th element of the array. If that element is located within the program itself, the SAVE and LOAD routine will consider its position to be the start of the string file, and will attempt to save everything from there up to STRG.

Second, from the point where the zero-th element is first invoked to the point where the array is saved or loaded, any variables, particularly simple variables, referenced or assigned, must have been previously dimensioned. Otherwise the variable stack will be shifted up to accommodate them, and the locational references will become meaningless. This is not a problem once the BSAVE or BLOAD process has been completed.

Finally, the FRE(0) statement is absolutely necessary to force a garbage collection and to reduce the size of the string file.

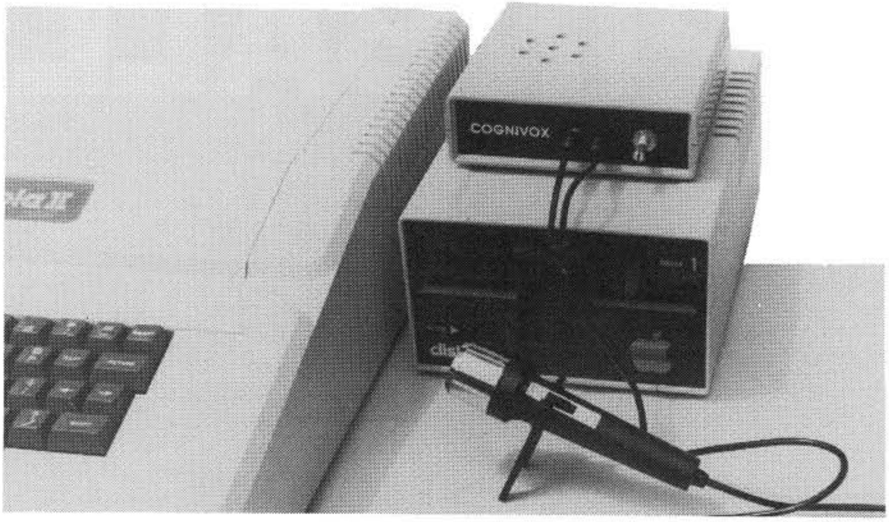
Extension to Other Systems

This method should be simple to implement in other versions of Microsoft BASIC, or any floating-point BASIC that structures and stores its variables in a similar manner. Or it could be implemented on a BASIC that has some provision for locating and saving the addresses of specific array elements. Given those prerequisites, all you need to do now is to determine the system-specific addresses for array space, string storage, and the locator's variable address dump, and to make the appropriate substitutions.

Hank Blakely is president of the Athena Group, a federally-certified corporation specializing in microsystem design for business and government. He has used an Apple II for three years, and is fascinated with graphics, artificial intelligence, and robotics.

MICRO

IF YOU CONSIDER THE PRICE, COGNIVOX AT $249 IS THE BEST BUY IN VOICE I/O.
IF YOU CONSIDER THE PERFORMANCE, YOU WILL BUY IT.
BECAUSE COGNIVOX OUTPERFORMS ALL VOICE INPUT OR OUTPUT PERIPHERALS FOR THE APPLE II.
AT ANY PRICE.



COGNIVOX VIO-1003 is a state-of-the-art Speech Recognition and voice output peripheral for the APPLE II computer. It enables the APPLE to recognize words or short phrases spoken by the user and it can talk with natural sounding voice.

SPEECH RECOGNITION

COGNIVOX recognizes words (such as "one," "enter," etc.) or short phrases (like "total amount," "net weight," etc.) from a vocabulary of 32 entries. The vocabulary entries are chosen by the user to suit his application. Then COGNIVOX is "trained" to the vocabulary by repeating each entry three times into the microphone under the prompting of the system.

During training, COGNIVOX analyzes the voice of the user and compresses all the important information in each entry into 48 bytes of data called the reference pattern. When training is complete, words spoken in the microphone are similarly analyzed and the resulting 48-bit pattern is compared with all the reference patterns to obtain a best match.

The power of COGNIVOX is derived from proprietary pattern generation and pattern matching algorithms that allow quick and easy training and give a recognition accuracy equal to much more expensive units.

Vocabularies larger than 32 words are possible by swapping reference patterns in memory using a key word, for example, "change vocabulary." Or the swap can be performed under program control.

VOICE OUTPUT

COGNIVOX can talk with a vocabulary of 32 words or short phrases. No restrictions are placed on the vocabulary which can be programmed simply by saying the words into the microphone. The speech waveform is then digitized using a data compression method and stored in memory.

When voice output is desired, the selected word or phrase is reconstructed and played back using a built-in speaker/amplifier. A jack is also provided that allows connection to external amplifiers or speaker.

This method of voice output offers two very important advantages: First, the user has full control over the selection of the vocabulary and the type and tone of voice. Second, the voice output is naturally sounding human speech which is pleasant and easy to understand. These features are not available in most other voice output devices in the market.

The voice output and speech recognition vocabularies are independent of each other and can be different. Thus it is possible to establish a dialog with the computer.

USING COGNIVOX

COGNIVOX is designed for extreme ease of use. It is a complete system, fully assembled and tested, including hardware in an instrument case, microphone, power supply, cassette with software and user manual. It plugs into the game I/O port in the APPLE and does not use up the valuable peripheral slots.

Software provided with COGNIVOX include demonstration programs and two voice operated, talking video games. All programs are unprotected so that the user can examine and modify them.

An optional diskette for DOS 3.3 includes all cassette software plus disk facilities to store and retrieve vocabularies on disk.

Adding voice I/O to your own programs is very simple. A statement in BASIC is all that is needed to either recognize or say a word. Complete instructions on how to add voice to your programs are given in the manual.

APPLICATIONS

COGNIVOX adds a whole new dimension to man-computer interaction. It can be used for data and command entry when hands and/or eyes are busy. As an educational tool. As an aid to handicapped. As sound effects generator. As a telephone answering machine. As a talking calculator, or talking clock.

The list is endless. With a BSR home controller interface it can be used to control by voice appliances and lights around the house. With an IEEE 488 interface card it can be used to control by voice instruments, plotters, test systems. And all these devices could talk back, saying their readings, alarm conditions, even their name. Finally, COGNIVOX is a super toy, a fascinating device to play with. Imagine an adventure game that talks to warn you of danger and listens to your commands!

SPECIFICATIONS

Recognizer type:
Isolated word, speaker dependent.
Vocabulary size:
32 words or short phrases for both recognition and voice response.
Dialog capability:
Recognition and response vocabularies can be different.
Word Duration
Greater than 150 ms and less than 3 seconds.
Silence gap between words:
150 ms minimum.
Training required:
Must pronounce vocabulary 3 times to train recognizer. Allows words to be individually retrained.
Recognition accuracy:
Up to 98%. Recognition accuracy depends on speaker experience and choice of vocabulary.
Type of voice output:
Digital recording of user voice.
Audio output:
130 mW
Frequency response:
100 to 3200 Hz.
Power consumption:
120 mW during recognition, 350 mW maximum during speech output.
Power supply:
9V DC, 300 mA, unregulated.
Dimensions:
5"x 6"x 1.25"
Memory requirements:
Approx. 4K bytes for program and tables. 1.5K bytes per sec. of speech for storage of voice response vocabulary (Approx. 700 bytes per word).

ORDER COGNIVOX TODAY
To order COGNIVOX by phone, call us at (805) 685-1854, 9AM to 4PM PST. Monday through Friday, and charge it to your MASTERCARD or VISA. Or you can order by mail. Send us a check or money order for $249 plus $5 shipping and handling (CA residents add 6% tax). Software on diskette (DOS-3.3) order part # DSK-3.3, $19. Foreign orders welcome, add 10% for air mail shipping and handling. COGNIVOX is backed by a 120 day warranty against manufacturing defects.

VOICETEK

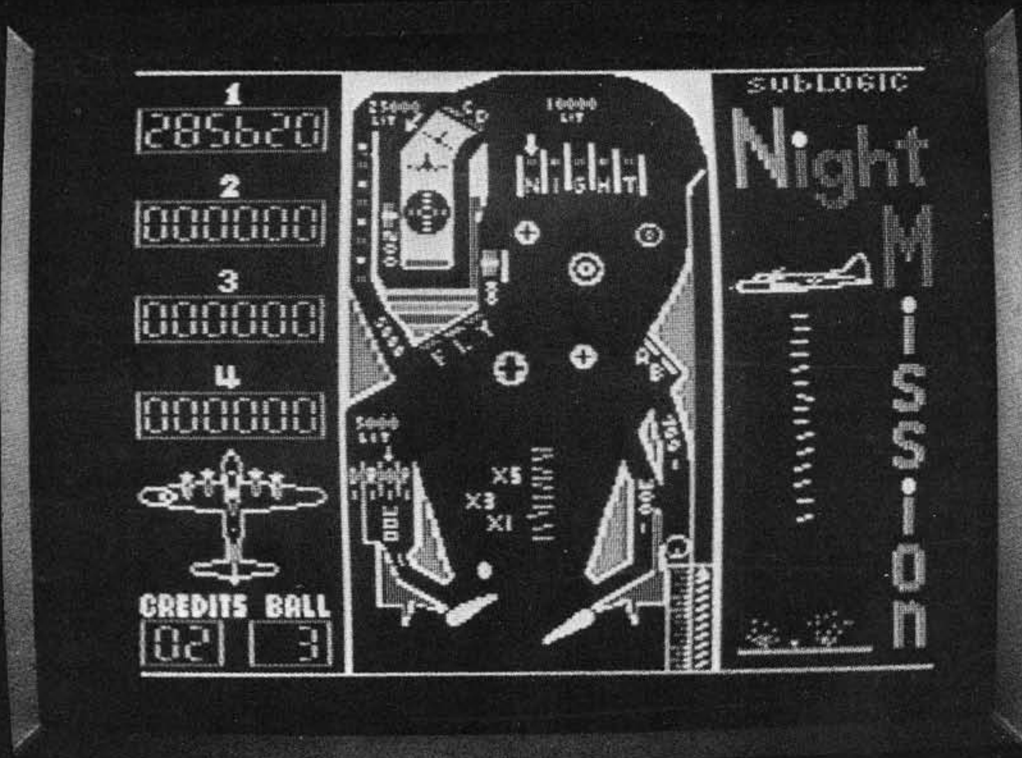
Dept G, P.O. Box 388
Goleta, CA 93116

ALSO AVAILABLE for COMMODORE COMPUTERS and the AIM-65.
CALL or WRITE for MORE INFO.

Introducing...

A2·PB1

PINBALL



This beautiful and sophisticated simulation offers superior ball action, superior flipper control, and many subtle perfections which will result in your superior enjoyment.

Please don't compare A2-PB1 Pinball with other pinball simulations. Nothing else even comes close. $29.95

The engineering and graphics experts opening a new era in computer simulation.

subLOGIC

Communications Corp.
713 Edgebrook Drive
Champaign, IL 61820
(217) 359-8482
Telex: 206995

See your dealer . . .

or for direct orders, add $1.50 and specify UPS or first class mail.
Illinois residents add 5% sales tax.
Visa and MasterCard accepted.

A Disassembler for the 6809

Here's a description of the operation of a disassembler for the 6809 microprocessor. The disassembler is written in Microsoft BASIC and will run either on the Apple II computer (using Applesoft) or on the Radio Shack Color Computer.

Mark J. Borgerson
1624 NW Kings Blvd.
Corvallis, Oregon 97330

In this article I will describe the development and use of a simple disassembler for the Motorola M6809 microprocessor. The disassembler is written in Microsoft BASIC and the source code appears at the end of the article. The program is designed to run on the Apple II computer. I originally developed the program on a Radio Shack color computer, but transferred it to the Apple to take advantage of the superior operating environment provided by Apple DOS 3.3. Since both Applesoft and Radio Shack Color BASIC were originally developed by Microsoft, transferring the program was fairly simple. Only a few changes in the input and output routines were needed to make the program run on the Apple. Conversely, it should be simple to make the necessary changes in the program so that it will, once again, run on the Radio Shack computer.

Disassembler Fundamentals

A disassembler is a program which will read a machine language program, either from memory or from some storage device, then produce a listing of the opcode mnemonics and their associated data bytes. Some disassemblers generate output that includes labels for all branches and referenced data locations. This type of disassembler, called a "source code generator," can be used to produce a text file which may be modified and reassembled.

Ø:	NEG	D	***	V	***	V	COM	D	LSR	D	***	V	ROR	D	ASR	D
8:	ASL	D	ROL	D	DEC	D	***	D	INC	D	TST	D	JMP	D	CLR	D
16:	***	V	***	V	NOP	H	SYNC	H	***	V	***	V	LBRA	L	LBSR	L
24:	***	V	DAA	H	ORCC	M	***	V	ANDCC	M	SEX	H	EXG	1	TFR	1
32:	BRA	R	BRN	R	BHI	R	BLS	R	BCC	R	BCS	R	BNE	R	BEQ	R
4Ø:	BVC	R	BVS	R	BPL	R	BMI	R	BGE	R	BLT	R	BGT	R	BLE	R
48:	LEAX	X	LEAY	X	LEAS	X	LEAU	X	PSHS	2	PULS	2	PSHU	2	PULU	2
56:	***	V	RTS	H	ABX	H	RTI	H	CWAI	2	MUL	H	***	V	SWI	H
64:	NEGA	H	***	V	***	V	COMA	H	LSRA	H	***	V	RORA	H	ASRA	H
72:	ASLA	H	ROLA	H	DECA	H	***	V	INCA	H	TSTA	H	***	V	CLRA	H
8Ø:	NEGB	H	***	V	***	V	COMB	H	LSRB	H	***	V	RORB	H	ASRB	H
88:	ASLB	H	ROLB	H	DECB	H	***	V	INCB	H	TSTB	H	***	V	CLRB	H
96:	NEG	X	***	V	***	V	COM	X	LRS	X	***	V	ROR	X	ASR	X
1Ø4:	ASL	X	ROL	X	DEC	X	***	V	INC	X	TST	X	JMP	X	CLR	X
112:	NEG	E	***	V	***	V	COM	E	LSR	E	***	V	ROR	E	ASR	E
12Ø:	ASL	E	ROL	E	DEC	E	***	V	INC	E	TST	E	JMP	E	CLR	E
128:	SUBA	M	CMPA	M	SBCA	M	SUBD	3	ANDA	M	BITA	M	LDA	M	***	V
136:	EORA	M	ADCA	M	ORA	M	ADDA	M	CMPX	3	BSR	R	LDX	3	***	V
144:	SUBA	D	CMPA	D	SBCA	D	SUBD	D	ANDA	D	BITA	D	LDA	D	STA	D
152:	EORA	D	ADCA	D	ORA	D	ADDA	D	CMPX	D	JSR	D	LDX	D	STX	D
16Ø:	SUBA	X	CMPA	X	SBCA	X	SUBD	X	ANDA	X	BITA	X	LDA	X	STA	X
168:	EORA	X	ADCA	X	ORA	X	ADDA	X	CMPX	X	JSR	X	LDX	X	STX	X
176:	SUBA	E	CMPA	E	SBCA	E	SUBD	E	ANDA	E	BITA	E	LDA	E	STA	E
184:	EORA	E	ADCA	E	ORA	E	ADDA	E	CMPX	E	JSR	E	LDX	E	STX	E
192:	SUBB	M	CMPB	M	SBCB	M	ADDD	3	ANDB	M	BITB	M	LDB	M	***	V
2ØØ:	EORB	M	ADCB	M	ORB	M	ADDB	M	LDD	3	***	V	LDU	3	***	V
2Ø8:	SUBB	D	CMPB	D	SBCB	D	ADDD	D	ANDB	D	BITB	D	LDB	D	STB	D
216:	EORB	D	ADCB	D	ORB	D	ADDB	D	LDD	D	STD	D	LDU	D	STU	D
224:	SUBB	X	CMPB	X	SBCB	X	ADDD	X	ANDB	X	BITB	X	LDB	X	STB	X
232:	EORB	X	ADCB	X	ORB	X	ADDB	X	LDD	X	STD	X	LDU	X	STU	X
24Ø:	SUBB	E	CMPB	E	SBCB	E	ADDD	E	ANDB	E	BITB	E	LDB	E	STB	E
248:	EORB	E	ADCB	E	ORB	E	ADDB	E	LDD	E	STD	E	LDU	E	STU	E

Table 1: **Mnemonics and Type Indicators for the First 255 Opcodes**

256:	1Ø21LBRN	L	1Ø22LBHI	L	1Ø23LBLS	L	1Ø24LBCC	L
26Ø:	1Ø25LBCS	L	1Ø26LBNE	L	1Ø27LBEQ	L	1Ø28LBVC	L
264:	1Ø29LBVS	L	1Ø2ALBPL	L	1Ø2BLBMI	L	1Ø2CLBGE	L
268:	1Ø2DLBLT	L	1Ø2ELBGT	L	1Ø2FLBLE	L	1Ø3FSWI2	H
272:	1Ø83CMPD	3	1Ø8CCMPY	3	1Ø8ELDY	3	1Ø93CMPD	D
276:	1Ø9CCMPY	D	1Ø9ELDY	D	1Ø9FSTY	D	1ØA3CMPD	X
28Ø:	1ØACCMPY	X	1ØAELDY	X	1ØAFSTY	X	1ØB3CMPD	E
284:	1ØBCCMPY	E	1ØBELDY	E	1ØBFSTY	E	1ØCELDS	3
288:	1ØDELDS	D	1ØDFSTS	D	1ØEELDS	X	1ØEFSTS	X
292:	1ØFELDS	E	1ØFFSTS	E	113FSWI3	H	1183CMPU	3
296:	118CCMPS	3	1193CMPU	D	119CCMPS	D	11A3CMPU	X
3ØØ:	11ACCMPS	X	11B3CMPU	E	11BCCMPS	E	LAST MNEMONIC	

Table 2: **Opcodes and Mnemonics for Special Opcodes**

The simpler disassemblers, like the one accompanying this article, simply produce a listing of addresses, opcodes and data bytes. The output is not suitable for reassembly without a lot of editing. The simpler disassemblers are generally used to examine code in memory, either to verify code you have written yourself, or to delve into the operation of code written by someone else. For instance, I have used the 6809 disassembler on the color computer to examine the code in the BASIC ROMs to find out how BASIC uses different areas of memory.

There are two possible approaches to take in writing a disassembler. First you examine each opcode byte, break it into a bit pattern which represents the mnemonic and addressing mode, then look up the proper mnemonic string in a table. If you examine a table of opcodes for the 6809, you will find that certain combinations of bits in the opcode bytes always occur with a particular addressing mode. By using logic and bit manipulation, you can deduce the proper mnemonic and addressing mode for the opcode byte. (This is the type of disassembler built into the Apple monitor.) However, I discarded this approach for two reasons:

1. The bit manipulations involved are most easily done in machine language — I was working with BASIC.
2. The wide variety of addressing modes available for the 6809 makes this approach more complex than when using a simpler processor like the 6800 or 6502.

The second approach to writing a disassembler is to use each opcode byte as an index into a table which contains both the mnemonic string and an indicator of the addressing mode. I chose this method because it is well-suited to implementation in Microsoft BASIC, which has an excellent string array facility. This approach requires a data array of at least 256 strings to hold each of the mnemonics (or an indicator for an illegal opcode byte — the 6809 has 34 of these). This method must also cope with the fact that there are 47 two-byte opcodes for the 6809.

The two-byte opcodes all have a first byte which is either $10 or $11. This simplifies the procedure for matching the opcode bytes to the mnemonics somewhat, but we would still need two more tables of 256 strings if we want to use the second byte as an index into a table for these instructions. Rather than set up this additional array space, I simply added the hexadecimal opcode representation to the beginning of the mnemonic string. Now, whenever the first byte of an opcode is $10 or $11, I simply search the mnemonic array until I find an entry which has the same hexadecimal representation as the code in memory.

Each mnemonic string ends with a single character which indicates the addressing mode for the opcode. Table 1 shows the mnemonic strings for the single byte opcodes. Table 2 displays the mnemonics and addressing modes for the two-byte opcodes. The address mode characters indicate the following types of addressing:

D Direct page
V Invalid opcode
R Relative addressing
X Indexed addressing
H Inherent mode
E Extended (16-bit) addressing
M Immediate mode (with 8-bit data)
L Long branch (16-bit offset)
1 Push or pull with single post-byte
2 Two-register mode (as in TFR X,D)
3 Immediate addressing with 16-bit data

The program (listing 2) runs in a tight loop contained in lines 110 to 390. In this loop the first opcode byte is read from memory (line 130) and the matching mnemonic is determined. Opcodes are matched to mnemonics in subroutines beginning at lines 7000 and 7500. The first of these determines mnemonics for single-byte opcodes, the second the mnemonics for two-byte opcodes.

The series of IF statements in lines 170-270 determines the addressing mode and call subroutines appropriate to each addressing mode. See table 3 for a list of the most important variables in the program and their functions.

Table 3

MN$(305)	The array of mnemonic strings
PR	Device number for output 1 = console, 2 = printer
O$	The string which contains the output line of disassembled code
HW	A number representing a hexadecimal 16-bit word
HW$	A string which contains a representation of a 16-bit hexadecimal word
BT	A number representing a hexadecimal byte
BT$	A string representing a byte in hexadecimal form
AD	The address of the next byte to be fetched from memory
OP	The value of the most recently fetched opcode
EA	A number representing an effective address — usually the target of a branch opcode
PB	The post byte in an indexed mode instruction
IM	A number representing the particular indexing mode determined by a post byte
CD$	A string which contains the data bytes for any data following the opcode
RR	A byte which represents the registers to be pushed or pulled
RR$	The hexadecimal string representation of RR

Using the Program

The disassembler is very simple to use. When you run the program it will ask for the hexadecimal address where you start the disassembly. Once you have entered the address, it will ask you whether you want the output directed to the screen or the printer. After disassembly begins, the program is not particularly fast. This isn't much of a problem since the output scrolls by at a comfortable reading rate when it is directed to the screen. If I find a particularly interesting spot, I halt the program by hitting any key on the keyboard. Hitting another key will resume the disassembly, hitting an ESC will restart the program and ask you again for the starting address and output device. Listing 1 is a sample of the output. (This particular code is part of a video driver for the Apple.)

I hope this program will be useful to any of you who are using the 6809 processor. Although BASIC may not be the best language for writing this type of program, sometimes you have to use the language available. This was certainly the case when I first purchased my Color Computer. Even if you are not using the 6809, the techniques used to decipher the bit patterns may interest you if you would like to work with other processors.

Listing 1: Sample of Disassembled Code

```
7877:   34 36      PSHS   A,B,X,Y,
7879:   7D C000    TST    $C000
787C:   2A 0C      BPL    $788A
787E:   17 0171    LBSR   $79F2
7881:   81 93      CMPA   #$93
7883:   26 05      BNE    $788A
7885:   17 016A    LBSR   $79F2
7888:   A6 E4      LDA    ,S
788A:   8A 80      ORA    #$80
788C:   81 A0      CMPA   #$A0
788E:   2C 4B      BGE    $78DB
7890:   81 8D      CMPA   #$8D
7892:   27 56      BEQ    $78EA
7894:   81 8A      CMPA   #$8A
7896:   27 54      BEQ    $78EC
7898:   81 87      CMPA   #$87
789A:   26 05      BNE    $78A1
789C:   35 36      PULS   A,B,X,Y,
789E:   16 00B6    LBRA   $7957
78A1:   81 88      CMPA   #$88
78A3:   26 17      BNE    $78BC
78A5:   0A 24      DEC    $24
78A7:   2A 11      BPL    $78BA
78A9:   96 20      LDA    $20
78AB:   9B 21      ADDA   $21
78AD:   4A         DECA
78AE:   97 24      STA    $24
78B0:   0D 25      TST    $25
78B2:   27 06      BEQ    $78BA
78B4:   0A 25      DEC    $25
78B6:   96 25      LDA    $25
78B8:   8D 7E      BSR    $7938
78BA:   35 B6      PULS   A,B,X,Y,PC,
78BC:   81 9D      CMPA   #$9D
78BE:   26 04      BNE    $78C4
78C0:   8D 51      BSR    $7913
78C2:   35 B6      PULS   A,B,X,Y,PC,
78C4:   81 8C      CMPA   #$8C
78C6:   26 05      BNE    $78CD
78C8:   17 009D    LBSR   $7968
```

Listing 2: 6809 Disassembler

```
10  REM   6809 DISASSEMBLER
15  HIMEM: 20000
20  REM    M.BORGERSON
30  REM    12/25/80
40  DIM MN$(305):H$ = "0123456789ABCDEF"
50  GOSUB 9000: REM    READ DATA
60  REM   GET STARTING ADDRESS
70  HOME : PRINT "        6809 DISASSEMBLER V1.0"
74 PR = 0
80  PRINT : INPUT "HEX ADDRESS:";HA$
90  GOSUB 9100: REM   CONVERT TO DECIMAL NUMBER
100 AD = HD
105  INPUT "OUTPUT TO SCREEN (1), OR PRINTER (2)?";PR
106  IF (PR < 1) OR (PR > 2) THEN 105
110  REM   NOW GO INTO LOOP
115 O$ = ""
120 HW = AD: GOSUB 8000:O$ = HW$ + ":"
130 OP =  PEEK (AD): IF ((OP = 16) OR (OP = 17)) THEN  GOSUB 7500: GOTO 150
140  GOSUB 7000
150  REM   NOW ASSEMBLE OPCODE STRING
170  IF TY$ = "H" THEN  GOSUB 400
180  IF TY$ = "D" THEN  GOSUB 500
190  IF TY$ = "R" THEN  GOSUB 1000
200  IF TY$ = "M" THEN  GOSUB 1500
210  IF TY$ = "E" THEN  GOSUB 2000
220  IF TY$ = "V" THEN  GOSUB 2500
230  IF TY$ = "X" THEN  GOSUB 3000
240  IF TY$ = "1" THEN  GOSUB 4000
250  IF TY$ = "2" THEN  GOSUB 4500
260  IF TY$ = "L" THEN  GOSUB 5000
270  IF TY$ = "3" THEN  GOSUB 5500
280  GOSUB 6000: REM  PRINT OPCODE STRING
290  IF  PEEK (49152) > 128 THEN 350
300  GOTO 110
310  REM  *****************************************************
350  REM  KEYBOARD HALT
360  GET Q$
370  IF Q$ =  CHR$ (27) THEN  PRINT  CHR$ (4);"PR#0": PRINT : GOTO 80
390  GET Q$: GOTO 110
400  REM   INHERENT MODE
420 O$ = O$ + CD$ + "          " + MN$: RETURN
500  REM   DIRECT MODE
```

(Continued)

MICRObits

(Continued from page 56)

OSI 65D V3.3. Guide

Contains the bugs, fixes and other data OSI didn't tell you about. Increase compatibility and run ability between 65D V3.X and V3.3. Run extended utilities under V3.3 and much more. Price $14.95 plus $1.00 shipping.

Buffalo Informational Technologies
209 Richmond Ave.
Buffalo, N.Y. 14222

6800/6809 Software

Includes compatible single-user, multi-user and network-operating systems, compilers, accounting and word processing packages. Free catalog.

Software Dynamics
2111 W. Crescent, Sta. G
Anaheim, CA 92801

Librarian/Contour Mapping

Filing and keyword retrieval system for books, articles, excerpts, etc. Produces automatic bibliographies. Initial data base included. $69.95. Contour Mapping: high quality contouring system for printer, plotter or CRT. Data and grid transformations, trend surfaces, residuals. Data manager. Geological and geographical software for Apple and other systems.

Geosystems, Inc.
802 E. Grand River
Williamston, MI 48895
(517) 655-3726

OSI Superboard Cabinet and Accessories

Pre-cut pine cabinet kit $27.95 ppd; RS-232 interface kit $9.95 ppd; 600 baud/band kit $8.95 ppd., complete with hardware and well-illustrated instruction sets. Free catalog of software, hardware, Molex, joysticks, cassettes, kits and more.

Dee Products
Department M
150 Birchwood Road
Lake Marion, IL 60110

Matrix Operations in Pascal on the Apple

Program matrixcalc provides menu-driven matrix calculations (including inverse and determinant), matrix scaling, and matrix I/O (disk, console or printer) ($45.00). Easily use matrix operations in your Pascal programs with *Unit Matrix30* ($45.00). Other Pascal software available.

Monterey Science and Engineering Co
177 Webster Street A307
Monterey, CA 93940

Computer Covers

Keep dust and dirt out of your equipment with a computer cover. Cover for full Apple $11.50, disk drive $4.25, stack disk (2) $6.25, monitor (state size) $10.25. Send check plus $1.50 postage.

M.P. Computer Services
P.O. Box 2396 Encinal Stations
Sunnyvale, CA 94087

Listing 2 *(Continued)*

```
510 BT =  PEEK (AD):AD = AD + 1: GOSUB 8500
520 O$ = O$ + CD$ + " " + BT$ + "      " + MN$ + "$" + BT$
530  RETURN
1000  REM   RELATIVE MODE
1020 EA =  PEEK (AD):AD = AD + 1
1030 BT = EA: GOSUB 8500:O$ = O$ + CD$ + " " + BT$ + "      " + MN$
1035  IF EA > 128 THEN EA = EA - 256
1040 EA = AD + EA:HW = EA: GOSUB 8000:O$ = O$ + "$" + HW$
1070  RETURN
1200  REM   LONG BRANCH
1210 EA = 256 *  PEEK (AD) +  PEEK (AD + 1):AD = AD + 2
1220 HW = EA: GOSUB 8000:O$ = O$ + CD$ + " " + HW$ + " " + MN$
1230  IF EA > 32767 THEN EA = EA - 65536
1240 HW = EA: GOSUB 8000:O$ = O$ + "$" + HW$
1250  RETURN
1500  REM   IMMEDIATE MODE
1510 EA =  PEEK (AD):AD = AD + 1
1540 BT = EA: GOSUB 8500:O$ = O$ + CD$ + " " + BT$ + "      " + MN$ + "#$" +
  BT$
1550  RETURN
2000  REM   EXTENDED MODE
2010 EA =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2
2020 HW = EA: GOSUB 8000:O$ = O$ + CD$ + " " + HW$ + "    " + MN$ + "$" + HW$

2040  RETURN
2500  REM   INVALID OP CODE
2510 O$ = O$ + CD$ + "         ???????"
2520  RETURN
3000  REM   INDEXED MODE
3005 EA$ = ""
3010 PB =  PEEK (AD):AD = AD + 1: IF PB < 128 THEN 3900
3015 BT = PB: GOSUB 8500:CD$ = CD$ + " " + BT$
3020 PB = PB - 128:RR =  INT (PB / 32):AM = PB - 32 * RR
3030 IM = 0: IF AM > 15 THEN IM = 1:AM = AM - 16
3040  IF IM = 1 THEN MN$ = MN$ + "("
3050  ON AM + 1 GOSUB 3100,3150,3200,3250,3300,3350,3400,3450,3500,3550,
  3600,3650,3700,3750,3800,3850
3070  IF IM = 1 THEN O$ = O$ + ")"
3080  RETURN
3100  REM   AUTO INCREMENT BY 1
3110 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + ",": GOSUB 3950:O$ = O$ + "+"

3120  RETURN
3150  REM   AUTO INCREMENT BY 2
3160 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + ",": GOSUB 3950:O$ = O$ + "++
 "
3170  RETURN
3200  REM   AUTO DECREMENT BY 1
3210 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + ",-": GOSUB 3950
3220  RETURN
3250  REM   AUTO DECREMENT BY 2
3260 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + ",--": GOSUB 3950
3270  RETURN
3300  REM   ZERO OFFSET
3310 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + ",": GOSUB 3950
3320  RETURN
3350  REM   ACC B OFFSET
3360 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + "B,": GOSUB 3950
3370  RETURN
3400  REM   ACC A OFFSET
3410 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + "A,": GOSUB 3950
3420  RETURN
3450  REM   NOT VALID
3460 O$ = O$ + CD$ + "        ?????": RETURN
3500  REM   8-BIT OFFSET
3510 CD$ = CD$ + " ":BT =  PEEK (AD):AD = AD + 1: GOSUB 8500
3520 CD$ = CD$ + BT$ + " ":EA = BT: IF EA > 127 THEN EA = EA - 256
3530 O$ = O$ + CD$ + " " + MN$ +  STR$ (EA) + ",": GOSUB 3950
3540  RETURN
3550  REM   16-BIT OFFSET
3560 CD$ = CD$ + " ":HW =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2:
  GOSUB 8000
3570 CD$ = CD$ + HW$:O$ = O$ + CD$ + MN$:EA = HW: IF EA > 32767 THEN EA = EA
  - 65536
3580 O$ = O$ +  STR$ (EA) + ",": GOSUB 3950
3590  RETURN
3600  REM   INVALID POST-BYTE
3610 O$ = O$ + CD$ + "  ???": RETURN
3650  REM   OFFSET BY D
3660 CD$ = CD$ + "      ":O$ = O$ + CD$ + MN$ + "D,": GOSUB 3950
3670  RETURN
3700  REM   8-BIT PCR
3710 BT =  PEEK (AD):AD = AD + 1: GOSUB 8500:CD$ = CD$ + " " + BT$ + " "
3720 EA = BT: IF EA > 127 THEN EA = EA - 256
3730 O$ = O$ + CD$ + MN$ +  STR$ (EA) + ",PCR"
3740  RETURN
3750  REM   16-BIT PCR
3760 HW =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2: GOSUB 8000:CD$ = CD
  $ + " " + HW$
3770 EA = HW: IF EA > 32767 THEN EA = EA - 65536
3780 O$ = O$ + CD$ + MN$ +  STR$ (EA) + ",PCR"
3790  RETURN
3800  REM  INVALID
```

(Continued)

```
381Ø O$ = O$ + "       ?????"
382Ø  RETURN
385Ø  REM   INDIRECT ADDRESS
386Ø HW =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2: GOSUB 8ØØØ
387Ø CD$ = CD$ + " " + HW$:O$ = O$ + CD$ + "       " + MN$ + "$" + HW$
388Ø  RETURN
39ØØ  REM   5-BYTE OFFSET
391Ø RR =  INT (PB / 32):EA = PB - RR * 32: IF EA > 15 THEN EA = EA - 32
392Ø BT = PB: GOSUB 85ØØ:O$ = O$ + CD$ + " " + BT$ + "       " + MN$
393Ø O$ = O$ +  STR$ (EA) + ",": GOSUB 395Ø
394Ø  RETURN

395Ø  ON RR + 1 GOTO 396Ø,3961,3962,3963
3955 O$ = O$ + "?": RETURN
396Ø O$ = O$ + "X": RETURN
3961 O$ = O$ + "Y": RETURN
3962 O$ = O$ + "U": RETURN
3963 O$ = O$ + "S": RETURN
4ØØØ  REM   2-REGISTER TYPE
4Ø1Ø RR =  PEEK (AD):AD = AD + 1
4Ø2Ø R1 =  INT (RR / 16):R2 = RR - 16 * R1
4Ø3Ø BT = RR: GOSUB 85ØØ:O$ = O$ + CD$ + " " + BT$ + "       " + MN$
4Ø4Ø RR = R1: GOSUB 42ØØ:O$ = O$ + RR$ + ",":RR = R2: GOSUB 42ØØ:O$ =
     O$ + R R$
4Ø5Ø  RETURN
4Ø6Ø  REM  ******************************************
42ØØ  REM   BREAK RR INTO RR$
421Ø RR = RR + 1: REM  SHIFT INTO RANGE
422Ø  ON RR GOTO 424Ø,425Ø,426Ø,427Ø,428Ø,429Ø,43ØØ,431Ø,432Ø,433Ø,434Ø,
     435Ø
423Ø RR$ = "?": RETURN
424Ø RR$ = "D": RETURN
425Ø RR$ = "X": RETURN
426Ø RR$ = "Y": RETURN
427Ø RR$ = "U": RETURN
428Ø RR$ = "S": RETURN
429Ø RR$ = "PC": RETURN
43ØØ RR$ = "?": RETURN
431Ø RR$ = "?": RETURN
432Ø RR$ = "A": RETURN
433Ø RR$ = "B": RETURN
434Ø RR$ = "CC": RETURN
435Ø RR$ = "DP": RETURN
45ØØ  REM   1 REGISTER PUSH/PULL
451Ø EA =  PEEK (AD):AD = AD + 1:O$ = O$ + CD$ + " ":BT = EA: GOSUB
     85ØØ:O$ = O$ + BT$ + "      " + MN$
452Ø RP$ = ""
453Ø  IF EA > 127 THEN RP$ = "PC," + RP$:EA = EA - 128
454Ø  IF EA > 63 THEN RP$ = "U," + RP$:EA = EA - 64
455Ø  IF EA > 31 THEN RP$ = "Y," + RP$:EA = EA - 32
456Ø  IF EA > 15 THEN RP$ = "X," + RP$:EA = EA - 16
457Ø  IF EA > 7 THEN RP$ = "DP," + RP$:EA = EA - 8
458Ø  IF EA > 3 THEN RP$ = "B," + RP$:EA = EA - 4
459Ø  IF EA > 1 THEN RP$ = "A," + RP$:EA = EA - 2
46ØØ  IF EA > Ø THEN RP$ = "C," + RP$
461Ø O$ = O$ + RP$: RETURN
5ØØØ  REM   LONG BRANCH
5Ø1Ø EA =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2
5Ø2Ø HW = EA: GOSUB 8ØØØ:O$ = O$ + CD$ + " " + HW$ + "    " + MN$
5Ø3Ø  IF EA > 32768 THEN EA = EA - 65536
5Ø4Ø HW = AD + EA: GOSUB 8ØØØ:O$ = O$ + "$" + HW$
5Ø5Ø  RETURN
55ØØ  REM   2-BYTE IMMEDIATE DATA
551Ø HW =  PEEK (AD) * 256 +  PEEK (AD + 1):AD = AD + 2
552Ø  GOSUB 8ØØØ:O$ = O$ + CD$ + " " + HW$ + "    " + MN$ + "#$" + HW$
553Ø  RETURN
554Ø  REM  ******************************************
6ØØØ  REM   OUTPUT TO SELECTED DEVICE
6Ø1Ø O$ =  LEFT$ (O$,17) + " " +  RIGHT$ (O$, LEN (O$) - 17)
6Ø2Ø  IF PR = 1 THEN  PRINT O$
6Ø3Ø  IF PR = 2 THEN  GOTO 65ØØ
6Ø4Ø  RETURN
6Ø5Ø  REM   RETURN TO MAIN LOOP
6Ø6Ø  REM   ****************************************
65ØØ  REM   SEND OUTPUT TO PRINTER IN SLOT #1
651Ø  PRINT  CHR$ (4);"PR#1"
652Ø  PRINT O$
653Ø  PRINT  CHR$ (4);"PR#Ø"
654Ø  RETURN
655Ø  REM   ****************************************
7ØØØ  REM   GET OP & MNEMONIC FOR STANDARD
7Ø1Ø SP = Ø:BT = OP: GOSUB 85ØØ:CD$ = "  " + BT$
7Ø2Ø MN$ =  LEFT$ (MN$(OP),6):TY$ =  RIGHT$ (MN$(OP),1)
7Ø3Ø AD = AD + 1: RETURN
75ØØ  REM   DETERMINE OP & MNEONIC FOR SPECIAL
751Ø HW = OP * 256 +  PEEK (AD + 1):AD = AD + 2:SP = 1
752Ø  GOSUB 8ØØØ:CD$ = HW$
753Ø  FOR I = 256 TO 3Ø3: IF  LEFT$ (MN$(I),4) = CD$ THEN MN$ =  RIGHT$
     (MN$ (I),7)
754Ø  NEXT I
755Ø TY$ =  RIGHT$ (MN$,1):MN$ =  LEFT$ (MN$,6)
756Ø  RETURN
757Ø  REM  ********************************************
```

MICRObits *(continued)*

Priced Right Apple Clock

Clock maintains in 16 memory address: MTH, DOM, WKD, and TOD to 1/10 seconds. Also maintains leap year and provides three (3) possible repeated or software set timed interrupts. Battery backed up. Priced right with documentation at $95.95, less batteries.

Buchanan
1870-B NW 59th Avenue
Sunrise, FL 33313
(305) 739-8160

Expand Apple Memory!

Memory boards from 16K to 256K by Microsoft, Saturn, Legend, SVA, etc. at lowest prices. Also, 8'' floppy drives and hard disks by LOBO, CORVUS, and SVA at big discounts!

Connecticut Informations Systems
218 Huntington Road
Bridgeport, CT 06608
(203) 579-0472

HTS KIM/AIM/OSI C1P Products

Timestack Operating System implements programmable, time-based controller. Handles programming event sequences, then controls performance of events. User-written event subroutines can handle I/O processing like data acquisition, power switching. Extensive documentation. Software manuals, $30.00. Specifications, $1.00. Catalog, free.

Hunter Technical Services
P.O. Box 359
Elm Grove, WI 53122

Free Classified Ad

for you in Consumer Electronics Marketplace. Buy, sell, trade microcomputers, software, video, hifi. Send your ad and $6.00 for subscription:

Consumer Electronics Marketplace
Box 86E
Highland Mills, NY 10930

OSI Superboard II, C1P

Star Chase: dodge the asteroids and try to get to your base, but watch out for the critters from hyper-space! Runs under 4K. $4.95. Send for free catalog with other programs.

Swany's OSI Software
2652 37th West
Seattle, WA 98199
(206) 282-7376

OSI Software

Debug lets you trace and breakpoint your machine code programs. *Universe* plays like *Cobra Copter* with keyboard or joystick. *Edit-All* has scroll window output, full cursor control and dynamic insert/delete. All in machine code. For C1P, C2-4 computers.

DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

MICRO

Borgeson Listing 2 *(Continued)*

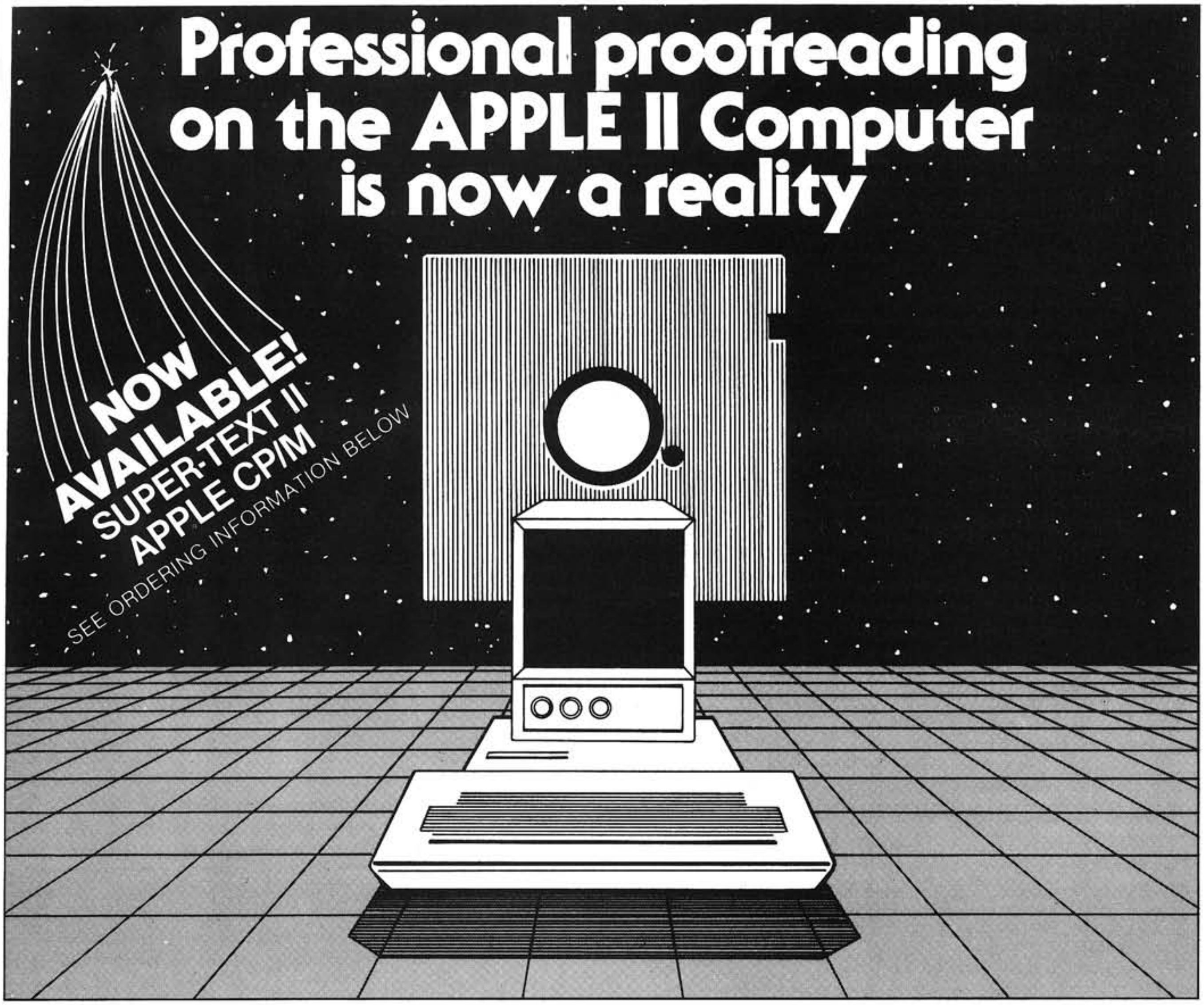
```
8000  REM    PUT HW INTO HEX HW$
8010 WH =  INT (HW / 256):WL = HW - 256 * WH
8020 BT = WH: GOSUB 8500:HW$ = BT$
8030 BT = WL: GOSUB 8500:HW$ = HW$ + BT$
8040  RETURN
8500  REM    PUT BT INTO HEX BT$
8510 NH =  INT (BT / 16):NL = BT - 16 * NH
8520 BT$ =  MID$ (H$,NH + 1,1) +  MID$ (H$,NL + 1,1)
8530  RETURN
9000  REM    READ DATA
9010  RESTORE
9020  FOR I = 0 TO 303: READ MN$(I): NEXT I
9030  RETURN
9100  REM    CONVERT HEXT TO DECIMAL
9110 HD = 0
9120  FOR I = 1 TO  LEN (HA$)
9130 HC =  ASC ( MID$ (HA$,I,1)) - 48: IF HC > 9 THEN HC = HC - 7
9140 HD = 16 * (HD + HC)
9150  NEXT I
9160 HD = HD / 16: RETURN
9500  REM    MNEMONIC DATA
9510  DATA  NEG   D,***   V,***   V,COM   D,LSR   D,***   V,ROR   D,ASR   D
9520  DATA  ASL   D,ROL   D,DEC   D,***   D,INC   D,TST   D,JMP   D,CLR   D
9530  DATA  ***   V,***   V,NOP   H,SYNC  H,***   V,***   V,LBRA  L,LBSR  L
9540  DATA  ***   V,DAA   H,ORCC  M,***   V,ANDCC M,SEX   H,EXG   1,TFR   1
9550  DATA  BRA   R,BRN   R,BHI   R,BLS   R,BCC   R,BCS   R,BNE   R,BEQ   R
9560  DATA  BVC   R,BVS   R,BPL   R,BMI   R,BGE   R,BLT   R,BGT   R,BLE   R
9570  DATA  LEAX  X,LEAY  X,LEAS  X,LEAU  X,PSHS  2,PULS  2,PSHU  2,PULU  2
9580  DATA  ***   V,RTS   H,ABX   H,RTI   H,CWAI  2,MUL   H,***   V,SWI   H
9590  DATA  NEGA  H,***   V,***   V,COMA  H,LSRA  H,***   V,RORA  H,ASRA  H
9600  DATA  ASLA  H,ROLA  H,DECA  H,***   V,INCA  H,TSTA  H,***   V,CLRA  H
9610  DATA  NEGB  H,***   V,***   V,COMB  H,LSRB  H,***   V,RORB  H,ASRB  H
9620  DATA  ASLB  H,ROLB  H,DECB  H,***   V,INCB  H,TSTB  H,***   V,CLRB  H
9630  DATA  NEG   X,***   V,***   V,COM   X,LRS   X,***   V,ROR   X,ASR   X
9640  DATA  ASL   X,ROL   X,DEC   X,***   V,INC   X,TST   X,JMP   X,CLR   X
9650  DATA  NEG   E,***   V,***   V,COM   E,LSR   E,***   V,ROR   E,ASR   E
9660  DATA  ASL   E,ROL   E,DEC   E,***   V,INC   E,TST   E,JMP   E,CLR   E
9670  DATA  SUBA  M,CMPA  M,SBCA  M,SUBD  3,ANDA  M,BITA  M,LDA   M,***   V
9680  DATA  EORA  M,ADCA  M,ORA   M,ADDA  M,CMPX  3,BSR   R,LDX   3,***   V
9690  DATA  SUBA  D,CMPA  D,SBCA  D,SUBD  D,ANDA  D,BITA  D,LDA   D,STA   D
9695  DATA  EORA  D,ADCA  D,ORA   D,ADDA  D,CMPX  D,JSR   D,LDX   D,STX   D
9700  DATA  SUBA  X,CMPA  X,SBCA  X,SUBD  X,ANDA  X,BITA  X,LDA   X,STA   X
9710  DATA  EORA  X,ADCA  X,ORA   X,ADDA  X,CMPX  X,JSR   X,LDX   X,STX   X
9720  DATA  SUBA  E,CMPA  E,SBCA  E,SUBD  E,ANDA  E,BITA  E,LDA   E,STA   E
9730  DATA  EORA  E,ADCA  E,ORA   E,ADDA  E,CMPX  E,JSR   E,LDX   E,STX   E
9740  DATA  SUBB  M,CMPB  M,SBCB  M,ADDD  3,ANDB  M,BITB  M,LDB   M,***   V
9750  DATA  EORB  M,ADCB  M,ORB   M,ADDB  M,LDD   3,***   V,LDU   3,***   V
9760  DATA  SUBB  D,CMPB  D,SBCB  D,ADDD  D,ANDB  D,BITB  D,LDB   D,STB   D
9770  DATA  EORB  D,ADCB  D,ORB   D,ADDB  D,LDD   D,STD   D,LDU   D,STU   D
9780  DATA  SUBB  X,CMPB  X,SBCB  X,ADDD  X,ANDB  X,BITB  X,LDB   X,STB   X
9790  DATA  EORB  X,ADCB  X,ORB   X,ADDB  X,LDD   X,STD   X,LDU   X,STU   X
9800  DATA  SUBB  E,CMPB  E,SBCB  E,ADDD  E,ANDB  E,BITB  E,LDB   E,STB   E
9810  DATA  EORB  E,ADCB  E,ORB   E,ADDB  E,LDD   E,STD   E,LDU   E,STU   E
9820  DATA  1021LBRN  L,1022LBHI  L,1023LBLS  L,1024LBCC  L,1025LBCS  L,1026
     LBNE  L,1027LBEQ  L
9830  DATA  1028LBVC  L,1029LBVS  L,102ALBPL  L,102BLBMI  L,102CLBGE  L,102D
     LBLT  L,102ELBGT  L,102FLBLE  L
9840  DATA  103FSWI2  H,1083CMPD  3,108CCMPY  3,108ELDY   3,1093CMPD  D,109C
     CMPY  D,109ELDY   D,109FSTY   D
9850  DATA    10A3CMPD  X,10ACCMPY  X,10AELDY   X,10AFSTY   X,10B3CMPD  E,10
     BCCMPY  E,10BELDY   E,10BFSTY   E
9860  DATA  10CELDS   3,10DELDS   D,10DFSTS   D,10EELDS   X,10EFSTS   X,10FE
     LDS   E,10FFSTS   E
9870  DATA  113FSWI3  H,1183CMPU  3,118CCMPS  3,1193CMPU  D,119CCMPS  D,11A3
     CMPU  X,11ACCMPS  X,11B3CMPU  E,11BCCMPS  E
9880  DATA  "LAST MNEMONIC"
```

MICRO

Write For MICRO!

We want to begin including applications-oriented articles in MICRO. How are you using your micro(s) these days? What, in your view, is the best interface between software and user, and why? Do you have any thoughts about databases and/or networking? Are you familiar with the 68000 chip? What would you like to write for or read in MICRO?

We want to provide prospective authors with any information and support they may need, promptly. We want MICRO to be the preferred information exchange for a wider range of computerists. If you have ideas for articles — or just ideas you would like to discuss — please write or call Laurence Kepple, Senior Editor at MICRO, P.O. Box 6502, Chelmsford, MA 01824; (617) 256-5515.



THE APPLE SPELLER fills the void that has consistently kept the large variety of excellent word processing packages for the Apple II Computer from approaching the power of a dedicated work processor. Finally, the first professional quality spelling verification program is available for the Apple II. The Apple Speller will certainly be the standard against which all other similar programs are compared.

The Apple Speller interfaces to the most popular Apple word processors, including Applewriter, Apple Pie, Executive Secretary, Letter Perfect, Magic Window, and Superscribe II, just to name a few. In fact, The Apple Speller can analyze the output of any editor that writes a standard Apple binary or text file to a diskette. In addition to this flexibility, the performance of The Apple Speller will astound the microcomputer world.

The Apple Speller is supplied with a 30,000+ word dictionary on a single 5¼" diskette with additional space to easily add another 8,000 words to suit your individual needs. The Apple Speller has built-in utilities to maintain the dictionary diskette. You can readily add words, delete words, and create an unlimited number of modified and/or new dictionaries for specific application.

The Apple Speller is unbelievably fast. The first pass reads your document and collects all the words it contains at a rate of 5,000 words per minute. Next, the words are compared to the dictionary for spelling errors at the incredible speed of 50,000 words per minute. Finally, all misspelled words are marked as such in your document with a rate of 1,000 words per minute. This translates to proofreading a 10 page document in 1 minute if there are no spelling mistakes and 2 minutes, 15 seconds for an unlimited number of spelling errors.

Numerous options are provided throughout the program to enable you to completely control all activities of The Apple Speller. These include the ability to ignore both control codes and formatting commands, an alphabetical listing of either mispelled words or all the words in your document along with usage frequencies, multiple options for the action taken with each misspelled word, and much, much more! A verification mode is provided to allow you to examine and dispense with misspelled words while viewing them in the actual context in which they appeared in your file.

The Apple Speller requires an Apple II/Apple II+ equipped with 48K, DOS 3.3, and 1 or 2 disk drives. Two disk drives are required to delete or add words to the dictionary.

The Apple Speller is being introduced at the incredible price of

$75.00!

Sensible Software
6619 Perham Drive Dept. MO
West Bloomfield, Michigan 48033
(313) 399-8877
Visa and Mastercard Welcome
Please add $1.25 postage and handling per diskette
Please specify program desired

APPLE is a registered trademark of APPLE Computer Company Registered Trademark of DIGITAL RESEARCH CORP. Registered Trademark of MUSE SOFTWARE

MICRO PET Vet

By Loren Wright

COMAL is a relatively new language developed in Denmark. The PET/CBM version is called "CBM COMAL 80," and was written especially for Commodore (Europe, not U.S.) by Mogens Kjaer. The name COMAL is an acronym for COMmon Algorithmic Language. It began as a few extensions to BASIC, but, as more was added, it became apparent that COMAL should stand alone. COMAL does indeed have a lot in common with BASIC, but the influence of Pascal is equally apparent. The intent was to combine the best features of BASIC (easy to learn, easy to program, interactive with the best features of Pascal, more powerful, easy-to-follow structured program flow). COMAL's creators have accomplished that goal.

Program Editing

Editing of COMAL programs is very similar to BASIC. However, there are convenient RENUMber, AUTO-number, and DELete commands. It is also possible to ENTER lines from a disk file. This makes building procedure libraries very easy.

The successively indented listing format, which shows the structure of a program, is provided automatically.

Program Operation

The COMAL disk includes two versions of the interpreter. One consists of the whole interpreter in one program file. The other splits it into "input" and "execute" modules. The combined version is easier to use, particularly when learning the language, but the split version allows much longer programs (15,358 bytes *vs.* 4949) and includes the PRINT USING function.

Features of COMAL

Variables:

COMAL supports numerical, string, and Boolean variables, as well as multi-dimensional arrays of numbers, and one-dimensional arrays of strings. There is apparently no distinction between floating point numbers and integers (and their different memory requirements). Names may be up to 16 characters, all of which are significant. String manipulation is simpler than it is in BASIC, but just as powerful.

Program Flow:

Procedures in COMAL work similarly to those of Pascal. They are called with an EXEC statement or as part of a numerical expression. The structure supports both one-way and two-way parameter passing, and a procedure may be CLOSED to make all of its internal variables local. By assigning a value to the procedure name before exiting, the procedure may be used as a function in an expression.

The best way to determine the power of a language is to examine how it makes decisions. PET BASIC has two decision-making structures: IF...THEN and ON...GOTO (or GOSUB). You can use these to solve just about any programming problem, but often the program flow becomes complex and error-prone. COMAL extends the IF...THEN structure to include ELSE and ELIF (=ELSE IF) functions. ON...GOTO and ON...GOSUB are replaced by the CASE structure, which, unlike many Pascal implementations, includes an ELSE capability.

COMAL has three structures that allow controlled repetition of a program segment. FOR...NEXT works exactly as it does in BASIC, except that it is possible to have a one-line loop without a NEXT. REPEAT...UNTIL allows a program segment to be repeated until a condition (tested at the end of the segment) is satisfied. DO ...WHILE...ENDWHILE allows repetition of a program segment until a condition (tested at the beginning of the segment) fails.

There is also a GOTO in COMAL, but it is used only to transfer control to a label, which is defined with a name followed by a colon on a line by itself.

Other Features

COMAL's PRINT function is similar to BASIC's, but it allows a little more flexibility. PRINT USING, implemented only in the split version of the interpreter, does a good job of formatting numerical data.

Most of the familiar BASIC built-in functions are supported. The RND function generates pseudo-random integers over a specified range.

Evaluation

COMAL is an excellent compromise between Pascal and BASIC. It is easy to learn, and the system is easy to operate. The gain over BASIC in structure, power, and readability is considerable. Few programmers really need all of the power of Pascal and some versions are actually less powerful than COMAL. The exacting nature of Pascal's syntax makes programming more difficult and tedious.

If you work on a number of different computers, you will find that knowledge of COMAL is not particularly transportable, but that may change. The price is right, and I recommend getting a copy, if only to get a taste of high-level programming.

The Future of COMAL

Much of the future of COMAL is in the hands of Commodore. Rumor has it that COMAL will eventually replace BASIC as the language supplied with CBM, PET, and VIC computers. There are already ROM versions in existence for nearly every Commodore machine, and there is an enhanced version for the 8096 which is extremely powerful. There is talk of a prototype color version in England, running in a 40-column VIC. As yet, none of this has been confirmed by Commodore, US.

Where to Get COMAL

COMAL was originally distributed (by arrangement with Commodore) by the COMAL Users' Group (5501 Groveland Terrace, Madison, WI 53716). Although they are no longer distributing the COMAL Starter Kit, they are still a good source of information. Many users' groups have the COMAL interpreters already in their libraries, since the disk versions were placed in the public domain by Commodore. If your group doesn't have it, contact another group or the COMAL Users' Group.

Most of the information on COMAL is published in Europe. Ellis Horwood Ltd. (Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB, England) has two books available: *Structured Programming with COMAL-80* by Roy Atherton and a tutorial by Borge Christensen. Len Lindsay's *COMAL Handbook* should be available soon from Reston Publishing Co. (Reston, VA).

MICRO

MICRO Software Catalog

Name: **Accounting System** (Integrated portion of EIS General Acct. System)
System: OS-65U
Memory: 48K
Language: BASIC
Hardware: Ohio Scientific C-2 or C-3 series

Description: Keeps detailed records of all transactions and generates income statements and balance sheets to provide information on fiscal activities.

Price: $1,500.00 includes 3 program disks and a step-by-step user's manual

Author: Electronic Information Systems, Inc.

Available:
Electronic Information Systems, Inc.
P.O. Box 5893
Athens, GA 30604
(404) 353-2858

Name: **Snow Watch**
System: Apple II Plus
Memory: 48K
Language: CP/M
Hardware: Disk II, Printer

Description: Computerized school and business closings for use in a severe weather emergency by radio and television stations. Schools phone in unique code numbers to tell whether they are open or closed. Program completely organizes status reports and messages. Prints full or update reports for on-air use.

Price: $350.00 includes diskette, documentation, and consultation

Author: Roger Skolnik

Available:
Media Service Concepts, Inc.
1713 N. North Park
Chicago, IL 60614

Name: **Arnold**
System: Apple II or Apple II Plus
Memory: 48K
Language: Exended BASIC
Hardware: MMI DAC board with Apple II or Apple II Plus

Description: Designed to teach tone recognition and melodic memory skills, *Arnold* asks you to recall and enter the tones of an ever-increasing melody by using solfeggio syllables or scale degree numbers. Utilizes patterns from 95 graded melodies with five levels of skill difficulty, beginning to very difficult. *Arnold* keeps your progress record on the disk.

Price: $190.00 includes program diskette and manual

Author: J. Timothy Kolosick

Available:
Micro Music, Inc.
P.O. Box 386
Normal, IL 61761
(309) 452-6991

Name: **Galactic Chase**
System: Atari 800/400
Memory: 16K
Language: Assembly

Description: Fast moving attack and destroy game featuring several skill levels to challenge accomplished gamesmen as well as beginners. *Galactic Chase* utilizes the extensive graphics capabilities of the Atari computer. Colorful creatures attack from the far reaches of space. The game is designed for one or two players, captains of a star ship that is the last defender of space.

Price: $24.95 cassette
$29.95 diskette

Available:
Spectrum Computers
26618 Southfield Rd.
Lathrup Village, MI 48076
(313) 559-5252

Name: **Edit All**
System: OSI Disk Systems
Memory: 1K at top of memory
Language: Machine
Hardware: OSI C1P MF, C2-4 MF

Description: *Edit All* is a full screen editor for OSI computers. It replaces the standard I/O routines to allow the user to edit any program line that is on the screen. As editing takes place, the line is dynamically expanded or contracted. *Edit All* supports a scroll window screen handler that allows you to define where on the screen you want your output to go. All output to the screen is via a window whose length, height and width are all user-changeable. Full cursor control is supported along with an instant screen clear. *Edit All* works with OS65D BASIC and Assembler.

Price: $19.95 includes 5¼" disk, documentation

Author: Dave Pompea

Available:
DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

Name: **EZAID**
System: PET/CBM
Memory: 8K - 32K
Language: Assembler
Hardware: New ROMs, cassette or disk, 40- or 80-column screen

Description: 4K EPROM chip which is available for any free socket and is intended for use with the EZASM chip. Any area of memory, even if not the execution address, may be disassembled in two passes, producing source code which is 100% compatible with the EZASM Assembler. User-defined labels may be included for clarity and you can make modifications to the source code generated as you wish. FIND function with optional replacement field and no restrictions as to BASIC keywords or whatever. AUTO line numbering. Renumber EZASM source program. Delete a block of lines. Repeat keys.

Price: $80.00 including shipping and instructions.

Author: Milton B. Bathurst

Available:
DataCap
73 Rue du Village
4545 Feneur
Belgium

Name: **Statistics with Daisy**
System: Apple II
Memory: 48K
Language: Applesoft ROM
Hardware: Apple II, disk drive with DOS 3.3

Description: *Daisy* offers a full range of statistical capabilities and excels on user conveniences. Statistics with *Daisy* is a statistical analysis package suitable for business, scientific, and social science applications. Some of its features include: full user assistance facilities HELP and INFO, math and time-series transforms, Hi-Res plots, basic statistics (mean, standard deviation, etc.), correlations, multiple regression (6 different procedures), model testing and evaluation, nonparametric statistics, hypothesis testing, and analysis of variance. Users can add their own programs as new *Daisy* commands. Disk commands exist to save, enter, examine, and overlay dozens of variables or hundreds of abbreviations. Data entry is through a "window" view into the data table.

Price: $79.95 includes disk and manual

Author: Kevin C. Killion

Available:
Rainbow Computing
19517 Business Center Dr.
Northridge, CA 91324

Name: **Merlin Dial/Data**
System: Apple II and Apple II Plus
Memory: 48K
Language: BASIC (Applesoft)
Hardware: Apple II or Apple II Plus, two disk drives, micro model

Description: Allows Apple user immediate access to Merlin data base which has been used by investment professionals for more than a decade. Gives daily and historical price information for all securities, options and commodities on all major exchanges. Automatic access and file handling. All prices are updated daily and system is Compu-trac compatible. Also available to other micro users who wish to write their own programs.

Price: $50 Apple software, monthly usage charges based on use of daily pricing service. Includes manual, data base creation and maintenance plus automatic access to Merlin DIAL/DATA time sharing system for prices.

Available:
Remote Computing Corp.
Dept. MS
1044 Northern Blvd.
Roslyn, NY 11576
(516) 484-4545

Name: **Mail Mate**
System: Apple II Plus
Memory: 48K
Language: BASIC (Applesoft) and Assembly

Description: *Mail Mate* is a mail-merge system that can operate with the Magic Window word processing system, or by itself as a stand-alone mailing/phone list system. Highlights are: quick sort; string search; 10 selection fields; flexible specification of selection codes for printing and logical ANDing between selection fields; operates on a single disk drive; prints one or two addresses across; flexible salutation specification; all fields fully edited.

Price: $85.00 Canada, $70.00 U.S. includes 13- and 16-sector versions plus 30-page user manual

Author: Managematics Ltd.

Available:
Evolution Software Inc.
1632 Bathurst Street
Toronto, Ontario M5P 3J5
(416) 787-3441

Name: **Pascal File Exchange (PFX)**
System: Apple II
Memory: 48K RAM
Language: Pascal
Hardware: Apple II, Language Card, 2 disk drives, Micromodem II or Coupler and Apple COM Card

Description: *PFX* is a Pascal File Transfer program with a novel feature — it transmits a copy of itself to another Apple even though initially the other Apple does not have any Pascal software to receive data from its modem. Thereafter, the operators may type messages in a "chat" mode, inspect the local and remote directories, schedule and exchange one or more files and initiate the execution of local and remote Pascal code modules.

Price: $45.00 includes disk with executable Code Files and formatted Documentation File

Author: Graeme Scott

Available:
Arrow Micro Software
11 Kingsford
Kanata, Ontario, Canada
K2K 1T5
(613) 592-4609

Name: **Multi-Tasking Kernel**
System: Any
Memory: 100 bytes
Language: Source Assembly
Hardware: 8085, Z-80, 6502, 6809, 6800

Description: The *Multi-Tasking Kernel* is a valuable tool for systems integrators to develop multiple real-time software tasks in a microprocessor-based product. It is a ROMable product which efficiently oversees both the selection and execution of each task. The kernel is fast, small, and easy to use. The *Multi-Tasking Kernel* is completely documented, fully tested, and available in source assembly form for the 8085, Z-80, 6502, 6800, and 6809.

Price: $195.00 includes assembly source code implementations for all five microprocessors.

Available:
U.S. Software
5470 N.W. Innisbrook Place
Portland, OR 97229
(503) 645-5043

Name: **Type**
System: SDOS or SDOS/MT
Memory: 48K minimum
Hardware: 6800/6809 CPU with CRT, disk and printer

Description: *Type* is a document-formatting program, used in word-processing or document production. Commands embedded in raw text files processed by *Type* control the formatting of that text on the output device. Output formatting includes full justification, page width and depth, page numbering, centering, spacing, titles and table of contents generation. *Type* is used in conjunction with the SD screen editor for easy data entry.

Price: $140.00 includes program, 100-page manual

Author: AMS

Available:
Software Dynamics (exclusively)
2111 W. Crescent, Su. G
Anaheim, CA 92801
(714) 635-4760

Name: **DOS/65 Version 1.2**
System: 6502
Memory: At least 16K
Language: Machine
Hardware: 8" or 5" "IBM Compatible" disk

Description: *Version 1.2 of DOS/65* is available for either 8" SSSD disks or 5" SSSD disks. It can be used with double density or double-sided disks and allows the user to specify the disk format. Included with the system is an editor, assembler, debugger, a full feature BASIC (BASIC-E/65), and several transient programs such as copy routines, file transfer routines and similar programs. BASIC-E/65 provides full disk I/O capability for random and sequential files and provides the usual arithmetic and string functions and statements.

Price: $125 to $175 depending upon customizing requirements. Includes disk and 200-page documentation package.

Author: Richard A. Leary

Available:
Richard A. Leary
1363 Nathan Hale Drive
Phoenixville, PA 19460

Name: **68000 Cross Assembler**
System: 6809 FLEX™ or UniFLEX™ System
Memory: 56K
Language: Assembler
Hardware: Any supporting 6809 FLEX or UniFLEX

Description: A full 68000 assembler which executes on a 6809. Accepts all standard Motorola instruction mnemonics with the exception of certain "suffix variations" to some root mnemonics. All expressions are evaluated to 32 bits. Numerous directives permit page formatting, symbol table listing, line numbering, command line parameters, file inclusion, etc. Macros and conditional assembly supported. Outputs S1/S2/S8/S9 records of ASCII hex data.

Price: $250 FLEX;
$300 UniFLEX
includes manual and diskette (manual assumes user is familiar with standard 68000 instruction set).

Available:
Technical Systems Consultants, Inc.
P.O. Box 2570
West Lafayette, IN 47906

(Continued)

Software Catalog *(continued)*

Name: **Morse Code Trainer**
System: Commodore VIC 20
Memory: 5K
Language: BASIC
Hardware: VIC 20 and tape player

Description: Practice Morse Code from 1-35 wpm rate sent by the VIC. Rate is controlled by your 60 Hz power line frequency. Enter your own practice message, or receive random letters, numbers, punctuation.

Price: $19.95 includes cassette and instructions

Author: Marion H. Taylor

Available:
Taylormade Software
8053 E. Avon Lane
Lincoln, NE 68505
(402) 464-9051

Name: **The Responsibility Life Dynamic**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II, Disk II

Description: This disk centers upon the following four games: *Speedway* (you'll have to be a great driver to make those hairpin curves); *Bean Reactions* (for two players who react to the impolite bean-reactions of each other with big clubs); *Ring the Bell* (carnival type game); and *Animal Bingo* (move 50 shape-table animals around on a Hi-Res playing board in totally unique ways to make "bingos" — five in a row).

Price: $15.95 includes disk, vocabulary card

Available:
Avant-Garde Creations
P.O. Box 30161 MCC
Eugene, OR 97403

Name: **Graphics Composer**
System: Atari 400/800
Memory: 32K RAM
Hardware: Cassette Player or Disk Drive

Description: With *Graphics Composer*, you use paddles or joystick to draw a picture outline on Hi-Res screen Mode 8 or 7. Then use color fill-in, color brushes and add text to complete your graphic designs. *Graphics Composer* allows easy creation of Player/Missile shapes which may be used in other programs. The Geometrics Figures program lets the user define circles, triangles, polygons, parallelograms, and even trigonometric curves. Loading routines are provided so that pictures can be used in other programs or traded with friends.

Price: $39.95 includes cassette or diskette and 27-page instruction booklet

Available:
Versa Computing, Inc.
3541 Old Conejo Rd.
Suite 104
Newbury Park, CA 91320
(805) 498-1956

Name: **Dentistaid**
System: Apple II
Memory: 48K
Language: CBASIC 2 with CP/M
Hardware: Apple II Microsoft Z-80 Softcard

Description: *Dentistaid* is a revolutionary new concept in dental office management systems. It is designed to streamline all major time-consuming tasks in your office and give efficiency, accuracy, and complete control of your office.

Price: $1,000.00

Author: Jerry Taylor

Available:
The Hayden Book Company, Incorporated
50 Essex Street
Rochelle Park, NJ 07662
(800) 631-0856

Name: **Tiny BASIC Compiler**
System: PET/CBM
Memory: 4K
Language: BASIC
Hardware: PET with cassette or diskette

Description: This is a floating point compiler supporting a subset of the PET BASIC language. The compiler reads your BASIC program and writes out a file containing the 6502 object code. All floating point arithmetic and functions are supported. If you have at least 16K, you can get a full assembly listing of the object code.

Price: $25.00 includes versions for all ROMs and sample program

Author: Mark Zimmermann and Dave Malmberg

Available:
Abacus Software
P.O Box 7211
Grand Rapids, MI 49510
(616)241-5510

Software Catalog *(continued)*

Name: **DIFF E-Q**
System: Apple II, Pascal language card
Memory: 64K
Language: Pascal
Hardware: One disk drive
Description: *DIFF E-Q* is a Pascal-based differential equation package for Apple II computers designed for use by engineers, scientists, mathematicians, college instructors, and students. It has high-resolution color graphics capabilities, a high-resolution screen editor, and electronic "Slidetray" and "Slideshow" features, making it ideal for lecture demonstrations and for group presentations. *DIFF E-Q* unlocks the door to a whole world of scientific adventure.

Price: $100.00 includes two diskettes, 40-page manual, and limited warranty.

Author: Mark Davidson

Available:
Sage Software Company
1322 La Loma Avenue
Berkeley, CA 94708

Name: **Plotting Graphs for Line Printer #26009**
System: Apple II or Apple II Plus
Memory: 32K RAM
Language: Applesoft
Hardware: Printer
Description: Where a line printer is available, these three programs will provide a hard copy of a particular graph, either for inclusion in a report or for later comparison with other results. The programs contained in this package are complete and require no additional programming. The following programs are included: *Cartesian Plots, Semi-Logarithmic Plots, Polar Plots*.

Price: $39.95

Available:
Advanced Operating Systems
450 St. John Road
Suite 792
Michigan City, IN 46360
(219) 879-4693

Name: **Chord Mania**
System: Apple II or Apple II Plus, DOS 3.2 or 3.3
Memory: 48K
Language: Extended BASIC
Hardware: Disk Drive, MMI DAC board
Description: A program designed with a game context for practice of chords, including recognition of four-voice chords in any combination of chord qualities (all triads and five different seventh chords) and inversions. Includes both aural and visual skills. Responses require the use of only three keys. Beginning to advanced levels.

Price: $190.00 includes disk and user's guide

Available:
Micro Music, Inc.
P.O. Box 386
Normal, IL 61761

Name: **Farm Ledger**
System: Apple II or Apple II Plus, DOS 3.3
Memory: 48K
Language: Applesoft
Hardware: 2 disk drives, printer
Description: With *Farm Ledger* the user can define up to 500 accounts and format financial reports. The general farm chart of accounts can be added to or modified to conform to a specific farming operation. Features include budgeting, departmentalizing, thorough audit trails, extensive error checking and data entry prompting. Detailed, non-technical documentation includes a practice session and glossary, SBCS provides free program updates, replacement of damaged disks, and friendly customer service.

Price: $349.00 includes program disk, program backup data disk, documentation.

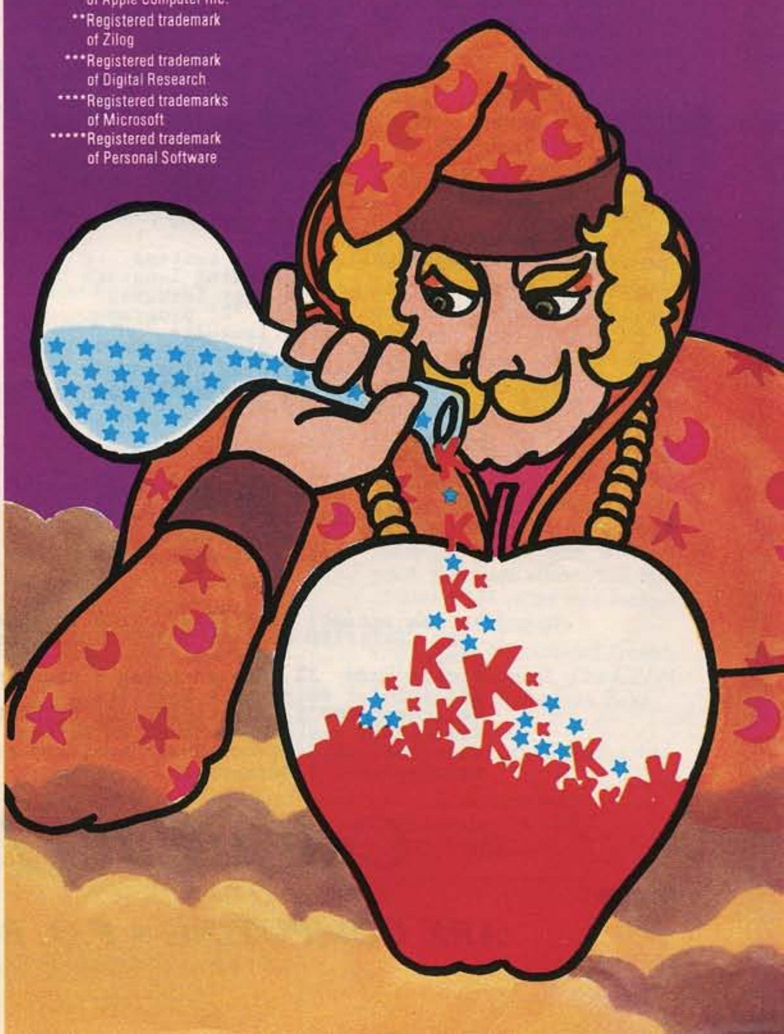
Author: David McFarling

Available:
Small Business Computer Systems
4140 Greenwood
Lincoln, NE 68504
(402) 467-1878

MICRO

Answer to Border Puzzle: "It's Really Very Easy!"

Answer to Circuit Puzzle: All 1's → 1, all 0's → 0, 13 of 16 possible combinations yield 0.



MICRO Hardware Catalog

Name: **GMS6521 65K ROM Module**
System: 6500/6800
Memory: 65K bytes
Hardware: 6" × 9.75" module
Description: Low cost, high density memory expansion module with 65,536 × 8 bits of EPROM/PROM/RAM with selectable address and enable/disable switches. Accepts 2758, 2408, 2716, 2516, 2732, 2532 EPROMs of 1, 2, 4K bytes each. Sixteen sockets may be individually enabled/disabled from top of module. Requires only +5V DC power. Low profile or zero force sockets optional.
Price: $230.00, single piece quantity
Available:
General Micro Systems
1320 Chaffey Ct.
Ontario, CA 91762
(714) 621-7532

Name: **GP300 Dot Matrix Printer**
Memory: 380-Char FIFO, optional 32K RAM for character generation
Description: 120-character (10 characters/inch) dot matrix printer. Capable of producing 9 × 9 data text at 300 characters/second and word processing printing at 80 - 120 characters/second (depending on font), 18 wire (9 × 9 interlaced) print head. RS-232C/Centronics interface. Options: tractor feed, front feed, reem paper handler and pedestal.
Price: $3165.00 one piece
$2685.00 100 pieces
includes 2 character generations, interface, power supply, friction feed.
Available:
Amperex Electronics Corp.
230 Duffy Ave.
Hicksville, NY 11802
(516) 931-6200

Name: **Display Board**
System: Apple
Description: Displays address bus, data bus (latched) and hold line. All lines are buffered. All LEDs are low current, high efficiency type. A RUN STOP STEP switch is provided so you can single step through a program one instruction at a time.
Price: $62.00 (Texas residents add 5% sales tax)
Available:
Applied Engineering
P.O. Box 470301
Dallas, TX 75247
(214) 492-2027

Name: **Hi-Pad Digitizer**
System: TRS-80 1/2/3, Apple, Atari, PET
Memory: 16K
Hardware: Digitizing Pad
Description: High-Resolution, high-quality, but low-cost digitizing pad. Serial RS-232 or parallel interface. Excellent replacement for Apple pad. No static sensitivity. Optional stylus.
Price: $825 - $925
Available:
Houston Instruments
One Houston Square
Austin, TX 78753

Name: **EZASM**
System: PET/CBM
Language: Assembler
Hardware: New ROMs, 8K-32K, cassette or disk, 40- or 80-column screen
Description: 4K EPROM chip which is available for any free socket and contains a very complete 6502 Assembler. Source code is stored as if it were a BASIC program, which allows for easy entry, editing and manipulation: one BASIC line is one line for the Assembler. Syntax is the MOS Technology standard and all addressing modes are supported. Operands may be symbols, symbol expressions, decimal, hexadecimal, binary or ASCII with limitless combinations. There is an optional Cross-Reference which lists the symbols used, in alphabetical order, followed by their value and each line number where the symbol was used.
Price: $80.00 includes shipping and instructions
Author: Milton B. Bathurst
Available:
DataCap
73, Rue du Village
4545 Feneur
Belgium

Name: **CD2-3 Floppy Drive Tester**
System: OSI
Hardware: 5¼" and 8" single- or dual-sided disk
Description: *CD2-3* uses existing drive cable for quick connection to isolate problems, exercise and repair floppy drives. Provides static and dynamic tests. Simplifies head load and stepper tests and adjustments. Monitors index, ready, write protect, clock and data circuits. Provides read, write and erase verification.
Price: $275.00 includes step-by-step familiarization and operating instructions
Available:
TEACO/Computer Center
P.O. Box E
Michigan City, IN 46360

Name: **CBM 2031 (Single floppy disk)**
System: PET/CBM
Memory: 1K RAM
Description: Low cost, single disk drive stores up to 170K bytes on a single 5¼" floppy diskette, and incorporates an IEEE-488 interface for use with Commodore's PET and CBM computer equipment. The 2031 is based on the same technology used in Commodore's 4040 dual disk drive unit, using the latest disk operating system (DOS). The 2031 diskettes are read/write-compatible with the CBM 4040 disk unit. Owners of the disk drive can expand their systems by adding additional 2031's (or Commodore disk drives) and running them in tandem.
Price: $595.00 includes disk operating system
Available:
Commodore Business Machines, Inc.
Computer Systems Division Authorized Dealers

Name: **Model 60 Universal RS-232 to RS-449 Converter**
System: All Interfaces
Hardware: Aluminum box, featuring three I/O connector ports
Description: The *Model 60, Universal RS-232C to RS-449 Converter*, provides a means of interconnecting hardware with these interfaces. The RS-449 specification requires a 37-pin connector for reverse channel operation. Since the allowable voltage range of RS-232 signals exceed that of RS-449, it is necessary to provide resistive terminations to prevent damage to RS-449 receivers from RS-232 drivers. The *Model 60* incorporates the switching mechanism to allow the user to select the RS-232 as a DTE or DCE.
Price: $115.00 complete
Available:
Remark Datacom Inc.
4 Sycamore Drive
Woodbury, New York 11797
(516) 367-3806

Name: **TKC Numeric Keypad**
System: Apple II
Description: The Keyboard Company's *Numeric Keypad* for the Apple II computer allows rapid numeric entry, easy arithmetic calculations and more efficient VisiCalc™ operations. The 24-key keypad is Apple-coordinated and may be comfortably positioned for maximum effectiveness.
Price: $149.95 includes keypad, interface and manual
Available:
Authorized TKC/Apple Dealers
Information:
The Keyboard Company
7151 Patterson Dr.
Garden Grove, CA 92641
(714) 891-5831

Name: **Z-Card**
System: Apple II or Apple III
Hardware: Z-80A microprocessor, CP/M operating software
Description: The *Z-Card* transforms the Apple II or Apple III computer into a CP/M-based system. The *Z-Card* offers lowest power consumption and highest reliability at an affordable price. CP/M software and ALS BIOS, which are included, increase the speed of the system and allow the user to convert DOS text files to CP/M. Features: full keyboard mapping, warm boot without reset, software allows full 60K of user RAM, copy and format in one pass through.
Price: $269.00 includes Z-card, diskette, manual, *The CP/M Handbook* by Rodnay Zaks
Available:
Advanced Logic Systems, Inc. (ALS)
1195 E. Arques Ave.
Sunnyvale, CA 94086
(408) 730-0306

MICRO

PET Audible Disk Alarm

This article describes a simple accessory which sounds an alarm when a disk error occurs. The audible alarm saves time spent in tracing system errors.

John E. Girard
676 Alma Ave. #106
Oakland, California 94610

I don't know how much time I've wasted trying to salvage my work, only to discover that I was hung up on a simple disk error. Now the investment of $3.67 in parts will allow us to *hear* disk errors and to correct them in record time. My device will work on 2040, 4040, and 8050 series disk units. Installation takes about 10 minutes.

Parts List

Piezo buzzer, Radio Shack #273-060, $2.99; resistor, 470 ohms, #271-019, .19; diode, 50 volt @ 1 A, #276-1101, .49 (8050 only); 8-inch length of red hookup wire; 7-inch length of black hookup wire; electrical tape (masking tape will suffice).

Installation

Please refer to figure 1. Remove the two cover screws located on the sides near the front, swing the top section up and prop it open with the wire brace. Proceed with the following steps:

1. Strip ½ inch of insulation from the ends of all wires.

2. Attach the red hookup wire to the red wire of the piezo buzzer.

3. *8050 disk drives:* Observe that the diode is a black cylinder with a white band encircling one end. Attach the banded end to the black hookup wire. The other end of the diode connects to one side of the 470 ohm resistor. Attach the remaining lead from the resistor to the black wire of the piezo buzzer.

   2040 and 4040 disk drives: Attach one side of the 470 ohm resistor to the black hookup wire; attach the other side of the resistor to the black wire of the piezo buzzer.

4. Locate the power plug for the error LED. Pull the plug out and separate the wires slightly. Remove ¼ inch of insulation from each wire but do not cut the wires. Replace the plug.

5. If your plug wires are orange and white, connect the red alarm wire to the orange plug wire and connect the black alarm wire to the white plug wire. Now, create an error condition (an easy one is to initialize an empty drive). When the error light comes on, the piezo buzzer will emit a mild but penetrating tone.

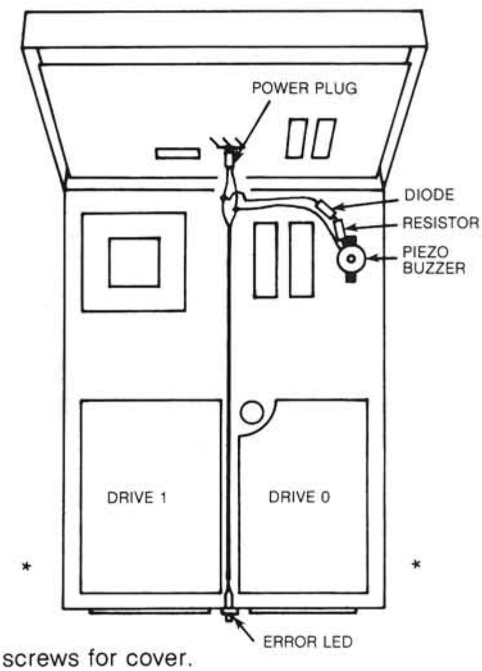
6. If your plug wires are not orange and white, then do not connect the alarm wires. Create an error condition. Once the LED is glowing red, try touching the alarm wires to the exposed plug wires. You have two choices: one combination will activate the buzzer, and the other will not. *8050 owners:* If neither combination works, you have wired the diode backwards. Return to step 3.

   When you have found the proper combination, clear the error and proceed to connect the alarm wires.

7. Cover all electrical connections with tape. Be sure to cover the bare wires on the diode and resistor as well. To mount the buzzer, select any open spot and secure it with two strips of tape. You may reduce the loudness by partially covering the buzzer with tape.

If your floppy is still under warranty. . .
. . . Do not strip the power wires in step 4. Wrap the alarm wires carefully around the plug prongs, push them to the base of the connector, and secure with narrow strips of tape before replacing the plug. Be sure to remove the alarm entirely before taking your floppy in for service!

***Figure 1:* Inside view**



* Locations of the mounting screws for cover.

MICRO

I/O Expansion for AIM

The AIM 65 computer is well-suited to low-demand process control applications, due to its user-dedicated 6522 VIA. This article describes a method of adding two more VIAs to the AIM to triple the capacity of its input/output control hardware.

Gary Finley
Room P-102
Biological Sciences
University of Alberta
Edmonton, Alberta
Canada T6G 2E9

Many users of 6502-based micro-computers who have an interest in control-type applications have discovered the wealth of hardware utilities that is provided in the 6522 versatile interface adaptor (VIA). This well-named device provides an impressive array of input/output (I/O) facilities inside one $10 package, including two 8-bit parallel ports, two 16-bit counter/timers, and an 8-bit shift register. The ports have bit-wise selectable data direction, optional input latching and four associated control (handshake) lines. The two timers can easily perform pulse counting, frequency generation and interval timing functions, and the shift register can be used to perform both serial-to-parallel and parallel-to-serial data conversions.

In the Rockwell AIM 65 computer, a 6522 chip is used to control the 20-column printer, the two cassette interfaces, and the teletype port. One of the timers in this chip is used to provide the five millisecond delay that is used to debounce the keyboard switches. The AIM 65 board holds a second 6522 VIA, and this one is available for user applications, with all 16 port bits and their four associated control lines brought out to the 44 contact 'applications' connector at the left rear of the computer board. It was the provision of this user VIA that made the AIM 65 computer attractive to my colleagues and me at the psychology department of the University of Alberta.

Many of the experiments in psychological research that are conducted here represent low demand control tasks to which the AIM 65 and its user VIA are admirably suited. In a typical stimulus-response experiment, a control computer is used to present an auditory or visual stimulus to the subject. The computer then waits for him to choose from a number of possible responses, and records as data the identity of the response selected and the time taken by the subject to make his choice.

These tasks are easily accomplished with the facilities of the user VIA and some simple peripheral hardware. A few port bits are configured as inputs and connected to debounced response switches, a few others act as outputs — controlling lamps or tone generators through Darlington transistor drivers, and the programmable timers look after the measurement of the reaction times. Thus, with a minimum of external hardware, the AIM 65 can provide an excellent control facility at very modest cost to the user.

In the early days of this work (early 1980) the available capacity of 16 port bits seemed easily adequate for the demands of the type of experiment then conducted, and for any reasonable demand then foreseen for the future.

How times change! In the intervening months, we have found that port space acts like spending money — given an adequate supply, one soon develops a need for much more. As our experimenters became familiar with the computer, their confidence in the technique grew, and their ambitions were right behind. Soon the AIM 65 was running very demanding experiments involving simultaneous inputs from several subjects, the operation of stepping motors (which consume four port bits each), or combinations of similar tasks. The wealth of port space that had once seemed so generous was soon all spent, and the next experiment in line needed "just a few more bits" for the control of the apparatus and collection of the data.

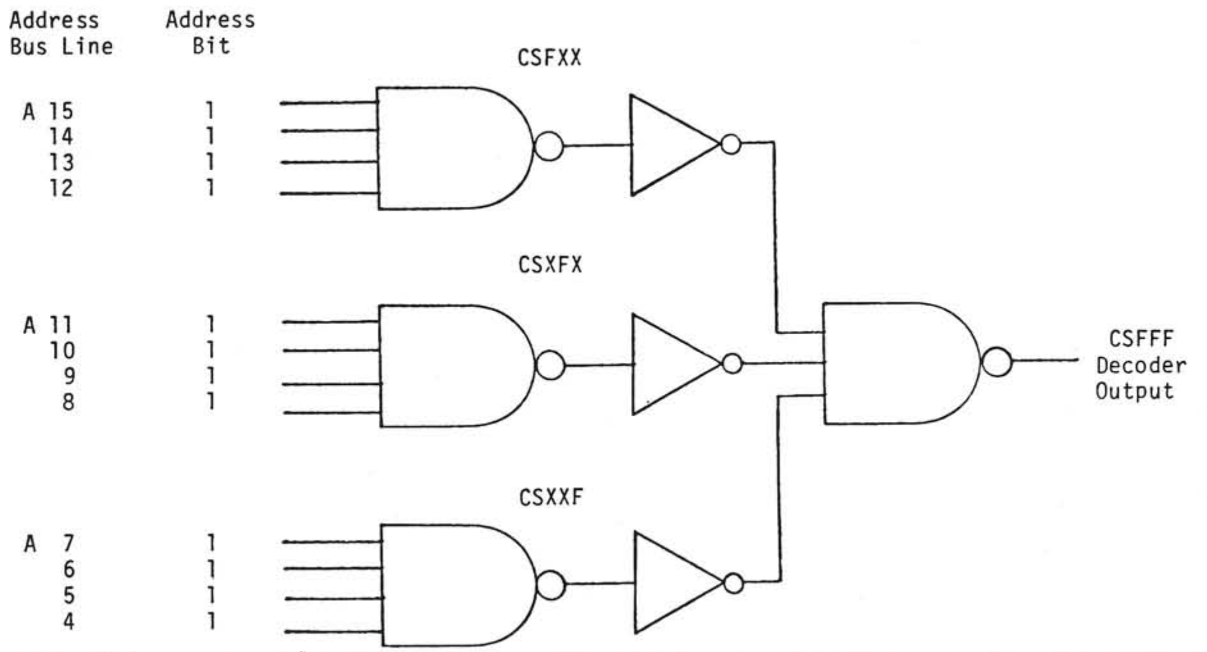
As these situations arose we met them at first with craft using tricks to make more efficient use of the available port space. By encoding 16 inputs into four bits, or multiplexing one set of four bits to run several motors at once, we made our 16 bits do the work of 24 or more. However, these stop-gap strategies were not very satisfactory. The hardware involved became complicated, and the programs became laden with extra routines containing the tricks that shared the port space among the various tasks to be done. Before too long it was clear that, although the AIM computer had plenty of computing power for our needs, the single user VIA was a bottleneck in the flow of control.

This article describes our solution to the problem: a simple add-on circuit for the AIM 65 that contains two additional 6522 VIAs. Using this design, any AIM 65 user who can do wire-wrap assembly can, in a few hours (at a cost of about $40), triple the I/O capacity of his computer to a total of six bidirectional ports, twelve control lines, three shift registers and six programmable timers.

Design of the I/O Expansion Board

To the 6502 microprocessor, a 6522 VIA looks like an array of 16 memory locations. Like any other memory chip, it must have connections to the system data bus and the control bus signals necessary for reading bytes of data from the data bus, or writing bytes to it.

Figure 1: A Decoder for Address $FFF



These control signals are: the system clock (Ø2), the read/write line (SYS R/$\overline{W}$), and the reset line ($\overline{RES}$). An additional connection is needed in order for the 6522 to be able to interrupt the microprocessor during interrupt-driven I/O functions. If this facility is desired, then each 6522 in the system must be connected to the interrupt request line ($\overline{IRQ}$).

In addition to the data bus and the above control signals, a 6522 VIA must have connections to the system address bus, so that its 16 internal registers can be accessed one-at-a-time for the exchange of control and data bytes with the microprocessor. With only 16 internal locations, the 6522 needs only to decode the lowest four bits of address information. Like any other memory chip, it relies on an external address decoder circuit to tell it which section of the address space it is to respond to. In this case, the section is only 16 addresses long. The design of such an address decoder is a central part of this project.

In general, the address decoder must contain the logic necessary to produce a chip select signal for each VIA when one unique combination of bits is present on the highest 12 lines of the address bus. In commercial products, the address decoder logic is usually very general so that the purchaser can adjust the decoder, by means of DIP switches or wire jumpers, to fit the address requirements of the product into an area of the memory map of his system that is not already occupied. This degree of generality is nice in principle, but it complicates the design of the decoder a little, increases the parts count, the cost of the project, and also adds to the construction time. In the design shown below, the decoder logic has been kept simple through the use of a fixed address assignment, chosen by the user before construction to suit the requirements of his system.

Address Decoder Theory of Operation

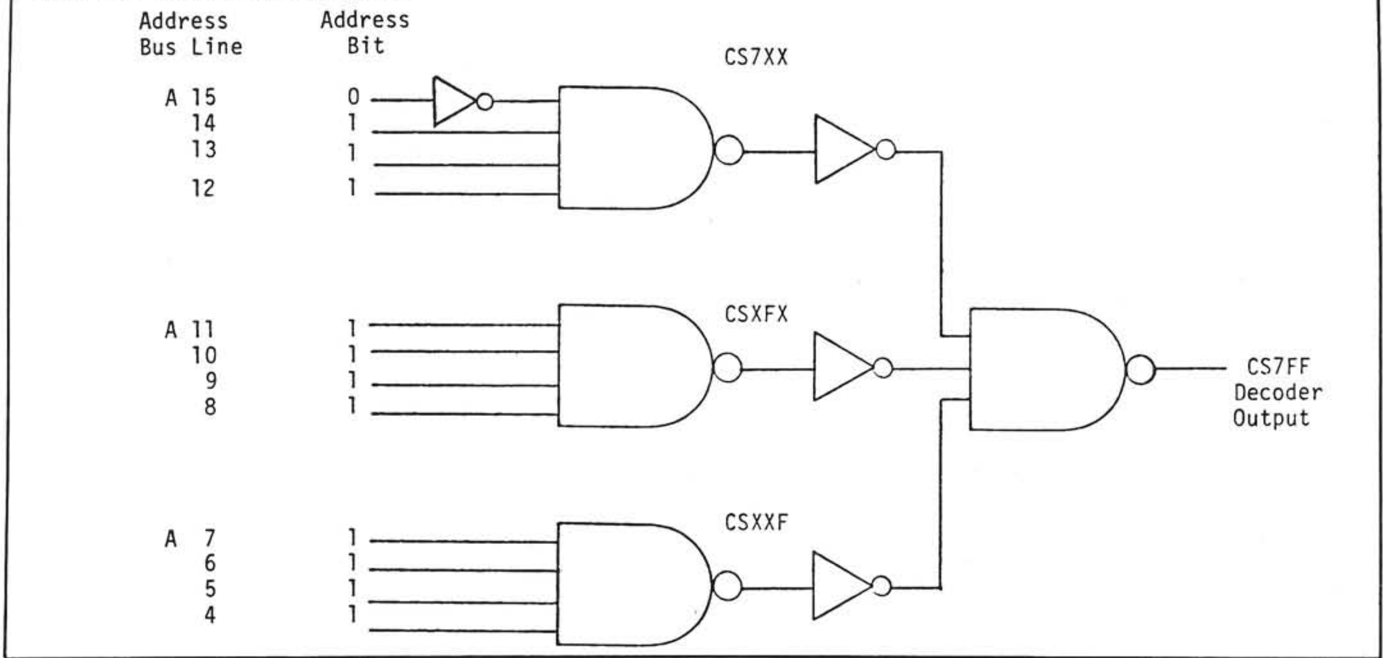
As an example of the operation of an address decoder, consider one built to produce a chip select signal for the unique 12-bit address $FFF (1111 1111 1111 binary). If the four address lines represented by each hexadecimal digit of the address are connected to the inputs of a four-input NAND gate (see figure 1), the gates will each produce logic LOW outputs only when all four of their inputs are in the logic HIGH state. If we form a second logical NAND function of the inverted outputs of these gates, the resulting output is a chip select signal that is LOW only when all 12 inputs are HIGH, corresponding to an input address of $FFF. Any other address will cause at least one of the four-input gates to have a HIGH output, and no chip select (LOW decoder output) will be produced.

This example decoder is of no use whatever in the AIM 65, since no addresses starting with $FFF are available to the user in that system. However, the decoder of figure 1 is easily changed into others that decode addresses that are available to AIM 65 users. If, in figure 1, the bit on the highest address line (A15) was changed from a one to a zero, the address would be $7FF (0111 1111 1111 binary). The decoder of figure 1 doesn't give an output for this address. However, by the introduction of a single inverter between the address line A15 and its NAND gate input, we can alter the decoder so that it does respond to this new address. As drawn in figure 2, the decoder gives an output only for the new address $7FF. From this example it is easy to generalize to the design of a decoder for any desired address in the range from $000 to $FFF. Simply write the address down in column form beside a diagram like figure 1, and wherever it contains a logic zero, put an inverter between that address line and the NAND gate input.

An Address Decoder for the I/O Expansion Board

An I/O expansion board containing two VIAs requires two address decoders of the type described above: one to produce a chip select signal for each VIA. However, by choosing the addresses of the VIAs appropriately, we can produce two signals with little more hardware than was needed for the first one. Since each VIA has 16 internal registers, two VIAs must have their base, or lowest addresses separated by at least 16 addresses to avoid overlap. This means that their hex addresses must differ by at least one number in the third hex digit. The component count of the decoder is minimized if both VIAs have common first and second hex address

Figure 2: **A Decoder for Address $7FF**



digits. In figure 3, the top four-input NAND gate decodes address bits A15 - A12, or the first hex digit, and gives an output when that digit is a '9.' Similarly, the second four-input gate decodes bits A11 - A8 (the second hex digit) and gives an output when that digit is an 'F.' By choosing both VIA addresses to be of the form $9FX (where X is any hex digit), we can use the above two signals in the generation of chip selects for both of them with no added hardware.

In most AIM 65 systems, there is a large range of address space that is available for use by the I/O expansion board. In a computer with 4K of on-board memory, the entire range from $1000 to $9FFF is available. In a machine having a 16K memory expansion board, this range is reduced to $5000 to $9FFF, but there is still plenty of room for the 32 addresses occupied by the I/O board. For this reason, it is usually possible to choose the I/O board addresses to be values convenient to the design of the decoder. In order to save wiring time, it is most convenient to choose the third hex digits of the addresses to use the fewest inverters in the decoder circuit. Since an 'F' uses no inverters, it is one logical choice. The other digit could be an 'E,' 'D,' 'B,' or a '7' using only one inverter. In the circuit shown in figure 3, this digit has been chosen to be an 'E,' because this way the two VIAs occupy 32 adjacent addresses, minimizing the address space between them which would otherwise probably be wasted.

Thus, in figure 3 we have the top four-input gate giving an output (logic zero) for 12-bit addresses of the form $9XX, and the second four-input gate giving an output for addresses of the form $XFX. The third four-input gate produces an output for addresses $XXF, and the bottom four-input gate for addresses $XXE. These signals all require inversion before they can be combined by the three-input gates on the right side of the diagram, so each is first passed through an inverter as shown in the middle. These inverters and the top three-input gate perform a logical AND function on the signals CS9XX (chip select from addresses of the form $9XX), CSXFX, and CSXXF to produce a chip select signal whenever the current address is of the form $9FF. The inverters and the bottom three-input gate do the same for the signals CS9XX, CSXFX and CSXXE to give a chip select signal for all addresses of the form $9FE.

When these two chip select signals are connected to the two VIAs on the I/O expansion board, those VIAs will respond in exactly the same way as the built-in user VIA, but with base addresses $9FE0 and $9FF0. For example, in the VIA chip the interrupt flag register has the location (base address + D) hexadecimal. The user VIA has a base address of $A000, so the user interrupt flag register is at the address $A00D. This same register is at the address $9FED in one of the added VIAs and $9FFD in the other.

The above base addresses were not chosen entirely to simplify the wiring of the address decoder. In the AIM 65 the system I/O addresses start at $A000, and in machines with both the ROM-based assembler and BASIC language installed, the system memory map is full all the way up to the top of the address space at $FFFF. The system address usage is put at the top of the address space to keep it out of the way of user memory which starts near the bottom (at $0200). In order to leave the maximum amount of uncluttered address space for user memory expansion, it is good policy to follow the example of the AIM's designers and put the expansion I/O board address usage as high in the available address space as possible. The address decoder in figure 3 puts the two added VIAs in the last 32 addresses below the system's reserved area.

In systems that have this area already dedicated to some device, the decoder of figure 3 can easily be changed to put the added VIAs somewhere else. Two hex-inverter packages (74LS04) were used for the design shown in figure 3. Of these 12 inverters only the four in the middle of the diagram (between the sets of four-input and three-input NAND gates) are in required positions. The remaining eight inverters (of which only three are used by the circuit in figure 3) are available for the selection of the address lines that are to contain logic zeros in the desired addresses.

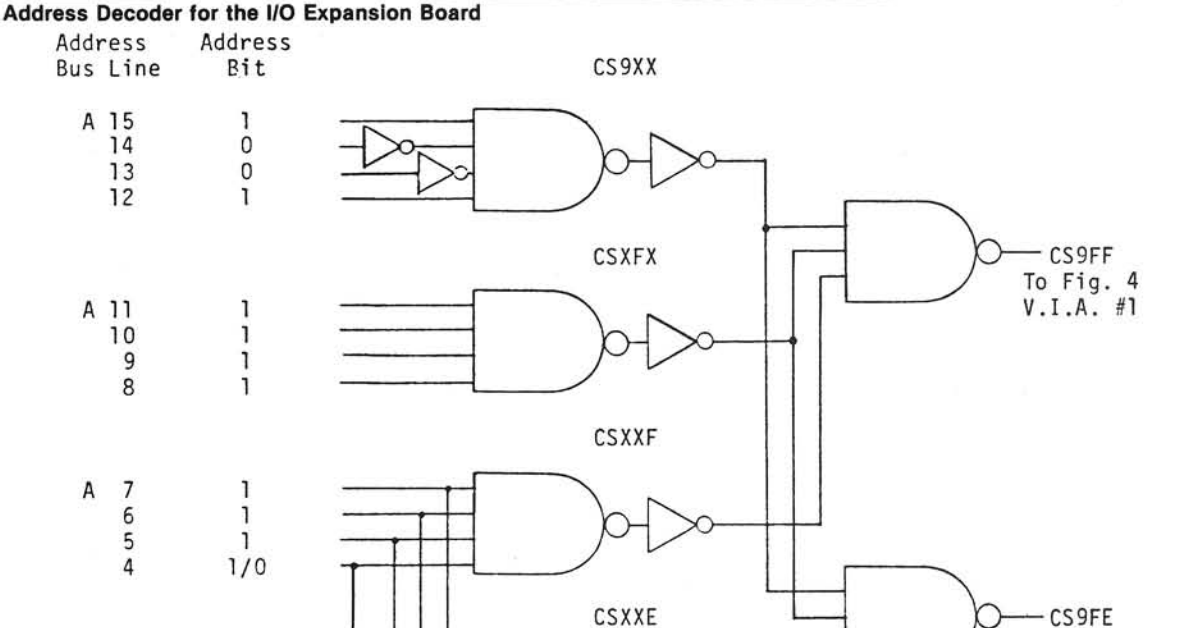
Figure 4 shows the circuit for the connection of the system data, address and control bus signals to the two VIAs. One of the chip select signals from the address decoder is connected to pin 23 of each one. The port bits and control lines on the right side of each package are connected to the peripheral devices as described in the assembly section below.

Once the address allocations of the I/O expansion board have been chosen, and an address decoder of the above form has been designed, the conceptual part of the project is finished. All that remains to be done is the actual wiring and the connection of the finished I/O expansion board to the computer.

Assembly of the I/O Board

A small single-unit project of this sort is most easily and quickly built by the wire-wrap technique, and this connection scheme is particularly well-suited to this project with its hard-wired address assignment. If some future expansion of the computer should require the changing of the I/O board addresses, this can easily be done by the re-wrapping of a small number of the wires which determine the address bus lines that are inverted before connection to the NAND gate inputs. This

***Figure 3:* An Address Decoder for the I/O Expansion Board**



sort of change would be inconvenient if a circuit board had been etched to suit the initial choice of addresses.

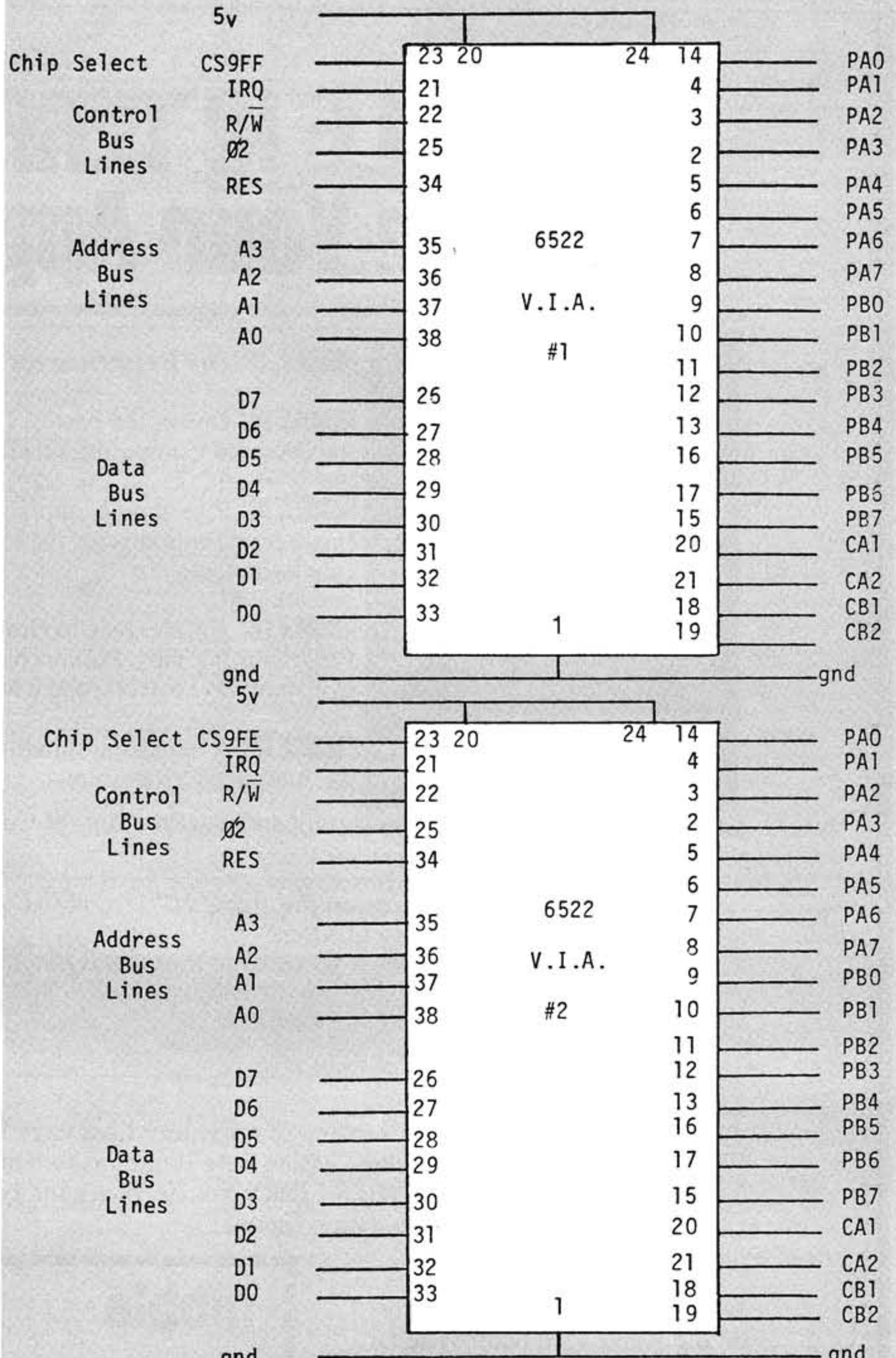
In the psychology department shop, we build these I/O expansion boards on a perforated prototyping board. Radio Shack stores sell one of these (part number 276-152) that has a 44-contact edge connector identical to the AIM 65's 'expansion' connector. By wiring the address, data, and control bus signals to the appropriate contacts when the I/O expansion board is being built (following the assignments on the AIM 65 connector), the board can be made to connect directly to the AIM 65 expansion connector with a simple parallel-bus connector made from two 44-contact edge connectors (like the EDAC 307-044-500-202) and a short length of ribbon cable. This cable should be kept shorter than six inches, or some degradation of the unbuffered bus signals may occur.

Many of the AIM 65 systems in use here are built around a five-slot card cage made for the KIM/SYM/AIM bus by Microtechnology Unlimited. The bus connector of this card cage is designed for the memory expansion, video display and disk controller boards made for these computers by this company. The I/O expansion board fits handily into one of the slots in these systems, saving the rather tedious fabrication of a cable-type connector.

A second connector is required to carry the port bits and control lines to the devices that interface with the VIAs on the I/O expansion board. For this purpose we use a 50-contact wire-wrap-style ribbon cable connector, such as the ALPHA FCC-152-50. The 32-port bits and eight control lines from the two VIAs are wrapped to contacts on this connector, which mounts into a row of holes near the free edge of the prototype board. The loads or data inputs are connected to a ribbon cable terminated with a mating connector (ALPHA FCC-120-50) which can be plugged into the one on the I/O expansion board.

Several of our AIM 65 systems use an I/O card cage of local design which has this ribbon cable crossing the back of the chassis as a port-bit bus. Cards plugged into this chassis mate with card-edge connectors (ALPHA FCC-171-50) that are pressed onto the ribbon cable. Some of these cards carry load drivers like power Darlington transistors or optically-coupled triacs for the control of external devices. Other

***Figure 4:* VIA Connections**



cards contain switch debouncing circuits for the input of switch-closure response data, or custom devices that require connection to I/O port space or control lines. With the advantages of expanded I/O capacity and modularly interchangeable I/O hardware, our AIM 65 computers have become powerful and flexible experiment controllers.

Conclusions

The I/O expansion project described in this article was undertaken as a simple, quick and inexpensive solution to the demand for increased I/O capacity. Some care must be taken by ambitious readers who desire a much more substantial addition to the I/O capabilities of their computers. The device described above should not be significantly expanded without some redesigning of the circuits.

Gary Finley is a member of the staff of the Psychology Department of the University of Alberta. He works in microcomputer software development and the design of custom peripheral hardware for both microcomputers and minicomputers.

MICRO

"NIBBLE® IS TERRIFIC" (For Your Apple)

NIBBLE IS: *The Reference for Apple computing!*

NIBBLE IS: One of the Fastest Growing Magazines in the Personal Computing Field.

NIBBLE IS: Providing Comprehensive, Useful and Instructive Programs for the Home, Small Business, and Entertainment.

NIBBLE IS: A Reference to Graphics, Games, Systems Programming Tips, Product News and Reviews, Hardware Construction Projects, and a host of other features.

NIBBLE IS: A magazine suitable for both the Beginner and the Advanced Programmer.

Each issue of NIBBLE features significant new Programs of Commercial Quality. Here's what some of our Readers say:

- ***"Certainly the best magazine on the Apple II"***
- ***"Programs remarkably easy to enter"***
- ***"Stimulating and Informative; So much so that this is the first computer magazine I've subscribed to!"***
- ***"Impressed with the quality and content."***
- ***"NIBBLE IS TERRIFIC!"***

In coming issues, look for:

☐ Stocks and Commodities Charting ☐ Assembly Language Programming Column
☐ Pascal Programming Column ☐ Data Base Programs for Home and Business
☐ Personal Investment Analysis ☐ Electronic Secretary for Time Management
☐ The GIZMO Business Simulation Game

And many many more!

NIBBLE is focused completely on the Apple Computer systems.

Buy NIBBLE through your local Apple Dealer or subscribe now with the coupon below.

Try a NIBBLE!

nibble

We accept Master Charge & Visa

Box 325, Lincoln, MA. 01773 (617) 259-9710

I'll try nibble!

Enclosed is my $19.95 (for 8 issues) Price effective Jan. 1, 1982

(Outside U.S., see special note on this page.)

☐ **check** ☐ **money order**

Your subscription will begin with the next issue published after receipt of your check/money order.

Card # ______________ Expires ________

Signature ______________

Name ______________

Address ______________

City ______________

State ______________ Zip ________

NOTE:

- Domestic U.S. First Class subscription rate is $36.50
- Canada Air Mail subscription rate is $42.50
- Outside the U.S. and Canada Air mail subscription rate is $47.50

All payments must be in U.S. funds drawn on a U.S. bank.

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

6502 Bibliography: Part XL

1200. A.N.A.L.O.G. Magazine No. 1 (Jan/Feb, 1981)

Bachand, Charles, "Parlez-vous Pascal?", pg. 6-8.
Pascal and other languages for the Atari.

Walter, Russ, "Graphically Speaking...," pg. 28-31.
A graphics tutorial for Atari users.

1201. Apple Assembly Line 1, Issue 8 (May, 1981)

Sander-Cederlof, Bob, "Hi-Res SCRN Function for Applesoft," pg. 2-3.
How to supply the HSCRN(X,Y) function corresponding to the Lo-Res SCRN(X,Y) on the Apple.

Anon., "Shift Operations in 6502 Assembly Language," pg. 6-10.
A tutorial on LSR, ROL, ROR, etc. op-codes.

Sander-Cederlof, Bob, "Commented Listing of DOS 3.2.1," pg. 12-20.
Covering $B800-BCFF of the Apple DOS 3.2.1.

1202. Abacus II 3, Issue 4 (April, 1981)

Yee, David R., "C.A.I. Programs: Binomial to the Nth Degree; Binomial Certain Term," pg. 4-5.
Useful Apple programs for the math teacher.

Avelar, Ed, "Some VisiCalc Applications," pg. 6-17.
Some notes and discussion of applications of VisiCalc on the Apple.

1203. A.N.A.L.O.G. Magazine No. 2 (March/April, 1981)

La Ferla, Robert, "What the PEEK and POKE Are You Talking About?" pg. 5.
A short instructional article on the PEEK and POKE functions of the Atari microcomputer.

Stewart, Ed, "Unleash the Power of Atari's CPU," pg. 20.
How to get an increase of 30% in speed from your Atari CPU.

Anon., "Flag Graphics Demo," pg. 22.
A short program in Atari graphics.

Repstad, Tom, "The Game Room," pg. 30-32.
An Atari tutorial on game programming.

Hartman, Robert and Pappas, Lee, "Download Terminal," pg. 36.
Store Compuserve, Source or Bulletin Board in RAM and off line send it to disk or printer.

Hoffman, Paul, "Machine Language to BASIC Conversion," pg. 43.
A program to read a portion of Atari memory and then write the contents of the screen to BASIC Data Statements.

1204. The Target (January/February, 1981)

Clem, Donald, "Memory-ASCII Dump," pg. 3-4.
Program to provide a memory dump of a user-specified AIM 65 memory range.

Wahlquist, John, "CB-2 Sound," pg. 5.
Hardware and software for a sound generation technique on the AIM 65.

Riley, Ron, "Riley Corner," pg. 7.
Schematic and parts list for a power supply for the AIM 65.

1205. ByteLines 1, No. 7 (May, 1981)

Anon., "Patch to Renumber," pg. 5.
A patch to overcome a slight bug in the renumbering program on the DOS 3.3 Master Disk.

1206. Mini'app'les 4, No. 5 (May, 1981)

Espinosa, Chris and Wyman, Paul, "PEEKs, POKEs and CALLs," pg. 8-9.
A table useful to Apple programmers.

1207. The Apple-Dillo (May 1981)

Sethre, Tom, "TRACE Debugging with DOS," pg. 13.
A solution to the problem of using TRACE to test a program that performs disk I/O.

1208. From The Core (May, 1981)

Anon., "Renumber," pg. 2
A patch for the bug in the Renumber programs in either DOS 3.2 or 3.3.

Malpass, Mike, "MX-80 Block Graphics Characters," pg. 4.
A hardware mod for the Epson/Apple interface card to permit printing block graphics characters.

1209. The Apple Barrel 4, No. 3 (May, 1981)

Pardue, G.H., "Modifications of File Cabinet V," pg. 5-6.
Changes to operate with the Epson MX-80 printer.

Kramer, Mike, "Epson MX-80 Horizontal Tab Problem," pg. 7.
A note on efficient use of the Epson MX-80 printer with the Apple.

1210. PEEK(65) 2, No. 5 (May, 1981)

Cook, Willis H., "Graphics Utility for OS-65D," pg. 2.
A utility for OSI users.

Peabody, Al, "The Beginning Assembler," pg. 2.
The first in a continuing series of instructional articles on machine language.

Peabody, Al, "Indirect Files for Program Transfer," pg. 7.
A tutorial on OSI files.

Shingara, T.M., "A Real Time Clock Program," pg. 18.
A clock program for the C1P-SBII.

1211. Compute! 3, No. 5, Issue 12 (May, 1981)

Lock, Robert, "Introducing 'Super PET'," pg. 4-8.
A new CBM micro has 134K mixed RAM and ROM with both 6502 and 6809 processors and separate ROM operating systems and several languages.

Davis, Tony and DeJong, Marvin, "Using the AIM 65 as a Remote Terminal for an Apple," pg. 42.
A telephone link with an AIM, a Novation CAT modem and an Apple on the other end.

DeJong, Marvin L., "A Floating Point Multiplication Routine," pg. 52.
A machine language routine for 6502 micros.

Hartman, Tony A., "Naming Compounds," pg. 58-63.
A PET CAI program for Chemistry students.

Smith, M.R., "Using Named GOSUB and GOTO Statements in Applesoft BASIC," pg. 64-68.
A routine to allow named subroutines in Applesoft as is possible in Apple Integer BASIC.

Boom, Michael, "Using Strings for Graphics Storage," pg. 82-84.
A method of Atari graphics generation using text strings to store pixel data.

Brannon, Charles, "Atari Machine I/O," pg. 84.
Three routines of interest to Atari machine language programmers.

Lindsay, Len, "Disk Directory Printer," pg. 86.
Two program listing routines for the Atari disk.

Bills, Richard, "Real-Time Clock on the Atari," pg. 88-89.
Utilization of the real-time hardware available on the Atari.

Butterfield, Jim, "Getting to the Machine Language Program," pg. 112-113.
Instructional article for PET users.

Hudson, Arthur C., "A Thirteen Line BASIC Delete," pg. 116.
A short program, entirely in BASIC, which allows you to delete any group of lines from an existing program on the PET.

Brandon, Eric, "Using the Hardware Interrupt Vector on the PET," pg. 126-127.
A utility for the PET.

Zumchak, Gene, "Build Your Own Controllers."
Notes on the design of single board computers, development systems, emulators, etc., based on 6502 systems.

1212. Call —A.P.P.L.E. 4, No. 4 (May, 1981)

Mosher, Paul W. and Howard, Clifton M., "A Catalog Label Printing Program," pg. 9-14.

Walker, David W., "Super Apple BASIC," pg. 17-18.
A review of a new language for the Apple.

Greenberg, Harvey, "In the Depths of the Pascal Directory," pg. 27-28.
Exploration of Pascal.

Hilger, Jim and Golding, Val J., "Converting Binary Files to Data Statements," pg. 47.
A utility for the Apple.

Golding, Val J., "DOS Version Finder," pg. 53.
This routine identifies DOS and master/slave type of disk.

Zant, Robert F., "Alchemy and Other Wonders," pg. 61-62.
A program to find file record length with random organization, on Apple disks.

Berg, Arnie, "Basic 'Val' Function in Pascal," pg. 66.
How to incorporate the VAL( ) function in Pascal.

1213. The Michigan APPLE-gram 3, No. 6 (June, 1981)

Rivers, Jerry, "INSTR$ Function in Applesoft Revisited," pg. 4.
A fix for a minor bug inproves this program.

Anon., "IAC ApNote: Modifications Needed to Operate the Apple Communications Card with a 9600 Baud Terminal,'" pg. 22-23.
A hardware mod for the Apple card.

Anon., "IAC ApNote: Apple II ROM Monitor Subroutines List," pg. 24-28.
Table giving name, purpose, entry conditions, uses and example.

1214. KB Microcomputing 5, No. 6, Issue 54 (June, 1981)

Brieger, G.R., "Plan Your Retirement on Easy Street," pg. 50-53.
A VisiCalc program for the Apple, PET, Atari, etc.

Carr, Ronald W., "OSI Baud Mod," pg. 56-57.
Double and quadruple your cassette storage speed with this circuit for the OSI Superboard and C1P.

Cohen, Geoff A., "Double-Good OSI Protection," pg. 96.
With this circuit you must hit the BREAK key twice.

Lancaster, Don, "To and Fro with Apple's Inverted Decimal Code," pg. 98-101.
A program to calculate Apple calls above 32767.

Scanlon, Leo J., "Multiplying by 1's and 0's," pg. 110-120.
Try binary multiplication in 6502 assembly language.

Macauley, Colin, "Dump It On Cassette," pg. 130-131.
An approach to saving OSI machine-language programs on tape.

1215. Apple Bits 3, No. 2 (March, 1981)

Kovalik, Dan, "Taking the Mystery and Magic out of Machine Language," pg. 3-6.
An Apple utility "Integer Basic String" to replace strings.

1216. Apple Gram 3, No. 4 (April, 1981)

Meador, Lee, "Word Processors," pg. 2-4.
A discussion of features of various available Apple word processor programs.

Firth, Mike, "Definition 'Garbage Collection'," pg. 17.
An explanation of what Apple garbage collection is all about.

1217. Apple Bits 3, No. 4 (April, 1981)

Kovalik, Dan, "Taking the Mystery and Magic out of Machine Language."
Bell tone program for the NEO Apple Clock.

1218. Stems From Apple 4, Issue 4/5 (May, 1981)

Ward, Dennis, "DOS Commands During the Listing?" pg. 7.
It is possible to have DOS commands executed during the listing of your program.

Ward, Dennis, "The A.P.P.L.E. Educator," pg. 8-9.
Notes on using assemblers on the Apple.

1219. Byte 6, No. 6 (June, 1981)

Carlson, Keith and Haber, Steve, "Four Word Processors for the Apple II," pg. 176-204.
A review of several programs for the Apple.

1220. The Apple Peel 3, No. 6 (June, 1981)

Jenkins, Jerry, "Formatter Exec File," pg. 7.
A utility for the Apple.

1221. Purser's Magazine: Special Atari Edition (June, 1981)

Staff, "Program Reviews," pg. 10-44.
Forty-four pieces of Atari software are reviewed.

1222. Softwalk 1, No. 10 (June, 1981)

Wagner, Roger, "Assembly Lines," pg. 16-20.
Several routines in assembly language for the Apple.

1223. MICRO No. 37 (June, 1981)

Tripp, Robert M., "It's Time to Stop Dreaming," pg. 7-9.
A description of the features of the 6809 microprocessor.

Faguy, Francois, "A C1P Dump Utility," pg. 27-30.
A debugging tool for OSI machine language and BASIC programs.

Cain, Les, "Machine Language to DATA Statement Conversion," pg. 33.
A utility for 6502 users.

Huntress, Wes, "Serial Line Editor for the Apple," pg. 59-63.
An extended line editor allowing inserting, deleting, and several other features.

Tenny, Ralph, "Improved KIM Communication Capabilities," pg. 65-69.
A hardware/software combination to add new capabilities to the KIM.

Hill, Alan G., "Amper Search for the Apple," pg. 71-77.
High speed machine language search routine to find character strings in BASIC arrays.

Boness, Fred, "Memory Expansion for the Superboard," pg. 79-80.
A less expensive way to add memory to the OSI Superboard.

Girard, John E., "Horizontal Screen Scrolling on the CBM/PET," pg. 81-82.
Horizontal scrolling is a convenient method of displaying graphics functions that are too wide to fit on a PET screen.

Guilbeau, James, "AIM 65 RS-232 Interface," pg. 97-98.
A hardware article for the AIM users.

Mason, James L., "Real Time Clock for Superboard," pg. 99-101.
By providing a brief pulse once each second to the Superboard, and implementing this short program, the OSI computer will maintain and display real time in a background mode.

Dial, Wm. R., "Resource Update," pg. 102-104.
A list of publications which are good sources of information on the 6502 microprocessor and 6502-based micros.

MICRO

FOR YOUR APPLE II

Industry standard products at super saver discount prices

PARALLEL PRINTERS

C-ITOH 8510 PRINTER Virtually identical to the NEC 8023. Same electro-mechanical internals. The only difference is the name and the price. Specifications: • 100 CPS dot matrix printer • 80 column print – 136 characters per line • Tractor/friction feed • 7 different print fonts included • 2K printer buffer • Proportional spacing • Bit image graphics and graphic symbols.

NEC & C-ITOH List: $800

Special Price with any other purchase $595

EPSON 100 PRINTER $749

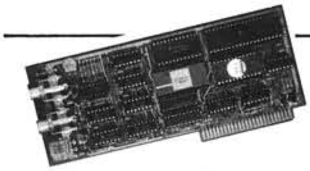
Z-80 CARDS FOR YOUR APPLE

MICROSOFT SOFTCARD. With CP/M® & MBASIC.
(List: $399) **$289**

NEW! **ADVANCED LOGIC SYSTEM Z-CARD. With C/PM®.**
Has everything the Softcard has except MBASIC. Works with Microsoft's disks too.
(List: $269) **$209**

NEW! **U-Z-80 PROCESSOR BOARD (from Europe)**
Software compatible with Softcard and
ALS Software .... (List: $199) **$149**
With CP/M®, ... add: (List: $75) **$49**

NEW! **ALS SYNERGIZER.** CP/M™ operating package with an 80 column video board, CP/M® interface, and 16K memory expansion for Apple II. Permits use of the full range of CP/M® software on Apple II *and* Apple III.
(List: $749) **$595**



80 COLUMN VIDEO DISPLAYS FOR APPLE II

SMARTERM.
(Not to be confused with SUPRTERM)

Software switching from 80 to 40 and 40 to 80 characters. 9 new characters not found on the Apple keyboard. Fully compatible with CP/M® and Apple PASCAL®. With lowest power consumption of only 2.5 watts.
(List: $360) **$279**

SMARTERM EXPANDED CHARACTER SET. Add to above: **$40**
7"x 11" matrix with true decenders.

VIDEX-80 .... $279
WIZARD $269

MONITORS FOR YOUR APPLE

AMDEX 300G $229

SUPER SPECIALS!

12" GREEN MONITOR $139
12" BLACK & WHITE MONITOR $125

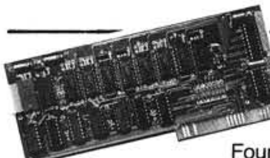
EXPAND YOUR APPLE II MEMORY

16K MEMORY EXPANSION MODULE. The preferred 16K RAM Expansion Module from PROMETHEUS. Fully compatible with CP/M® and Apple Pascal®. With full 1-year parts and labor warranty.
(List: $169) **$99**

MEMORY MANAGEMENT SYSTEM. Provides more free memory in BASIC by moving DOS into 16K card.
(List: $49) **$35**

16K RAM ADD-ON KITS. Eight 16K full specification RAMS.
Reduced to **$19**

SATURN SYSTEMS 32K RAM CARD $199



VERSAcard from PROMETHEUS

Four cards on one! With true simultaneous operation. Includes: (1) Serial Input/Output Interface, (2) Parallel Output Interface, (3) Precision Clock/Calendar, and (4) BSR Control. All on one card. Fully compatible with CP/M® and Apple Pascal®.
(List: $249) **$195**

JOYSTICK

Takes the place of two Apple Paddle Controllers. From BMP Enterprises. Heavy duty industrial construction and cable. Non-self centering. With polarity switches for consistent motion control.
(List: $59) **$39**

MODEMS FOR YOUR APPLE II

HAYS Smartmodem $229
NOVATION Apple Cat $349

CENTRONICS COMPATIBLE PARALLEL INTERFACE

NEW! Look out Grappler! From PROMETHEUS. For use with Epson, NEC, C-ITOH, and other printers. Fully compatible with CP/M® and Apple Pascal®.
Only **$69**
With Screen Dump Graphics for EPSON printer, add ... **$25**

Phone for Quick Shipment!
(415) 490-3420

SGC

"The mail order specialists"

342 Quartz Circle, Livermore, CA 94550

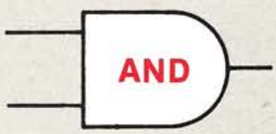
All equipment shipped factory fresh. Manufacturers' warranties included. Please add $3.00 per product for shipping and handling. California: add 6% tax; BART Counties: 6½%.

Boolean Algebra

Boolean Algebra is the science of True and False. If statement A is True and statement B is False, then the statement "A AND B" must be False, because one of the component statements is False. However, the statement "A OR B" would be True, since only one of the components needs to be True. In high level programming, a typical Boolean statement would be, "IF Q P and P=R−5 THEN...." Both of the component statements have to be True for the statements immediately after the THEN to be executed. If the AND is replaced with OR, then only one of the statements needs to be True.

Assembly language programming relies a great deal on Boolean Algebra, too. Each bit position in a binary number may contain only a 1 (=True) or a 0 (=False). When a Boolean operator acts on a pair of numbers, the values for bit 0 in each are compared to determine the value for bit 0 in the result, and so on for all the bits in the two numbers. The 6809 has a COMplement instruction, which is equivalent to the Boolean NOT, and both the 6502 and 6809 have AND, OR, and EOR instructions.

These functions also apply to digital circuitry. Circuit elements called "gates" compare two or more signals to arrive at a single resulting signal. Usually +5 VDC is considered "True" and 0 VDC is considered "False." So, if an OR gate receives 0 V on one input and +5 V on the other, the output is +5 V. An AND gate with the same inputs would produce a 0 V output.



Description: If both statements are True (bit=1), then the result is True. Otherwise, the result is False.

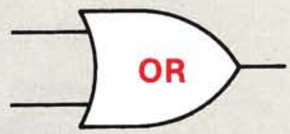
ML Application: ANDing a non-standard character code (such as for reverse field on the PET or Apple) with 0111 1111 will usually return the ASCII value.

Syntax: A AND B, A · B, AB
A × B
A ∧ B   A ∩ B

A	B	A OR B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

Example:
```
     1011 0110
AND  0111 0000
     ---------
     0011 0000
```



Description: Returns a True value if either statement or both statements are True (bit=1).

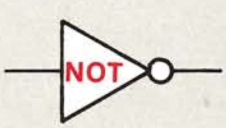
ML Application: Use to make sure a particular bit is set. ORing a character code with 1000 0000 will return the same code, except with bit 7 always set.

Syntax: A OR B
A + B
A V B,   A U B

A	B	A AND B
True	True	True
True	False	True
False	True	True
False	False	False

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Example:
```
     1011 0110
OR   0111 0000
     ---------
     1111 0110
```



Description: If a statement is True (bit=1), NOT will return False (bit=0), and *vice versa.*

ML Application: This can be accomplished with the 6502 using an XOR with 1111 1111 ($FF).

Syntax: NOT A
$\overline{A}$
A'
^A, ↑A

A	NOT A
True	False
False	True

A	NOT A
1	0
0	1

Example:
```
    A = 1011 0110
NOT A = 0100 1001
```

Description: Exclusive Or returns True if either statement is True, but not if both are True.

ML Application: A number XORed with 0100 0000 will result in the same number with bit 6 toggled; i.e., =1 if it was originally 0, and =0 if it was 1.

Syntax: A XOR B, A EOR B
$A \oplus B$

Example:

```
     1011 0110
XOR  0111 0000
     ---------
     1100 0110
```

A	B	A XOR B
True	True	False
True	False	True
False	True	True
False	False	False

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

Description: The result is True only if both statements are False (bit = 0).

Application: This function is applied more in electronics than in programming.

Syntax: A NOR B $\overline{A + B}$ A ↓ B

Equivalent: NOT(A OR B)

A	B	A NOR B
True	True	False
True	False	False
False	True	False
False	False	True

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

Example:

```
     1011 0110
NOR  0111 0000
     ---------
     0000 1001
```

Description: If either statement is False (bit = 0), then the result is True.

Application: This is used more in electronics than in programming. In fact most early logic used only NAND gates and inverters (NOT).

Syntax: A NAND B $\overline{AB}$ A ↑ B

Equivalent: NOT(A AND B)
NOT A OR NOT B

A	B	A NAND B
True	True	False
True	False	True
False	True	True
False	False	True

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

Example:

```
      1011 0110
NAND  0111 0000
      ---------
      1100 1111
```

Summary Table

A	B	AND	OR	XOR	NAND	NOR
1	1	1	1	0	0	0
1	0	0	1	1	1	0
0	1	0	1	1	1	0
0	0	0	0	0	1	1

FOCUS ON THE 6809 MICRO WORTH SWITCHING FOR

FOCUS ... ON ITS SUPERIOR HARDWARE

- 6809E Microprocessor provides 8-bit economy with 16-bit capabilities, position independent code and many advanced features
- Commercial quality Keyboard with full UPPER/lower case ASCII, numeric keypad and cursor control keys
- Two double sided, double density mini diskettes with 640K bytes of IBM compatible storage
- Full Video with programmable screen formats, bit mapped graphics, user definable character sets, reverse video and hardware scrolling, plus a light pen interface
- Full Communications support for RS-232 at 50 to 19.2K baud with programmable data formats
- Built-in I/O Drivers include 6 parallel I/O ports, 3 serial I/O ports, 6 timers/counters, 20 mA current loop, and a programmable cassette interface

FOCUS ... ON ITS EXTRAORDINARY SOFTWARE

- Choice of two 'Universal' 6809 Disk Operating Systems supported by many companies: FLEX ™ or OS-9™
- Extended Floating Point Disk BASIC Interpreted and Compiled
- Text Editor and Macro Assembler
- Full feature MicroMon debugging monitor
- Built-in Word Processor

FOCUS ... ON ITS VERSATILE EXPANDABILITY

- IEEE-488 Bus Controller option for instrumentation
- Supports multiple 8" diskette drives
- PASCAL, FORTH and other high level programming languages are available
- Complete Application Packages are available from many software sources
- Add-on Hardware includes: Date/Time, Opto-Isolators, Stepper-Motor Controller, A/D and D/A, RAM and EPROM Memory, and others from several manufacturers

FOCUS ... ON YOUR APPLICATION REQUIREMENTS

- FOCUS provides a completely integrated system of hardware and software, so that you can concentrate on your application requirements

THIS IS FOCUS™

$3495.00

Very Complete

See us at the West Coast Computer Faire
Booths #1629 and #1631
March 19-21

THE COMPUTERIST®

34 Chelmsford Street
Chelmsford, MA 01824
Phone: 617/256-3649 Telex: 955318 INTL DIV

* Licensing arrangement for FLEX and OS-9 pending. Please contact The COMPUTERIST, Inc. for further details.

Get more out of your Apple with the MICRO on the Apple *series.*

VOLUME 2— just released!

You liked Volume 1—You'll love Volume 2!

More than 40 new programs on diskette to help you get more from your Apple:

- Machine Language Aids
- I/O Enhancements
- Runtime Utilities
- Graphics and Games
- Hardware and Reference Information

MICRO's new book for Apple users lets you:

1. Speed up machine language programming using 5 powerful machine language aids.
2. Add additional editing and input/output features at no cost!
3. Enhance the capabilities of Applesoft and Integer BASIC to include sorting, print using, and MORE.
4. Use your high-resolution graphics to plot graphs and display "compressed" pictures.
5. Add an interesting hardware project to your Apple.
6. Play the intriguing game "GalactiCube"—

And much, much more!
31 choice articles
46 tested programs on diskette
(13 sector DOS 3.2 format)

Buy **MICRO on the Apple 2** at your local computer store

or

Call **1-800-227-1617, ext. 564 TOLL FREE** (in CA, 1-800-772-3545, ext. 564) and charge it to your Mastercard or VISA

or

Send check for **$24.95** plus $2.00 surface or $5.00 air shipping to:
MICRO, Dept. MI
P.O. Box 6502
Chelmsford, MA 01824

Volume 1 also available at $24.95. Together **MICRO on the Apple 1 & 2** provide more than 60 programs on diskette for less than $1.00 each. No need to type in hundreds of lines of code.

Please allow 4-6 weeks for surface mail delivery.

Massachusetts residents add 5% sales tax.

/11101001a01001111/01000100o01100000/10001010x10101101/10011110na10101101/10011110no110

01011/01011110a01110010/01001000o1000001/11100011x10000010 /10011011na10101011/00100100no1001001/11111001a01111011/

MICRO

It's All 1's and 0's

Much of this month's recreational page consists of exercises in Boolean algebra. To learn more about Boolean algebra, consult this month's Technical Data Sheet (pages 121-122) and Marvin De Jong's article "Beginning Boolean: A Brief Introduction to Boolean Algebra for Computerists" (MICRO 22:29).

Each column represents the truth table for one of the simple Boolean operators (AND, OR, XOR, NAND, NOR). Fill in the missing 1's and 0's and name the functions. See technical data sheet (page 122) for answers.

Function

A	B	☐	☐	☐	☐	☐
1	1	0	☐	1	☐	1
1	0	☐	0	☐	1	☐
0	1	☐	☐	0	☐	1
0	0	0	1	☐	1	☐

Andrew Mossberg of North Miami Beach, Florida, sent in this list of secret machine instructions:

AII	Scatter Deck
BAH	Branch And Hang
BBBF	Branch on Bit Bucket Full
BCF	Branch on Chip box Full
BF	Blow Fuses
BOHP	Bribe Operator for Higher Priority
BSST	BackSpace and Stretch Tape
CUN	Cancel all User Numbers
EMW	Emulate Maytag Washer
ERD	Eject Removable Disk
EXOI	EXecute Operator Immediately
IA	Illogical And
KCE	Kill Consultant on Error
MST	Mount Scotch Tape
MVLR	MoVe and Lose Record
PDM	Play Drum Memory
PLSC	Perform Light Show on Console
PS	Print and Smear
RBP	Read Print and Blush
RCASD	Read Card And Scramble Data
RFSC	Read Feed and Shred Card
RIG	Read Inter-record Gap
RSD	Read and Shuffle Deck
RWRT	Read While Ripping Tape
SD	Scatter Deck
SPD	SPin dry Disk
SSD	Seek and Scar Disk
UER	Update and Erase Record

Harold Mathias of Southfield, Michigan, sent us this limerick:

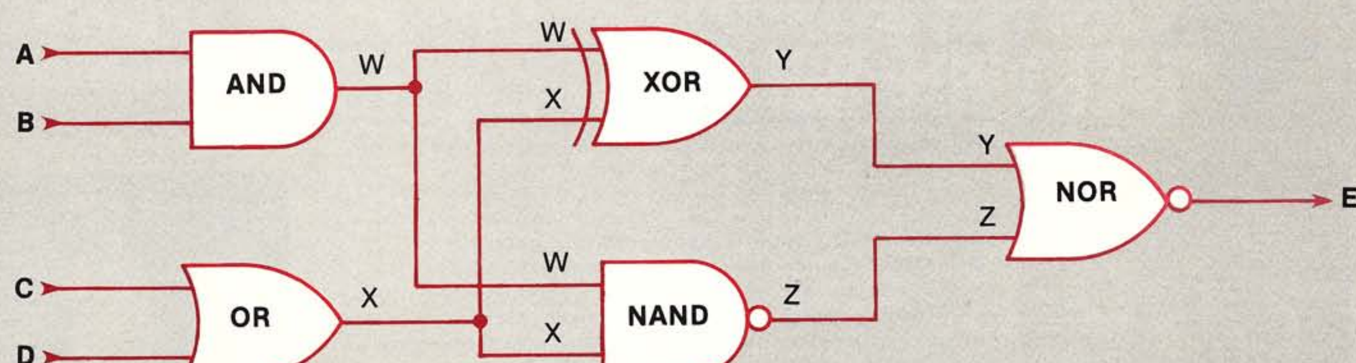
A MICRO subscriber named Tweed
Taught his microcomputer to read.
Now the damn machine
Won't give up the magazine
Though he does beg and plead.

Border

Rules and Hints: Combine the eight bits preceding each alphabetic separator with the eight bits following the separator to form a single 8-bit logical result which represents one ASCII character. The alphabetic separators stand for the following logical operations: a = AND, o = OR, x = Exclusive OR, na = NAND, and no = NOR. For example: 11010110a01011100 would AND the two values and get as a result 01010100, which is the ASCII code for the letter T. To make life easy, this month's message starts in the upper left corner. The / does not count for anything. It simply separates the individual values.

For answers to Border and Circuit puzzles, see page 103.

Circuit Puzzle



What is the result if inputs A, B, C, and D are all 1's? All 0's?

How many of the 16 possible input combinations result in an output (E) of 0?

(*Hint:* Set up a table with the inputs, intermediate results, and final result in separate columns.)

11011000x10011000 /01011111a11110110/01000000o01100100/11000100x10110110/10000000na10000001

10 /11111111no11011111/11111111na10111010/00010100no10001010/01010011o00100010 /01111001a11111111/11111111x11011110/

MICRO

Advertiser's Index

Next Month in MICRO

6809 Feature

Our April issue will cover 6502 to 6809 program translation, the FLEX operating system, and enhancements to Percom Data's CBUG monitor for the Radio Shack Color Computer. The PET Vet column will cover the role of the 6809 in the SuperPET. And, our new review department will make its debut with a concentration on 6809 products.

PLUS...

- **A Single Drive Copier in BASIC —** Here's a versatile disk copying program for OSI C2-4P and C4P-MF systems.
- **Applesoft Array Dump —** This handy debugging utility presents a "dump" of current variable array values for Applesoft in ROM.
- **AIM User Device Arbiter —** Expand the AIM's user input and output ports. Up to 83 devices each with the User Device Arbiter.
- **A Real Tape Operating System —** Learn how to get the most out of your PET cassette system, in terms of convenience and reliability.
- **7SEG — PET Giant Character Set —** Use this routine to display alpha-numeric characters in a large, seven-segment display on the screen.
- **Programmable Reverse Video for the C1P —** This article provides instructions for adding 100% programmable reverse video, character by character. It also offers programming hints for highlighting your graphics listings or games.
- **Integer Cross-Reference Utilities —** This article and the accompanying program confront the task of generating a complete cross-reference table for Apple Integer BASIC programs.