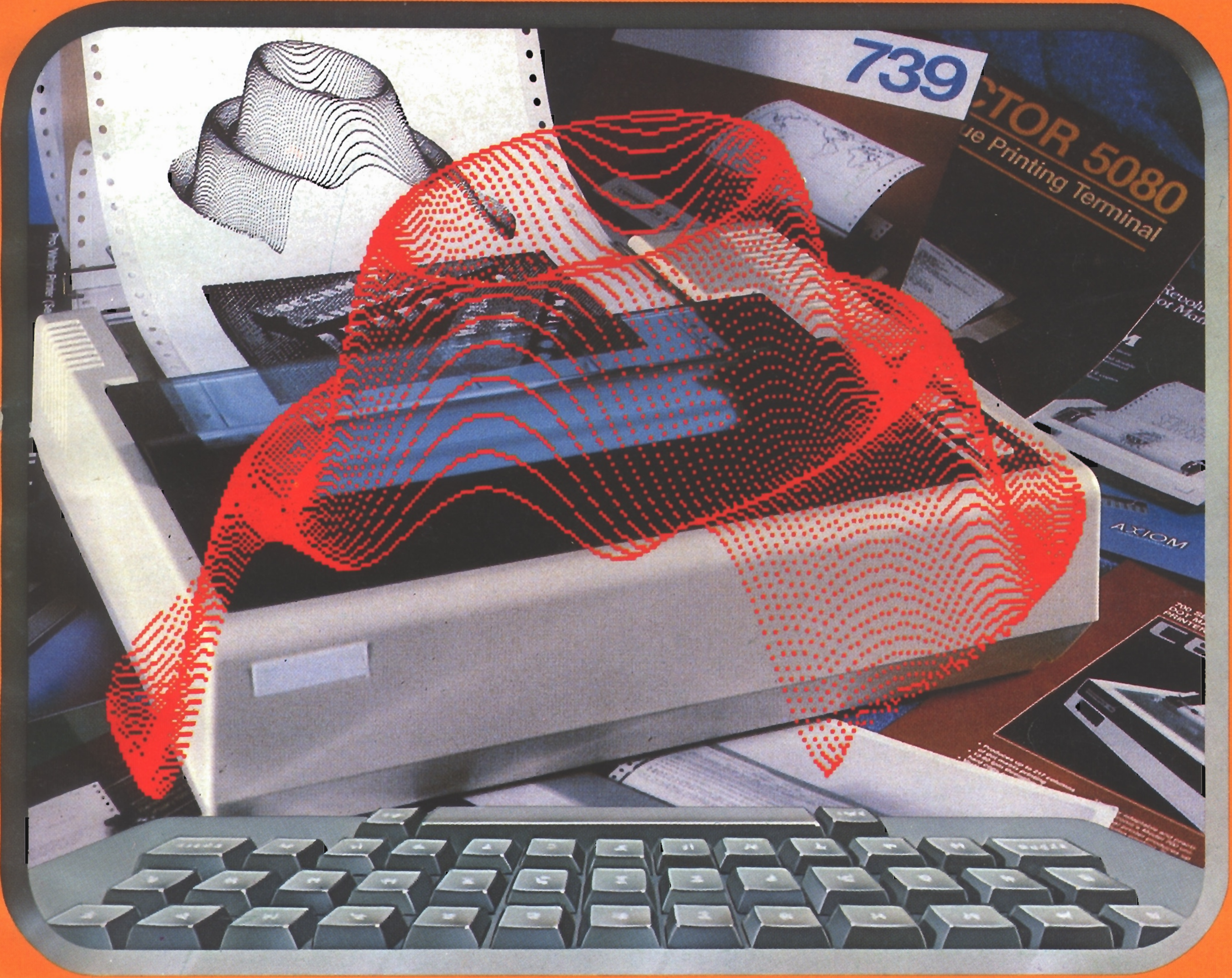
NO. 39 $2.50 AUGUST 1981

MICRO™

THE 6502/6809 JOURNAL



Printer bonus section

Apple bonus section

Expanding the Superboard

Microcrunch, Part 1

Improved *n*th Precision

Disassembling to Memory on AIM 65

16K RAM EXPANSION BOARD FOR THE APPLE II* $195.00

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.

*Apple II and Applesoft are trademarks.

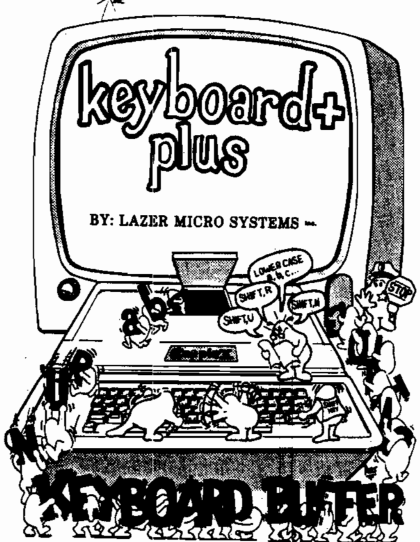
ANDROMEDA
INCORPORATED
P.O. Box 19144
Greensboro, NC. 27410
919 852-1482

Distributed By:

COMPUTER DATA SERVICES
P.O. Box 696
Amherst, NH. 03031
603 673-7375

the BEST keyboard buffer

Lazer



& SHIFT KEY UPPER/LOWER CASE CONTROL

$119.95

+ More buffer than others.
+ Clear buffer control.
+ SHIFT key entry of upper/lower case.
+ Easy CTL key access to special chars " | ~ ` { } _ [\ □".
+ Allows BASIC programs with standard INPUT to support Lower Case without software modification.

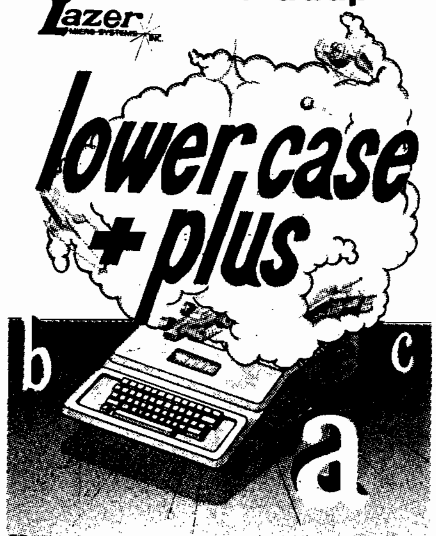
⇦ AND ⇨

Separately, they have more features and out perform all the rest. But together as a team they perform even better. Look for the Graphics +Plus soon. It's a RAM based character generator to complement the Lower Case +Plus. It will allow you to define the character set to your needs. You could load German, French, Scientific, Engineering or any other special characters into the Graphics +Plus and use it as if the Apple II was designed specially for that application. And that's not all. If you define the characters as graphics, you can do extremely fast HI-RES type graphics on the text screen without all those cumbersome and slow HI-RES routines and 8K screen. For all the details on this triad of products, send for our free booklet "Lower case adapters and keyboard buffers from the inside out". This booklet gives all the details about lower case adapters and keyboard buffers in general. It also has a section on the Graphics +Plus (RAM based character generator).

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

the BEST lower case adapter

Lazer



GRAPHICS & LOWER CASE CHARACTER GENERATOR FOR THE APPLE II COMPUTER

$69.95

+ Normal & Inverse Lower Case.
+ 2 complete character sets on board.
+ Graphics character font built in.
+ Expansion socket allows access to external character sets.
+ 2716 EPROM compatible char generator.
+ More supporting software. (on diskette)
+ Keyboard +Plus & Graphics +Plus designed around the Lower Case +Plus.

the Keyboard +Plus

The Keyboard +Plus is a multi-purpose input device for your Apple II. The first thing the Keyboard +Plus will do for you is save you lots of time. When the old adage "time is money" being more true than ever, you naturally want to know how this device can save you and your employees time. We'll start with the input buffer. With the normal Apple II, you can only input data when the computer is ready for it. Not when the disk drive is running or when a printer without buffer is operating, not when Applesoft is performing the FRE(0) function and not when the Apple is off performing time consuming multiple calculations. Sometimes these time (takers) take only a brief time and sometimes they take a long time. Even if they only take a brief time, the operator can loose his train of thought and have to re-orient himself to get back to speed. With the Keyboard +Plus' buffer, the operator can keep right on typing. The buffer will store up all those keystrokes until the computer comes back and requests another input. In most conditions, you will never be more than 2 or 3 keystrokes ahead of the computer. At most, you will probably never have much more that 35 or 40 characters ahead. The Keyboard +Plus has room for 64 characters to be stored, which is far more that you will probably need. The second timesaver the Keyboard +Plus has to offer is the SHIFT Key control of upper/lower case entry. You no longer have to re-orient yourself from the typewriter style keyboard and the Apple II keyboard every time you switch from one to the other. The frustration of the difference without the Keyboard +Plus is worth the cost alone. There are other benefits such as CTRL key entry of all the special character you could not access before and a lot of the Apple keyboard bounce (getting two characters for one stroke) will disapear. Besides these features, there is a keystroke command to clear the buffer as well as RESET key protection for the older Apples. With all these features, it's no wonder that Lazer MicroSystems is becoming known as the company that puts thought into all their products.

the Lower Case +Plus

The Lower Case +Plus is a plug in (not I/O slot) device that will allow your Apple II to display lower case and graphic characters on the video text screen. The Lower Case +Plus is compatible with ALL word processors that support lower case. With an optional (extra cost) character generator, it will also allow some word processors, such as Applewriter and the 40 column Easywriter, to display normal upper and lower case on the screen with no software modifications. The Lower Case +Plus is compatible with all software that operates with any other lower case adapter. However, since the Lower Case +Plus has features and capabilities that no other lower case adapter has, there is software available that will operate properly only with the Lower Case +Plus. Maybe that is the reason the Lower Case +Plus has become the leading lower case adapter for the Apple II.

Lazer MicroSystems' products are in computer stores all across the country. However, if you cannot locate one, you can order direct from us.

* California residents must add 6% sales tax.
* Master Card & Visa (W/all vital info) welcome.
* Allow 2 weeks additional for checks to clear.
* Orders outside U.S.A. add $15.00 for shipping & handling.

Lower Case +Plus, Keyboard +Plus, and Graphics +Plus are trademarks of Lazer MicroSystems, Inc., Corona, CA.

Apple II and Applewriter are trademarks of Apple Computer, Inc., Cupertino, Calif.

Easywriter is a trademark of Cap'n Software, and produced by Information Unlimited Software, Inc., Berkeley, Calif.

MICRO™
THE 6502/6809 JOURNAL

ARTICLES

PRINTER BONUS

APPLE BONUS

DEPARTMENTS

MICRO

Letterbox and Microbes

Dear Editor:

As a long-time supporter of the 6800 and 6502 families, (I was one of the first dealers to sell Apple I, OSI Challengers and SWTP M6800 microcomputers), I am glad that MICRO will now cover the 6809. This greatly improved micro offers so many advantages, that new users rave about this chip once they understand it. I am sure your excellent series will encourage many to try it. The SS-50 Bus users are about a year ahead in the understanding and use of the 6809, but I am sure that the Apple, PET/CBM, SYM, KIM, and AIM users will catch up fast. We have to welcome a new group into the fold — the TRS-80 Color Computer users. They not only use the 6809, but they can cable into the SS-50 Bus for expansion, before Radio Shack offers it to them.

The point that I really would like to make is that 6809 is an interim processor. For all it's excellence, it is a forerunner to the M68000, which is the microprocessor of the future. The M68000 is so far above anything we use today that we will need all the technical help we can get, to understand it and use its great power. I would like MICRO to not only raise our sights to the 6809, but beyond it to the 68000. Thank you for your excellent magazine.

Stanley Veit

We'd like to take this opportunity to thank everyone who has written. Unfortunately we cannot publish all the letters that we receive. However, your letter has a better chance of being published if you are brief, to the point, and cover only one topic per letter.

Jan Skov of Denmark sent this note:

In MICRO (36:37) you made a disastrous comment. SYM-BASIC does indeed support integer variables. Your mistake is understandable as the manual nowhere mentions %-type variables.

I know that integer variables work because I never bothered to read the manual; I just programmed and assumed. Please tell your readers!

Mark L. Crosby of Washington, D.C., sent this microbe:

In the June issue of MICRO some errors of omission occurred in Alan Hill's article "Amper Search for the Apple" (37:71). These might be difficult for novice assembly language programmers to figure out.

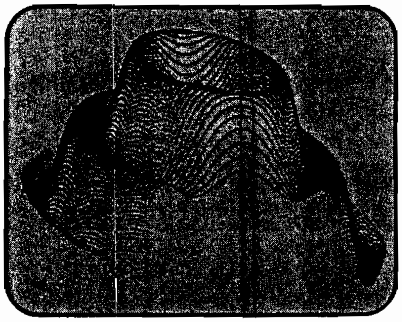
Although the original program was created with a different assembler, the corrections in figure 1 were done on the Apple Tool Kit Assembler/Editor (by Apple Computer Inc.).

The corrections begin at the section headed "DATA STORAGE."

```
956D C1 CD D0          ASC 'AMPER-SEARCH'
9570 C5 D2 AD
9573 D3 C5 C1
9576 D2 C3 C8
9579 C1 CC C1          ASC 'ALAN G. HILL'
957C CE A0 C7
957F AE A0 C8
9582 C9 CC CC
9585 C3 CF CD          ASC 'COMMERCIAL RIGHTS'
9588 CD C5 D2
958B C3 C9 C1
958E CC A0 D2
9591 C9 C7 C8
9594 D4 D3 A0
9597 D2 C5 D3          ASC 'RESERVED'
959A C5 D2 D6
959D C5 C4
959F CB 93    LOC      DFB $CB,$93          ; DEALLO-1
95A1 23 92             DFB $23,$92          ; SEARCH-1
95A3 44       CHRTBL   DFB $44              ; D
95A4 53                DFB $53              ; S
95A5 8D       MSG1     DFB $8D              ; <CR>
95A6 D6 C1 D2          ASC 'VARIABLE'
95A9 C9 C1 C2
95AC CC C5 A0
95AF A0 A0 A0 NAME     ASC ' '              ; 16 SPACES
95B2 A0 A0 A0
95B5 A0 A0 A0
95B8 A0 A0 A0
95BB A0 A0 A0
95BE A0
95BF 8D                DFB $8D              ; <CR>
95C0 CE CF D4          ASC 'NOT FOUND'
95C3 A0 C6 CF
95C6 D5 CE C4
95C9 C0                DFB '@'              ; CTRL-L
95CA A0 A0 A0 SV50     ASC ' '              ; 6 SPACES
95CD A0 A0 A0
95D0 A0 A0 A0 ZPSV     ASC ' '              ; 32 SPACES
95D3 A0 A0 A0
95D6 A0 A0 A0
95D9 A0 A0 A0
95DC A0 A0 A0
95DF A0 A0 A0
95E2 A0 A0 A0
95E5 A0 A0 A0
95E8 A0 A0 A0
95EB A0 A0 A0
95EE A0 A0
```

Figure 1

About the Cover



The Printer Revolution

Just as processor technology has exploded in the past several years, so has printer technology. Printers available today offer several times the features of yesterday's printers, at a fraction of the price. The old mechanical monstrosities, so common in computer rooms before the microprocessor boom, could hardly produce a legible, life-long hard copy, let alone a letter-quality output. Now, a new breed of printer, controlled by microprocessor instead of relays, can produce graphical output as well as a variety of printing fonts. Parallel interfaces have enabled these printers to output at much greater speeds than their ancestors. And along with the increase in versatility, quality, and speed, in the past several years we have seen a noticeable decline in price! This decline is due in part to new technologies in thermal and dot-matrix printing, and in part to the commercial popularity of such printers. In this issue, with its special printer section, MICRO salutes the "printer revolution."

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and Avon, MA 02322
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
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

Subscription rates	*Per Year*
U.S.	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

MICRO Editorial

This issue marks an unusual and important occasion for MICRO. After thirty-eight consecutive editorials, the first of which appeared back in 1977, Editor/Publisher Bob Tripp has finally decided to take a break. Thus, the task of writing this month's editorial has been passed to the editorial staff, and has landed on me! My name is Ford Cavallari, the Apple specialist at MICRO and the editor of the series *MICRO on the Apple.* Starting this month, I assume additional responsibilities for the magazine as an associate editor. Let me take this opportunity to share with you some thoughts that I, along with the rest of the staff, have been having about the magazine's course.

This month, the first non-system oriented bonus section makes its appearance in our magazine. In June, as you may recall, we enlarged MICRO, in part to extend our coverage of the Apple, and in part to expand our coverage of other systems and other areas. The two bonus sections which now appear in each issue afford us quite a bit of editorial flexibility, and this flexibility is reflected in this month's special printer bonus. With this new format, we have tackled an in-depth special on printers without sacrificing other areas of the magazine's coverage. In fact, we did it with ease, and still provided additional Apple coverage!

In the coming months, we will be presenting more widely varied bonuses, ranging from more system-oriented coverage of the PET, the OSI, the Apple, and the single boards, to some more concept-oriented features on topics like games, computer languages, and the 6809.

I am particulary excited about the coming games bonus section which will be appearing in November. While MICRO has historically leaned more toward the serious computer user than toward the gamester (see September 1980 Editorial, 28:5), we do realize, and concede, that there are few microcomputer demonstrations quite as graphic or fun as a good game. Also, there are very few ways to get children interested in computers, aside from games. Our games bonus section will feature games articles, games programs, and games advertising, just in time for the gift-giving season. If you have original material which you feel would be appropriate for this section, please send it in, and we will consider it. We plan each issue months in advance, so send us your original games and articles quickly.

Another coming feature is our Pascal bonus section, scheduled for January. Pascal is now available on many microprocessors, and will soon become available on more. It is evident, in both the micro and mainframe communities, that Pascal is going to be very important in the future. The Pascal bonus section should be of interest to the novice and expert alike, for it will include both introductory tutorial material and programs demonstrating advanced techniques. Other languages to be featured in future bonuses are FORTH, BASIC, and assembly language.

OSI reasders will notice the ommission this month of the *Small Systems Journal.* The Journal has not moved to another publication. Rather, it has been suspended indefinitely by Ohio Scientific. We regret this, because we believe the Journal provided OSI users with a valuable service in a format unique to the microcomputer industry. If you feel strongly about the Journal, why not let OSI hear directly from you in writing! In the mean time, keep the OSI articles coming in and keep reading MICRO as we schedule more OSI bonuses.

One last word on the Reader Survey Form appearing in last month's MICRO. When Bob Tripp started MICRO back in 1977, it was partially due to the fact that he felt the 6502 community to be a more cohesive, enthusiastic group than, say, the 8080 community. The tremendous response that we've gotten so far from the Reader Survey indicates that your group enthusiasm has not waned. If you haven't sent in your form, and if you wish to have a direct influence on the magazine, here is your chance. In order for us to schedule features and bonuses, we have to have some idea of who is going to read them. Thanks for the response so far. Let's make it 100 per cent.

OSI **AARDVARK** OSI

AARDVARK NOW MEANS BUSINESS!

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 (0S65D 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

THE EDSON PACK ALL MACHINE CODE GAMES FOR THE 8K C1P

INTERCEPTOR —You man a fast interceptor protecting your cities from Hordes of Yukky Invaders. A pair of automatic cannon help out, but the action speeds up with each incoming wave. It's action, action everywhere. Lots of excitement! $14.95

MONSTER MAZE — An Arcade style action game where you run a maze devouring monsters as you go. If one sees you first, you become lunch meat. Easy enough for the kids to learn, and challenging enough to keep daddy happy. $12.95

COLLIDE — Fast-paced lane-switching excitement as you pick up points avoiding the jam car. If you succeed, we'll add more cars. The assembler code provides fast graphics and smooth action. $9.95

SPECIAL DEAL—THE ENTIRE EDSON PACK— ALL THREE GAMES FOR $29.95

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

ACCOUNTS RECEIVABLE — This program will handle up to 420 open accounts. It will age accounts, print invoices (including payment reminders) and give account totals. It can add automatic interest charges and warnings on late accounts, and can automatically provide and calculate volume discounts.

24K and OS65D required, dual disks recommended. Specify system.
Accounts Receivable. $99.95

*** * * SPECIAL DEAL — NO LESS! * * ***

A complete business package for OSI small systems — (C1, C2, C4 or C8). Includes MAXI-PROS, GENERAL LEDGER, INVENTORY, PAYROLL AND ACCOUNTS RECEIVABLE — ALL THE PROGRAMS THE SMALL BUSINESS MAN NEEDS. $299.95

P.S. We're so confident of the quality of these programs that the documentation contains the programmer's home phone number!

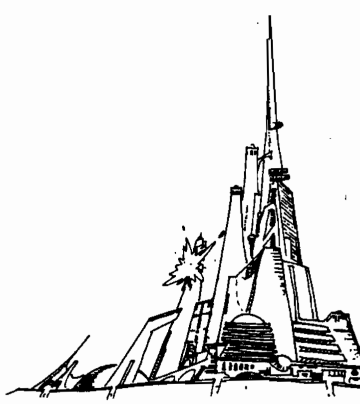
SUPERDISK II

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 (5 1/4") $34.95 (8").

AND FUN, TOO!



BOOKKEEPING THE EASY WAY — WITH BUSINESS I

Our business package 1 is a set of programs designed for the small businessman who does not have and does not need a full time accountant on his payroll.

This package is built around a **GENERAL LEDGER** program which records all transactions and which provides monthly, quarterly, annual, and year-to-date PROFIT AND LOSS statements. GENERAL LEDGER also provides for cash account balancing, provides a BALANCE SHEET and has modules for DEPRECIATION and LOAN ACCOUNT computation.
GENERAL LEDGER (and MODULES) $129.95.

PAYROLL is designed to interface with the GENERAL LEDGER. It will handle annual records on 30 employees with as many as 6 deductions per employee.
PAYROLL - $49.95.

INVENTORY is also designed to interface with the general ledger. This one will provide instant information on suppliers, initial cost and current value of your inventory. It also keeps track of the order points and date of last shipment.
INVENTORY - $59.95.

GAMES FOR ALL SYSTEMS

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

MINOS - 8K - — Features amazing 3D graphics. You see a maze from the top, the screen blanks, and when it clears, you are in the maze at ground level finding your way through on foot. Realistic enough to cause claustrophobia. — $12.95

NEW — NEW — NEW

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.

TIME TREK - 8K - Real Time and Real graphics Trek. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! — $9.95

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

Please specify system on all orders

This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.

OSI

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

OSI

MICROCRUNCH: An Ultra-fast Arithmetic Computing System

Part 1

Extremely fast floating point processing can be attained by coupling an INTEL 8231 arithmetic processing unit to the OSI Superboard II and using a partial compiler to generate machine code representations of mathematical equations and loops written in BASIC.

John E. Hart
Dept. of Astrogeophysics
University of Colorado
Boulder, Colorado 80309

An editorial in *BYTE* magazine (*BYTE*, vol. 5, number 10, Oct. 1980) quoted a survey that indicated that 40% of the readers of that microcomputer magazine were scientists or engineers. Obviously a very large number of small system users got into microcomputing because they hoped to use their machines for mathematical problems occurring in these fields. Although many applications of 6502 processors have been in tasks that do not require sophisticated mathematical manipulation (like graphics, games, word processing, etc.) there is certainly a host of interesting and/or practical problems that can be approached via numerical analysis on a microcomputer. These problems span the entire spectrum of mathematical modeling, from ecosystems to weather systems, from circuit analysis to support calculations in data analysis.

Such applications are only limited by the product of the floating point throughput (or speed) of the microprocessor and associated software, and the patience of the operator to wait around for the answer. It is often most profitable and convenient to approach mathematical problems in an interactive mode, where, for example, a problem depending on a certain parameter is iterated to an end point. The result is then inspected by the operator, the parameter varied, and the solution repeated, until the desired answer is obtained. Such a scheme would be fruitful if the iteration time is fairly short. If you have to wait half an hour between answers it can be very frustrating. The iteration time is, of course, proportional to the length of the mathematical problem, in terms of the total number of floating point operations per iteration, divided by the effective computing speed of the machine being used. Unfortunately when it comes to floating point number crunching, microcomputers can be annoyingly slow. The purpose of this series of two articles is to describe a 6502-based system called MICROCRUNCH that is extremely fast at floating point mathematical number crunching.

The system consists of an OSI Superboard II with the 610 board memory expansion, interfaced to an INTEL 8231 math chip, which will be discussed later, in detail. This article describes the hardware necessary to accomplish this interface.

True number crunching speed is only possible if such a math chip is treated as a co-processor in the sense that floating point operations executed by the 8231 are done asynchronously as the 6502 is preparing for the next operation. Thus a special BASIC compiler that converts higher order statements into optimal 6502 machine code is a must if the potential for fast execution inherent in the 8231 is to be realized. Part 2 of this series will describe the software necessary to do this. We start by indicating what kind of speeds can be attained with the MICROCRUNCH system.

Computing speed for mathematical applications is usually measured in terms of megaflops (Mflops); or millions of floating point operations (+, −, *, /) that a computer, plus associated support software can execute per second. Obviously no one expects a micro to compete with a 32-bit mainframe designed specifically to do scientific computing, but it is interesting to compare a few typical systems in this regard and to note how well a little 8-bit micro can perform. Computing speed can be crudely estimated by running the following simple benchmark program on several machines.

```
A=1.00013
X=1
FOR I=1TO40000
X=X * A
NEXT I
PRINT X
STOP
```

From this, one gets a pretty good idea of the Mflop capability of a machine, since usually, the overhead for the FOR loop part of this little program is small compared to the time it takes to look up the variables X and A, and to perform the multiplication. I have tried this little loop on a variety of computers, some of which used a FORTRAN version. The results are shown in table 1.

There are several conclusions that can be made from this table, such as:

1. Traditional 6502 or Z-80 machines with BASIC interpreters are quite slow, doing about 100 to 200 flops per second. A calculation with 10,000 flops would take a couple of minutes, which is too slow for comfortable interactive computing.

2. The use of a compiler (Pascal or FORTRAN) on the straight 6502 machines only helps by a factor of 2 or so in speed. Although for a compiler the variable fetch and line decode times go way down, the time for actual addition, division, etc., in floating point stays the same.

3. Increasing the computer clock helps in direct proportion to the clock increase. At most, this might gain a factor of 4 if the typical 6502 micro can be made to run at 4 MHz.

4. Floating point chips without compilers are almost useless.

5. The optimal 8-bit system described here outperforms many standard minicomputers, at a fraction of their cost.

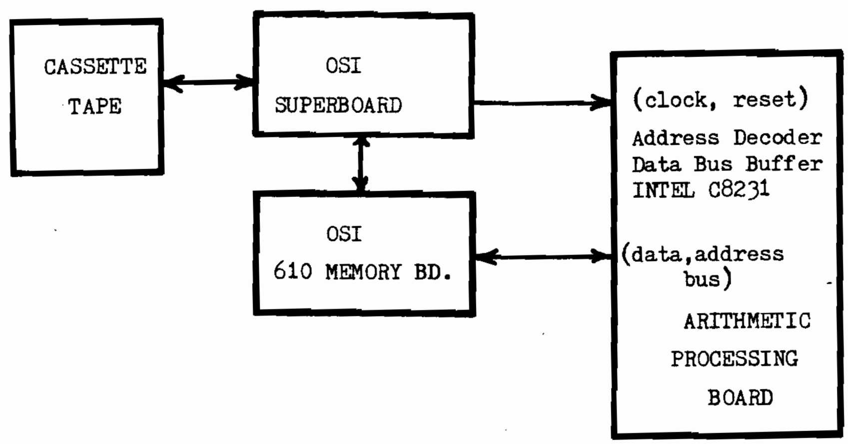
6. If you want personal number crunching in excess of around .01 Mflops (10^4 floating point operations per second), be prepared to spend a large amount of money.

Assuming the reader is interested in attaining floating point throughput in excess of 50 times the typical micro performance, we proceed to outline the MICROCRUNCH hardware, including circuits, a layout, and a parts list.

Table 1: **Approximate Megaflop Rates for Several Computing Systems**

Computer	Language	Mflop (million flops/sec)
TRS80 model I (Z-80)	BASIC interpreter	.00012
TRS80 model II (Z-80)	" "	.00026
INTERCOLOR (Z-80)	" "	.00014
APPLE II (6502, 1 MHz)	" "	.00019
APPLE II	Pascal compiler	.00034
APPLE II w/AMD9511 floating point board (Calif. Digital)	APPLEFAST interp.	.00026
OSI Superboard II (1 MHz)	BASIC interpreter	.00022
OSI Superboard II (2 MHz)	" "	.00044
PDP1103 w/Hdw. floating point board (DEC)	FORTRAN	.004
*MICROCRUNCH (OSI 2 MHz + INTEL 8231)	BASIC compiler	.011
PDP 1134	FORTRAN	.04 approx.
VAX 11/750 (DEC)		.4 "
CDC 7600	"	4-6 "
CRAY I	"	60 "

Figure 1: **MICROCRUNCH Hardware**



The Hardware

The physical system is shown in figure 1. The basic computer is the OSI Superboard II. It has been connected to a fully populated OSI 610 memory board. Thus the starting element is essentially a 6502 computer with 32K of RAM. The 610 board has an expansion plug that contains buffered data, address, phase two, read/write, and interrupt lines. This is connected to the arithmetic processor board (APB) whose circuit is given below. This APB board could be connected to any 6502 system that has available the same buffered lines as on the OSI 610. These are given in more detail in table 2.

Thus, in principal, the APB circuit can be used on a variety of machines (AIM, Apple, etc.) provided the address assigned to the arithmetic processor does not conflict with the memory map of the host computer. Because the compiler described in part 2 of this article uses up 20K of memory, and the upper 12K of this system is needed for source and object code storage, there is not much room left for a disk operating system. So, I use magnetic tape as a bulk storage medium. This would not be necessary if a machine with 48K of RAM were employed. However, the tape storage system I use is almost as fast as disk, so there is little performance loss here (see "An Ultra-Fast Tape Storage System," J.E. Hart, MICRO, November 1980, 30:11).

In addition I have jumped the fundamental clock on my Superboard up to 2 MHz as described by J.R. Swindell ("The Great Superboard Speedup," MICRO, February 1980, 21:30). The timing for the MICROCRUNCH system in table 1 was with a 2 MHz clock. For 1 MHz, the Mflop rate is .007. The tape baud rate and clock modifications are not necessary for successful operation of the APB, but they are useful changes that increase performance and convenience.

The APB part of the system consists of an address decoder, a data bus buffer, a read/write/command/data decoder and the INTEL 8231 arithmetic processing unit. In order to understand the circuits that follow it is necessary to give a brief description of the 8231.

Anyone getting into this project should obtain the 8231 manual from a local INTEL representative, since only a brief sketch of the processor can be given here. When ordering this part, be sure to get the C8231, since this will run at 4 MHz and the regular 8231 will not. The 8231 has the following features of interest:

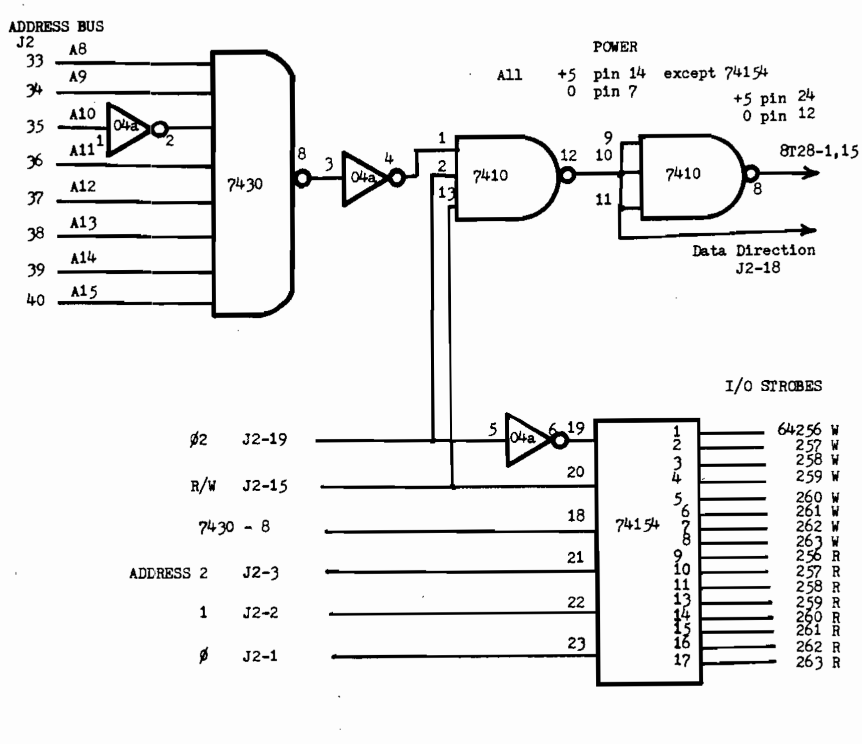
1. An operand stack that stores 4 floating point numbers with 6½ decimal digit precision and a range of about $10^{\pm 20}$. Each floating point number is represented by 4 bytes: 1 for the exponent and 3 for the mantissa. The floating point format will be discussed in part 2. It is, unfortunately, not the same as that used by Microsoft BASIC.

2. A 1-byte status register that can be read into the 6502. This status register contains a busy bit that in-

Table 2: **Connector J2a on Arithmetic Board**

Pin	Function	
1	buffered address bus bit	0
2	" " " "	1
3	" " " "	2
4	" " " "	3
10	buffered data bus bit	0
	" " " "	1
11	" " " "	2
12	" " " "	3
13		
15	buffered read/write (read to 6502 if high)	
18	data direction (enable read to 6502 if low)	
19	buffered phase 2 clock	
28	buffered data bus bit	4
		5
29	" " " "	6
30	" " " "	7
31		
33	buffered address bus bit	8
		9
34	" " " "	10
35	" " " "	11
36	" " " "	12
37	" " " "	13
38	" " " "	14
39	" " " "	15
40	" " " "	

Figure 2: **Address Decoder Circuit**



dicates whether a previously initiated floating point command is still in progress, and an error field that indicates if the previously completed command resulted in an error (overflow, underflow, divide by zero, improper function argument like square root of a negative number, etc.).

3. A 1-byte command register that is written into by the 6502. This initiates a floating point operation on the operand(s) that are stored on the stack in the 8231. These operations include +, −, *, / and a host of transcendental functions like SIN, COS, ARCTAN, etc. (See the manual for a complete description of these.) Suffice it to say that just about any problem you could have done with Microsoft BASIC you can do within the 8231, only much faster. The result of a calculation or operation appears on the top of the stack and can hence be read as a four-byte block transfer back into memory, under control of the 6502. These manipulations and some quirks of the floating stack are discussed in part 2, since they have more to do with software than hardware.

The scenario that emerges is as follows: A mathematical program written in BASIC is compiled by the 6502. There the object code, so generated, causes appropriate 4-byte transfers in and out of the APB, of floating point variables appearing in the mathematical expressions that were compiled. The 6502 also sends operation commands at the appropriate times and checks for errors after an operation is completed. Thus the main task of the hardware is to allow the 6502 to transfer data in and out of the 8231 stack, command, and status registers. Thus, we are really concerned with a fast I/O problem.

Readers of the 8231 manual will note that it also does fixed point arithmetic (16- or 32-bit). None of these functions are used in the MICROCRUNCH system, but software could be written to use these if needed.

Circuit Description

Described below is the circuit for the APB and its interconnections to the 610 board. The components for this board, all bought retail, cost about $340, with $270 going for the INTEL C8231. In addition, the 8231 uses 12V DC so a regulated supply of some sort (low current, 100 mA is fine) is needed. It should be mentioned here that the 8231 is identical in architecture and pin-outs to the older AMD 9511. The latter chips are a little cheaper ($195), but are designed to run at 2 MHz instead of 4 MHz. I went with the INTEL because the speed increase seemed worth the extra money.

The main interface with the 610 board is via its connector J2. This 40-pin connector is linked to a similar 40-pin IC socket-type connector on the APB with a ribbon cable. Table 2 shows the lines available on J2 that are used on the APB.

In addition to this interface, an additional connector J3 must be used to supply the following signals from the Superboard itself. In my unit this is a 14-pin IC socket connected by a ribbon cable to a similar socket set in one of the prototype holes in the Superboard.

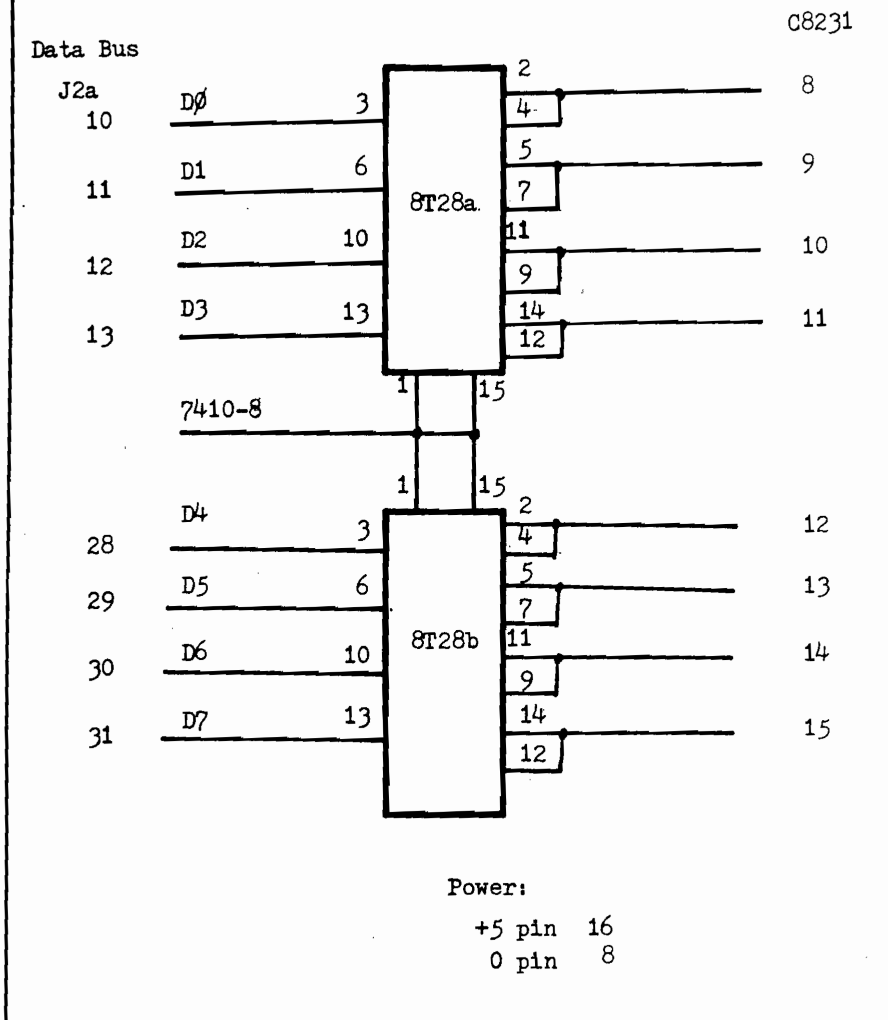
J3 - 1	4 MHz clock (SBD II, U30 - 2)
J3 - 2	Ground
J3 - 3	BRK line (low = reset) (SBD U8 - 40)

The APB circuit will work with any 6502 computer that supplies the I/O connections as described above.

Figure 2 shows the address decoder circuit. Address lines 8 through 15 are fed into an 8 input nand gate, and line 10 is inverted. The output of this gate will go low whenever the address high byte goes to $FB. This is the basic block address for the APB. The output of this gate is fed to one enable input of a 74154 4-to-16-line demultiplexer, and to a set of inverters and gates whose purpose is to generate a data direction pulse in phase with the 02 clock pulse. The outputs of the 74154 are a set of strobes that go low in phase with 02 whenever address FB is selected. Only one strobe is fired, depending, as well, on the R/W, A0, A1, and A2 lines. These strobes can be used to select various I/O devices, 16 in all. For the APB we shall use only 5 of these lines, so the others can be used for future expansion (A-D, D-A, etc.). The data direction pulse does two things. It informs the data buffers on the 610 board when data is going to be fed back to the 6502 (J2-18, low = read) and after inversion, chip 7410-8 does the same for the data buffers just ahead of the 8231.

Figure 3 shows the interconnections for the two on-board 8T28 tri-state buffers needed to drive the cable connecting the APB to the 610 board.

Figure 3: Bus Buffer



Finally, figure 4 shows the interconnections between the strobe lines from the address decoder and the 8231. During a write operation pin 1 of the 7402 NOR gate will go low. This signal is inverted and fed through another part of the 7402 quad NOR gate to give a low CHIP-SELECT pulse. The 8231 timing requirements indicate that the active low WRITE pulse must be shorter than the CHIP-SELECT input so the WRITE strobe is shortened by feeding into a 74123 one-shot. If an operand is being written onto the 8231 floating stack, pin 21 must go low. This is accomplished by sending the inverted WRITE OPERAND strobe to 7402-8. The resulting inverted OR pulse then becomes the appropriate C/$\bar{D}$ line.

A read of either the operand stack or the status register is preceded by a READ INITIATION strobe. For example, a READ STATUS START strobe (e.g. LDA ABS FB00) sets flip-flop 74LS76A. The output of this flip-flop goes high and causes the CHIP-SELECT line (APU-18) and the READ line (APU-20) both to go low. The 8231 then proceeds to send the status register contents to its internal data bus buffer. This takes several clock cycles (like an EPROM), so data is not entered into the 6502 accumulator until a READ ENTER strobe is fired. That is, flip-flop A stays set until an LDA-ABS FB06 instruction is executed. Then strobe line 74154-16 goes low terminating the read by resetting the flip-flop on its rising edge.

Typically, then, two consecutive LDA's are used to read from the 8231. Data is read by LDA-ABS FB01, LDA-ABS FB06. The only difference between this and a status read is that flip-flop B sets the C/$\bar{D}$ line low (via 7402-10) in addition to pulling the CHIP-SELECT and READ lines low. The double LDA read cycle required by this circuit is slightly (20%) less efficient in time than using the 6502 ready line in a pause circuit. Unfortunately, in the Superboard this line is tied to ground. However, during long mathematical manipulations one is almost always writing data and commands into the APU, reading only at the end of a string of operations. Therefore, this lost time becomes insignificant.

The 4 MHz clock and the reset pulses are connected as indicated.

Table 3 gives the APB addresses and typical commands used to communicate with it. For machines other than OSI, these addresses may fall in already assigned areas of the memory map. If so, the base address FB can easily be changed by altering the inputs to the 7430 address gate. For example, if the inverter on line 10 is not used, the high part of the APB address will be $FF. If this is done, however, some straightforward address changes will need to be made in the software presented here and in part 2.

***Table 3:* Arithmetic Board Addresses and Machine Code Access Statements**

Address		Function	Machine Code	
64256	FB00	APU READ STATUS start	LDA-ABS	FB00
64257	FB01	APU OPERAND READ start	LDA-ABS	FB01
64262	FB06	APU WRITE OPERAND	STA-ABS	FB06
64262	FB06	APU READ DATA (status or operand, as determined by previous start pulse)	LDA-ABS rt	FB06
64263	FB07	APU WRITE command to initiate operation	STA-ABS	FB07

***Figure 4:* Arithmetic Processing Unit Control**

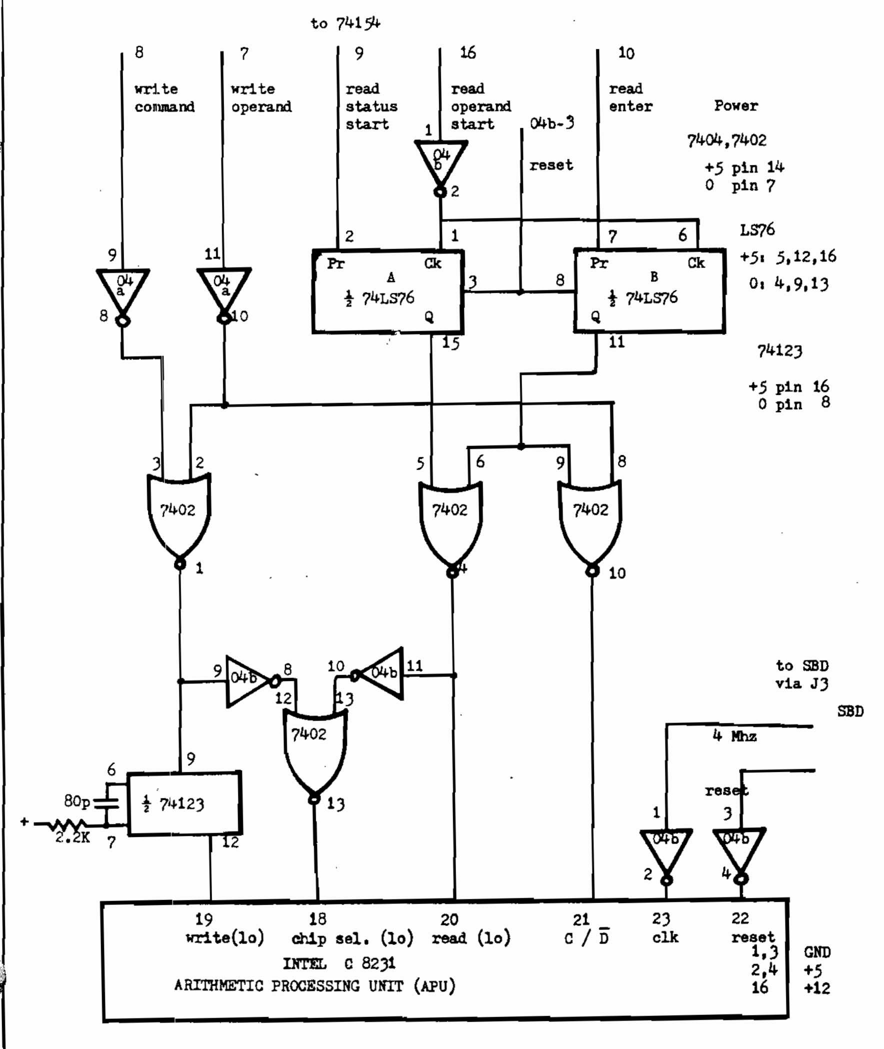


Figure 5 gives a typical layout for the APB. One first installs the wire-wrap sockets (assuming the board will be wire-wrapped, not soldered), and routes the power lines. Install .01 mfd bypass capacitors on each chip between the +5 volt line and ground. After wrapping the preceding circuits, the board should be tested using some simple programs presented below. The basic questions are, can you get operands in and out of the unit, and can you command it to execute operations?

Testing

The first program listed in the appendix asks for an operation code. Among some useful ones for testing are: 26 = push constant pi onto top of operand stack, 16 = floating add, 17 = floating subtract, 18 = floating multiply, 19 = floating division, 2 = SIN, 3 = COS, 25 = exchange top operand with next lower operand. At the first request for an operation code, enter 26. The program then reads the stack, and assuming all is well, the top four bytes should represent the constant pi in the APU format. The arithmetic processor representations of several useful numbers are (most significant byte first):

pi =	2,201,15,218
1.0 =	1,128,0,0
−1.0 =	129,128,0,0
2.0 =	2,128,0,0
0 =	0,0,0,0

Thus the sequence of operations 26,26,3,25,3,17 should result in a zero on the top of the stack. Or 26,26,3,25,3,18 should result in a 1 there. The status register is also read and displayed.

The second program, when run, asks for a number between zero and 255. It writes this onto the top byte of the 8231 stack and then reads it. If what went in equals what comes back, the program asks for another number, otherwise an error message is printed out. With these two programs enough simple testing can be done to insure that the circuit is working correctly. With this hurdle completed we will be ready to look at the software aspects of the system as described in part two of this article, which will be presented next month.

Appendix

Error codes, Parts list, BASIC test programs, and APU op codes.

INTEL 8231 Error Codes (decimal values of status register)

128 or greater	busy, operation not completed
64	top-of-stack negative, no error
32	top-of-stack zero, no error
16	divide by zero
8	negative argument of function not allowed (e.g. square root)
24	argument of function too big (e.g. Arc Sine, Arc Cosine, exponential)
4	underflow, number $< 2.7 \times 10^{-20}$
2	overflow, number $> 9.2 \times 10^{18}$
0	non-negative, non-zero result, no errors

Parts List

1 Vector board (at least 6" × 6")
1 40-pin wire-wrap socket
2 24-pin sockets
7 14-pin sockets (including 1 for connection to Superboard)
3 16-pin sockets
11 .01 disk capacitors (bypass)
1 80pf capacitor
1 2.2K resistor
2 7404 hex inverters
1 7402 quad NOR gate
1 7410 tri, three input NAND gate
1 7430 8 input NAND gate
1 74LS76 edge trigger flip-flop
1 74LS123 re-triggerable one shot
1 74154 4- to 16-line demultiplexer
2 8T28 tri-state buffers
1 INTELC8231 arithmetic processing unit
Ribbon cable and connectors (40 and 14 wire)

MICRO

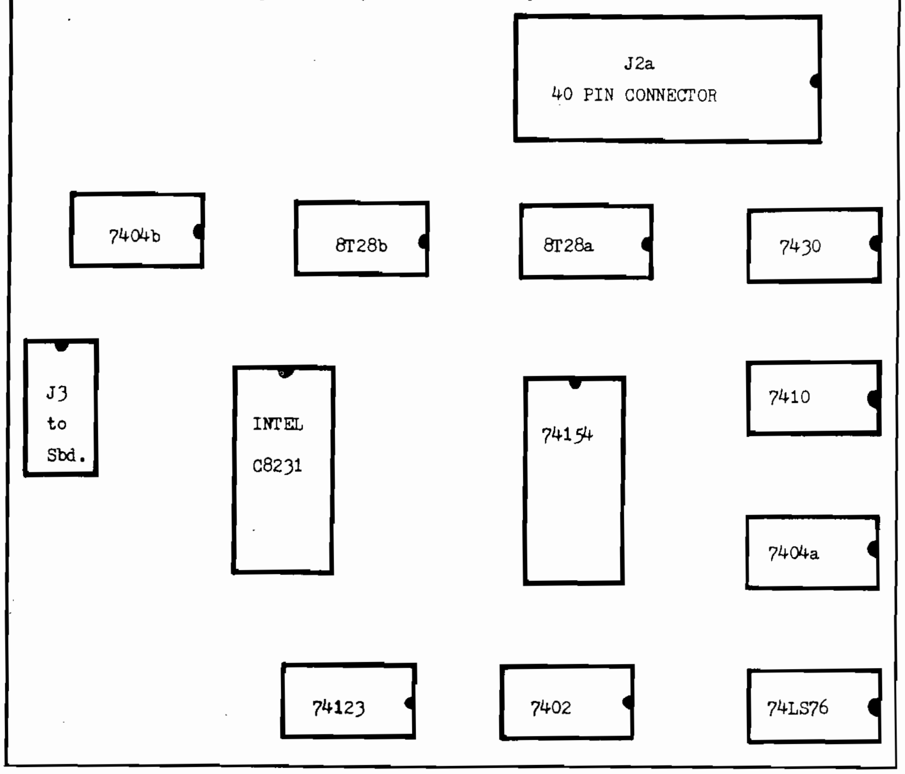


***Figure 5:* Layout — Wirewrap Side of Board**

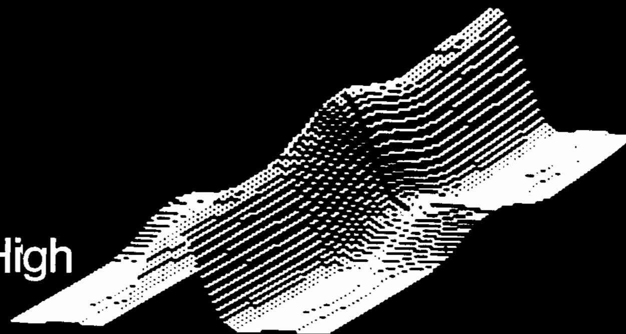
Listing 1

```
1  REM  APU TEST 1
2  REM  ENTER OPERATION COMMAND NUMBER
3  REM  STACK IS PRINTED FROM TOP DOWN.
 STACK HOLDS 4,4-BYTE FLT NMBRS.
9  INPUT "COMMAND";Y: POKE 64263,Y
10 A = 64257:B = 64262: PRINT : PRINT
11  PRINT "FOR COMMAND CODE=";Y
17 X =  PEEK (A - 1): PRINT "STATUS=";
PEEK (B)
20  FOR J = 1 TO 16:X =  PEEK (A): PRIN
T  PEEK (B)
25  NEXT J
27  GOTO 9
```

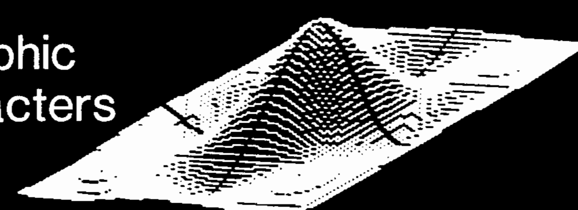
Listing 2

```
1  REM  APU TEST 2
2  REM  ENTER INPUT BYTE BETWEEN ZERO A
ND 255
3  REM  POKE TO APU, THEN READ. IF EQUA
L, OK.
10  INPUT "X=";X
12  POKE 64262,X: REM  WRITE OPERAND ON
 TOP OF APU STACK
15 Y =  PEEK (64257): REM  OPERAND READ
 START
16 Y =  PEEK (64262): REM  READ DATA
20  IF Y <  > X THEN  PRINT "APU R/W ER
ROR": PRINT "X=";X;"Y=";Y
22  IF Y = X THEN  PRINT "R/W OK"
25  GOTO 10
```

GRAPHICS
FOR OSI COMPUTERS

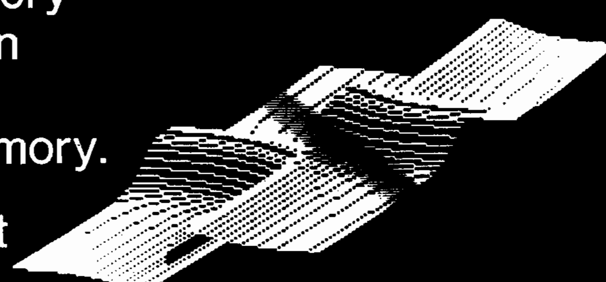


☆ You Can Produce The Images Shown Or Yours And Program Motion With Our 256 By 256 High Resolution Graphics Kit. Thats 65,536 Individually Controlled Points On Your TV Screen.



☆ 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.

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

MICRO
PET Vet

By Loren Wright

HESLISTER

The most efficient way to enter a BASIC listing is shown in listing 1. Multiple statements on a line make execution faster, and the lack of spaces makes the program occupy considerably less memory. These listings are difficult to read, let alone understand. Do you remember which reverse field characters represent which cursor controls?

Listing 2 is the same set of lines as output by HESLISTER. Spaces have been inserted and multiple statements appear on separate lines. The cursor control characters appear as two-letter abbreviations within brackets. Also, IF...THEN and FOR...NEXT structures are indented appropriately. Since PET programs on cassette cannot be read as data, HESLISTER works only on disk. It is available for $9.95 from:

Human Engineered Software
3748 Inglewood Blvd., Rm. 11
Los Angeles, California 90066

VIGIL from Abacus Software

Many of us have contemplated writing interactive games for the PET, but have never gotten beyond the contemplation stage. Moving large objects across the screen with BASIC can be very slow, and it takes time to write and debug the required machine language routines. If you want the use of paddles or sound, further complication is added.

VIGIL, an acronym for Video Interactive Game Interpretative Language, is a new "language" offered by Abacus Software. A few simplifications have been made. Instead of BASIC variables, there are 26 registers which can have a value from 0 to 255. Normal input is only from 16 keys on the numeric keypad. Also, only one statement is allowed per program line and no spaces may be embedded in commands. Anything appearing after a space is treated as a comment and ignored.

The commands, in general, are very powerful. There are four "Test and Skip" commands and three "Step and Test" commands, which transfer program control depending on the value of a particular register. Control of PET's double resolution (or quarter-box) graphics is particularly easy. You can display a pattern at a specified *x*-, *y*-coordinate and erase it simply by repeating the display command. Whenever displaying a pattern overwrites another (as in a rocket hitting a plane!), the Z-register is affected. Messages and PET graphic characters are also displayed by specifying *x*-, *y*-coordinates.

Other features include sound (for a speaker hooked to CB2 of the parallel user port), timer control, key-testing, and variety of data movement and program control commands.

The VIGIL interpreter begins at $033A (826) and runs to $1300 (4864). Not much room is left for programs in an 8K machine, but there is still a lot that can be done. The tape (or disk) comes with nine sample programs: BREAKOUT, ANTI, SPACE WAR, SPACE BATTLE, U.F.O., CONCENTRATION, MAZE, KALEIDOSCOPE, and FORTUNE-TELLER. All these work with 8K, and they serve as good examples of different VIGIL programming techniques.

I also have a few complaints. Restricting input to the numeric keypad makes it awkward to play two-person games. Sometimes the speed is a little disappointing — not up to pure machine language speed, but certainly faster than pure BASIC. Finally, some of the commands are difficult to remember. For example, THEN prints a character string at a specified location and Z and B are "increment and test" commands. It does take a little experience to get really comfortable with VIGIL, or any new language. The documentation is very good, and a separate reference list of commands is provided.

VIGIL, complete with user's manual and sample program, is available on disk or cassette (for BASIC 3.0 only) for $35 from:

Abacus Software
P.O. Box 7211
Grand Rapids, Michigan 49510

October PET Bonus

The October MICRO will have a special PET bonus section — five or six articles. Features include "Growing Knowledge Trees" and "Character Set Substitution."

MICRO has Assemblers

MICRO has copies of HESBAL, MAE, and ASM/TED assemblers. We can accept articles with source files on disk or cassette in any of these formats.

Listing 1

```
165 IFT=ZTHENIFC$=";"THENIFM$=""THENS=88:GOTO210
2140 FORK=ZTOW:IFG$=LEFT$(L$(K),D)THENL=K:T$=MID$(L$(K),D+3,U):K=W
2145 NEXT:RETURN
3000 PRINT"[illegible]";
```

Listing 2

```
165   IF T=Z
      THEN IF C$=";"
        THEN IF M$=""
          THEN S=88:
            GOTO 210
2140 FOR K=Z TO W:
        IF G$=LEFT$(L$(K),D)
        THEN L=K:
          T$=MID$(L$(K),D+3,U):
          K=W
2145 NEXT :
     RETURN

3000 PRINT "[CH][CD][CD][CD][CD][CD][CR][CR][CR]";
```

It's Time to Stop Dreaming

Part 3

Robert M. Tripp
Editor/Publisher
MICRO

Part 1 of this series (MICRO 37:9) introduced the Motorola 6809 as a candidate for the 6502 "Dream Machine" and discussed its basic architecture and fundamental characteristics. Part 2 (MICRO 38:27) presented the details on several major features of the 6809, particularly the support for writing position-independent code (PIC) and the extensive stack operations. Part 3 describes the instruction set in detail using terms familiar to MICRO readers, by comparing it instruction-by-instruction to our beloved 6502.

Table 1 presents the entire 6809 instruction set, with the exception of the Branches, which are presented in table 2. The table lists the instructions by both the 6502 and 6809. A brief study of the table will show how similar the instruction sets are. Most of the instructions available on the 6502 are also available on the 6809. The standard mnemonics are even identical for the most part. If a particular instruction is not available on one or the other processor, this has been indicated in the table by "---."

Notes and comments about the instruction set from the 6502 point of view:

1. The Carry Flag is not treated identically on the two processors. On the 6502, the Carry Flag is Cleared to indicate a "borrow" and Set to indicate "no borrow." (Remember the SEC before an SBC?) On the 6809, the Carry Flag is Set to indicate a "borrow" and Cleared to indicate "no borrow." While this "reversal" may cause a little difficulty at first, it does make sense if you think about it. You can start all arithmetic operations with a Clear Carry (CLC) instruction.

   Since the sense of the Carry Flag is reversed on the "borrow/no borrow," a Compare instruction, followed by a BCC or BCS, will function differently on the 6502 and 6809. This should not cause any trouble since the 6809 offers additional Branches including Branch on Less (BLS), Branch on Low (BLO), which is actually identical to the Branch on Carry Set (BCS), and so forth. Since the BCC and BCS are normally used as "Branch on Less" types of operations after a Compare on the 6502, the inclusion of additional branches for these purposes on the 6809 is helpful.

2. The programmed setting and clearing of the Condition Codes or Flags is handled quite differently on the 6809, but can be treated as almost identical forms. The 6502 has separate instructions for each Clear and Set. The 6809 uses a single instruction for Clearing any number of Flags and another single instruction for Setting any number of Flags. Flags may be Cleared by the ANDCC instruction which is two bytes: the opcode, and the mask which determines which Flags will *not* be cleared. Flags may be Set by the ORCC instruction which is also two bytes: the opcode, and the mask which determines which Flags will be set.

   An SEI on the 6502 would be equivalent to ORCC #$10 on the 6809; a CLI would be ANDCC #$EF. Since the 6800 has a set of individual instructions for each Flag just like the 6502, many 6809 assemblers will accept the 6800/6502 form and assemble it for the 6809. For example, many 6809 assemblers will accept SEI as a mnemonic and generate the object code for an ORCC #$10.

3. The ASL and LSL instructions are actually one and the same on the 6809. The 6809 has simply provided two sets of mnemonics. The ASR and LSR, however, are *not* equivalent. The ASR shifts the most significant bit back into the most significant position, thereby extending the sign for the original byte. The LSR shifts a zero into the most significant bit.

4. The EXG and TFR instructions may be used between any two registers of the same size, (that is, between any two 8-bit registers or any 16-bit registers), but may not be used between an 8-bit and a 16-bit. Therefore, the following instructions which would be valid on the 6502 would not be valid on the 6809:

   TAX, TXA, TAY and TYA

5. The Push/Pull Stack operations on the 6502 require only one byte each. The Push/Pull Stack operations on the 6809 require two bytes, but can accomplish a lot more. On a single PSH, up to eight registers may be pushed. Which registers are to be pushed is specified in the second byte of the instruction. There is a fixed order in which registers are pushed onto the stack, and all of the registers may be pushed onto the stack, not just the A reg and Condition Codes as on the 6502. Similarly, a single PUL can pull one to eight registers. The order is: CC (Condition Codes) A B DP (Direct Page) X Y U or S PC.

6. There are two independent Stacks on the 6809. The "S" Stack is similar to the 6502 stack, except that it has a 16-bit pointer and can be anywhere in memory. The "U" (User) Stack has all of the same operations as the "S" Stack, but is *not* used for hardware interrupt and subroutine processing.

Table 1: 6502/6809 Instruction Comparison Table

6502	6809			Notes and Details
---	ABX			Add B Reg to X Reg
ADC	ADCA	ADCB		Add with Carry Bit
---	ADDA	ADDB	ADDD	Add without Carry Bit
AND	ANDA	ANDB		Logical AND
ASL ASLA	ASLA	ASLB	ASL	Arithmetic Shift Left
---	ASRA	ASRB	ASR	Arithmetic Shift Right
BRK	SWI	SWI2	SWI3	6809 has three Software Interrupts
BIT	BITA	BITB		Binary Bit Test
---	CLRA	CLRB	CLR	Clear: Set to Zero
CLC, CLI, CLV	ANDCC			Clear Condition Codes by ANDing
CMP	CMPA	CMPB	CMPD	Compare Reg to Memory
CPX	CMPX			Compare Index Reg to Memory
CPY	CMPY			Compare Index Reg to Memory
---	CMPS	CMPU		Compare Stack Reg to Memory
---	COMA	COMB	COM	One's Complement
---	DAA			Decimal Adjust replaces Decimal Mode
DEC	DECA	DECB	DEC	Decrement
DEX				(Part of Auto Decrement Index Mode)
DEY				(Part of Auto Decrement Index Mode)
EOR	EORA	EORB		Logical Exclusive OR
---	EXG	R1,R2		Exchange Specified Reg Contents
INC	INCA	INCB	INC	Increment
INX				(Part of Auto Increment Index Mode)
INY				(Part of Auto Increment Index Mode)
JMP	JMP			Jump to Address
JSR	JSR			Jump to Subroutine
LDA	LDA	LDB	LDD	Load Reg
LDX	LDX			Load Index Reg
LDY	LDY			Load Index Reg
---	LDS	LDU		Load Stack Reg
---	LEAX	LEAY	LEAS	Load Effective Address into Index Reg
---	LSLA	LSLB	LSL	Logical Shift Left
LSR LSRA	LSRA	LSRB	LSR	Logical Shift Right
---	MUL			Unsigned multiply: A*B=D
---	NEGA	NEGB	NEG	Two's Complement
NOP	NOP			No Operation
ORA	ORA	ORB		Logical OR
PHA,PHP	PSHS	PSHU		Push Specified Regs on Specified Stack
PLA,PLP	PULS	PULU		Pull Specified Regs from Specified Stack
ROL ROLA	ROLA	ROLB	ROL	Rotate Left
ROR RORA	RORA	RORB	ROR	Rotate Right
RTI	RTI			Return from Interrupt
RTS	RTS			Return from Subroutine
SBC	SBCA	SBCB		Subtract with Borrow
SEC,SED,SEI	ORCC			Set Condition Codes
---	SEX			Sign Extend B Reg into A Reg
STA	STA	STB	STD	Store Reg into Memory
STX	STX			Store Index Reg into Memory
STY	STY			Store Index Reg into Memory
---	STS	STU		Store Stack Reg into Memory
---	SUBA	SUBB	SUBD	Subtract without Borrow
TAX, TAY, TYA, TXA	---			Replaced by Transfer Instruction TFR
TSX, TXS	---			Use LDS/LDU, STS/STU, EXG or TFR
---	TSTA	TSTB	TST	Set Sign and Zero Condition Codes
---	T F R	R1,R2		Transfer Reg R1 to Reg R2

***Table 2:* Branch Instruction Comparison Table**

6502	6809		Branch Operation
	Simple Branches		
BCC	BCC	LBCC	Branch on Carry Clear
BCS	BCS	LBCS	Branch on Carry Set
BEQ	BEQ	LBEQ	Branch on Equal Zero
BNE	BNE	LBNE	Branch on Not Equal Zero
BMI	BMI	LBMI	Branch on Minus
BPL	BPL	LBPL	Branch on Plus
BVC	BVC	LBVC	Branch on Overflow Clear
BVS	BVS	LBVS	Branch on Overflow Set
	Signed Branches		
---	BGT	LBGT	Branch if Greater
---	BGE	LBGE	Branch if Greater or Equal
---	BLE	LBLE	Branch if Less or Equal
---	BLT	LBLT	Branch if Less
	Unsigned Branches		
---	BHI	LBHI	Branch if Higher
---	BHS	LBHS	Branch if Higher or Same
---	BLS	LBLS	Branch if Lower or Same
---	BLO	LBLO	Branch if Lower
	Other Branches		
---	BSR	LBSR	Branch to Subroutine
---	BRA	LBRA	Branch Always
---	BRN	LBRN	Branch Never !!!

Notes: The 6809 has two forms of each Branch. The "short form" is identical to that on the 6502, using a one-byte offset which permits it to branch only to locations within plus or minus 128 decimal bytes from the branch instruction. The "long form," preceded by an L in the table, uses a two-byte offset which permits it to branch directly to any location in a 64K memory.

7. The Clear instruction is simply a quicker way to load a zero into the A or B registers or into a memory location.

8. There are two complement instructions. COM performs a one's complement on the A or B register or memory. This simply complements each bit of the specified location. NEG performs a two's complement which is equivalent to a COM plus one. This makes the negative value of the original number.

9. On the 6809 you can simply increment or decrement the A and B registers with the INC and DEC commands. The 6502 requires a CLC, ADCIM #$01 for an INC on A or an SEC, SBCIM #$01 for a DEC on A. There is no specific INC or DEC for the X or Y registers, but this is normally handled in the auto-increment or auto-decrement indexed instruction modes.

10. The LEA (Load Effective Address) is a powerful addition to the 6809 which has no counterpart in the 6502. It is one of the features that really makes the 6809 a "dream machine," but it will take some getting used to.

11. The inclusion of three separate software interrupts, in place of the single BRK on the 6502 should not upset anyone. It should make error trapping, debugging, and other interrupt-driven operations, considerably simpler to write and use.

12. The 6502 requires that a two-byte address be provided in the form low byte/high byte. The 6809 uses the more natural form of high byte/low byte. At the Assembler level this does not make any difference, but at the Object level it does. All two-byte addresses on the 6809, including indirect addressing via tables, interrupt vectors, and so forth are high/low. Compare:

```
8D 34 12  STA $1234  on the 6502
B7 12 34  STA $1234  on the 6809
```

The two-byte address on the 6502 in object form is 34 12; on the 6809 it is 12 34.

This list may make it seem that there are a great number of differences between the 6502 and the 6809. The significant differences are actually quite minor, and in many cases the differences are in the direction of improved operations on the 6809. MICRO

Improved *n*th Precision

This article discusses code optimization for small systems, using Golla's add/subtract routines (MICRO 27:27) as an example.

Glenn R. Sogge
Fantasy Research & Development
P.O. Box 203
Evanston, Illinois 60204

This article began as a couple of short notes on ways to optimize the coding of machine language programs for the 6502. The article and program in the August, 1980 issue of MICRO (27:27) by Lawrence R. Golla presented two routines for multiple precision adding and subtracting. These routines were transparent as far as register contents were concerned and returned the correct information in the flags.

As I began the actual recoding of the routines to satisfy a couple of my pet optimizing prejudices, I discovered that the zero checking routine seemed overly complicated and slow. The resulting "optimized code" is a complete reworking of the status information code, with a few other goodies thrown in, that increase the execution speed and lower the memory requirements.

Relocation

The first step was to make the routines position-independent. Whenever I find a short, versatile routine, I try to adapt it for easy use in most situations without the time-consuming process of individual relocations. I believe that any short routine that can easily be coded with branch instructions (even if a two- or three-stage branch is required) is preferable to one that contains absolute jumps. The only exception to this is in code that is critically time-dependent; even then, alternate codings can often be used. I think it is preferable to recode a routine once and just load it where and when it is needed rather than having to remember which routines need which bytes changed. As the use of computers spreads through the public, I think it is the responsibility of programmers to make the use of their codes as easy as possible for the neophytes. Hand relocation of short routines is quite easy for someone with a little bit of programming experience but it is still not a conceptually trivial task.

A collection of routines coded this way can make up a very useful library that can be customized without the "big system" overhead of relocatable assemblers and linking loaders. Only as many of the system utilites as are needed get loaded into the machine.

Sometimes, the best way to improve a routine is not through the peephole optimization of small bits of code but by using a different algorithm. This kind of large-scale optimization is what really pays off in the long run. In these routines, I checked for a zero result in a very straight-forward and fast manner. The code begins (after the math is done) at MOUT by saving the C and V flags and assuming the result is probably not zero and that it is not negative. The code then starts checking the result bytes from lowest to highest. As soon as a non-zero byte appears, it exits this check code and leaves the Z flag at 0 (i.e., it found something to prove its assumption). Only as many bytes are checked as are necessary to prove this assumption; this might range from 1 to 128 but it only checks all 128 (unlike Golla's routine) if it has to. If the result does turn out to be zero, only then does it go through the Z flag machinations.

A similar logic is used for the N flag. It is assumed to be positive and changed only if this assumption is not true. A peephole technique was used to save the C and V flags and clear the N and Z flags with one instruction — the AND #$7D just after MOUT followed by the saving of this status on the stack (actually IN the stack).

Playing with the Stack

A big advantage of a hardware stack is the "free" temporary storage it provides. In the 6502, this chunk of memory is hardware address dedicated and rarely gets used for anything else. With a proper understanding of how to access this area, another page of temporary scratchpad RAM is available to the user. This can be important in small systems with small memories or in big systems whose software grabs all the page zero locations it can find.

Another advantage of accessing the stack memory is that the addresses need not be hard-coded in the software. It is possible to write everything relative to the current stack pointer and the hardware will do the translation into the proper bits on the address bus. This creates a very small virtually-mapped memory. Location $4 relative to the stack pointer might be a different physical address every time the instruction is executed but the logical space is always the same.

In my recoding of the math routines, I used this technique for only one of the locations — the flags to be passed back to the calling routine. This ensures that that data will not be accidentally clobbered by the stack as might happen with Golla's use of locations $100 and $101; it also avoids the problems of selecting another address (page zero or elsewhere) that would conflict with locations used by other systems' hardware and software.

There is, unfortunately, no way to locate the pointers in equally flexible locations; if these locations conflict with others in the user's system, the code will have to be changed. Unlike the more advanced chip designs that make all kinds of relocation easy (data and programs), such as the 6809, we have to sacrifice some flexibility for the speed and size savings possible with the 6502's instruction set.

When data is pushed on the 6502's stack, the stack pointer determines where the storage address is on the page (most systems have the stack at $100-$1FF, although it is possible to put the stack at $0-FF with some 6502 designs). After storing the byte, the stack pointer is decremented (the stack grows downward) and points to the next available location. By transferring the stack pointer to the X-register (which we've already saved or don't care about), we can absolute index into this page as normal memory.

Examples:

next free	$100,X
top of stack	$101,X
second on stack	$102,X
third on stack	$103,X
fourth on stack	$104,X

One problem with this technique is the lack of wrap around. Unlike the page zero,X mode, the resulting addresses do *not* wraparound to the beginning of the page. If the base address you are using plus the stack pointer offset sums to more than $1FF, you'll end up indexing into the $200-$2FF page. This is not likely to happen if the stack pointer gets initialized to the top of the page — like $FF — and you know the stack won't grow all the way down and wrap around. If it does, however, you may end up with a situation where your base address is $110 (from passing lots of parameters before a subroutine call) and the stack pointer is $F8. The resulting address is $208, not $108. As I said, this is not likely to happen unless the stack pointer is never initialized to a known value. Some systems may not initialize the pointer because it is restricted by hardware to the $100-$1FF range; the "unknown stack" or "no RAM stack" conditions of other processors cannot happen and the initialization step might be skipped. User programs should either initialize the stack or be sure of its ranges before using the technique outlined here.

The actual use of this technique in math routines is straightforward. Space is allocated for the returning flags by saving the caller's flags upon entry. The byte at this "semi-absolute address" is then modified according to the results of the math routines and passed back to the caller by popping them off the stack at the end of the code.

Notice that no flags other than the ones used by the routine are altered before they are passed back. The interrupt mask, the break flag, and the decimal flag in effect at entry time will be restored upon exit. Thus, this binary

Listing 1

```
                        *******************************
                        *
                        * LAWRENCE R. GOLLA'S ORIGINAL
                        * NTH PRECISION ROUTINES AS
                        * PUBLISHED IN MICRO 8/80
                        * PAGES 27-29
                        *
                        *******************************
                        *
                                SYM
                        PTR1    EQU   $10
                        PTR2    EQU   $12
                        PTR3    EQU   $14
                        PREC    EQU   $16
                        AEND    EQU   PTR1
                        AGAND   EQU   PTR2
                                ORG   $4000
                                OBJ   $4000
                        *
4000: 48                ADD     PHA
4001: 98                        TYA
4002: 48                        PHA
4003: 8A                        TXA
4004: 48                        PHA
4005: A4 16                     LDY   PREC
4007: 18                        CLC
4008: D8                        CLD
4009: B8                        CLV
400A: B1 10             LOOP1   LDA   (AEND),Y
400C: 71 12                     ADC   (AGAND),Y
400E: 91 14                     STA   (PTR3),Y
4010: 88                        DEY
4011: 10 F7                     BPL   LOOP1
4013: 30 13                     BMI   OUT
                        *
4015: 48                SUB     PHA
4016: 98                        TYA
4017: 48                        PHA
4018: 8A                        TXA
4019: 48                        PHA
401A: A4 16                     LDY   PREC
401C: D8                        CLD
401D: 38                        SEC
401E: B8                        CLV
401F: B1 10             LOOP3   LDA   (AEND),Y
4021: F1 12                     SBC   (AGAND),Y
4023: 91 14                     STA   (PTR3),Y
4025: 88                        DEY
4026: 10 F7                     BPL   LOOP3
                        *
4028: A4 16             OUT     LDY   PREC
402A: A9 00                     LDA   #$00
402C: 51 14             LOOP2   EOR   (PTR3),Y
402E: 08                        PHP
402F: 30 07                     BMI   NZER
4031: 88                LOOP4   DEY
4032: 30 0B                     BMI   OUT1
4034: 28                        PLP
4035: 4C 2C 40                  JMP   LOOP2
4038: 28                NZER    PLP
4039: 09 01                     ORA   #$01      SET Z=0
403B: 08                        PHP
403C: 4C 31 40                  JMP   LOOP4
                        *
403F: 68                OUT1    PLA
4040: 29 7F                     AND   #$7F
4042: 8D 00 01                  STA   $100
4045: C8                        INY
4046: B1 14                     LDA   (PTR3),Y
4048: 49 00                     EOR   #$00      ADJUST N-FLAG
404A: 08                        PHP
404B: 68                        PLA
404C: 29 80                     AND   #$80
404E: 0D 00 01                  ORA   $100      ADD TO FLAGS
4051: 8D 00 01                  STA   $100
4054: 68                        PLA
4055: AA                        TAX
4056: 68                        PLA
4057: A8                        TAY
4058: 68                        PLA
4059: 8D 01 01          RESET   STA   $101
405C: AD 00 01                  LDA   $100      GET STATUS
405F: 48                        PHA
4060: AD 01 01                  LDA   $101
4063: 28                        PLP
4064: 60                        RTS
```

Listing 2

```
                           ********************************
                           *
                           * NTH PRECISION ROUTINES AS
                           * MODIFIED BY GLENN R. SOGGE
                           * FANTASY RESEARCH & DEVELOPMENT
                           * P.O. BOX 203
                           * EVANSTON, IL 60204
                           *
                           * AUGUST 7, 1980
                           *
                           ********************************
                           *
                           STACK    EQU  $100
                           STKLOC   EQU  STACK+4
                           *
                                    ORG  $4100
                                    OBJ  $4100
                           *
4100: 18                   MADD     CLC
4101: B0 38                         BCS  *+$3A       HIDES 'SEC' ($38)
                           MSUB     EQU  *-1
                           *
4103: 08                            PHP  SAVE        ALL THE REGISTERS
4104: 48                            PHA  INCLUDING   ROOM FOR THE STATUS
4105: 8A                            TXA
4106: 48                            PHA
4107: 98                            TYA
4108: 48                            PHA
4109: D8                            CLD
410A: B8                            CLV
410B: A4 16                         LDY  PREC
410D: B0 0B                         BCS  MSUB1       C STILL SET FROM ENTRY
                           *
410F: B1 10                MADD1    LDA  (PTR1),Y
4111: 71 12                         ADC  (PTR2),Y
4113: 91 14                         STA  (PTR3),Y
4115: 88                            DEY
4116: 10 F7                         BPL  MADD1
4118: 30 09                         BMI  MOUT
                           *
411A: B1 10                MSUB1    LDA  (PTR1),Y
411C: F1 12                         SBC  (PTR2),Y
411E: 91 14                         STA  (PTR3),Y
4120: 88                            DEY
4121: 10 F7                         BPL  MSUB1
                           *
4123: 08                   MOUT     PHP
4124: 68                            PLA  RESET       N & Z (=0) BUT
4125: 29 7D                         AND  #$7D        SAVE C & V
4127: BA                            TSX  GET         POINTER TO STASH
4128: 9D 04 01                      STA  STKLOC,X    STORE IN ORIGINAL P SAVED
412B: A4 16                         LDY  PREC
                           *
412D: B1 14                ZCHK     LDA  (PTR3),Y
412F: D0 0B                         BNE  NCHK        LEAVE AS SOON AS FIND <>0
4131: 88                            DEY
4132: 10 F9                         BPL  ZCHK        KEEP LOOKING
4134: BD 04 01             ZFLG     LDA  STKLOC,X    X STILL SET
4137: 09 02                         ORA  #$02        MAKE Z=1
4139: 9D 04 01                      STA  STKLOC,X
                           *
413C: A0 00                NCHK     LDY  #$00
413E: B1 14                         LDA  (PTR3),Y
4140: 10 08                         BPL  EXIT        LEAVE N=0
4142: BD 04 01             NFLG     LDA  STKLOC,X
4145: 09 80                         ORA  #$80
4147: 9D 04 01                      STA  STKLOC,X    MAKE N=1
                           *
414A: 68                   EXIT     PLA
414B: A8                            TAY
414C: 68                            PLA
414D: AA                            TAX
414E: 68                            PLA
414F: 28                            PLP  PULL        FLAGS AS MODIFIED
4150: 60                            RTS  AND         EXEUNT
                           *
                                    ORG  $4200
                                    OBJ  $4200
                           *
```

math routine could be called by a decimal math program and not interfere with the main program. (Interpreting the results is another matter.)

(A modification of these routines would be to NOP the CLD instruction to allow the code to work in whichever base was in effect for the calling program or to change the CLD to a SED for decimal operands and results. The N and V flags will *not* be correct if decimal is the base in effect when the code runs, but the answers and the C and Z flags will still be right.)

Code Sharing and Duplication

The original routines duplicate quite a bit of set-up code at their begir nings (saving registers, clearing flags getting the precision, etc.). In fact, th only differences are in the setting of th carry flag. By setting the carry flag appro priately as the first action upon entry, th duplicate code can be shared and thei branched out of on the basis of the carr — if it's clear, add; it it's set, subtract

The very first bytes are a trick technique I picked up from some of th Apple peripheral card firmware. Entry a the first byte clears the carry and thei encounters a branch instruction it wil never take (BCS — branch if set) an falls through into the main code. Th second byte of the branch instructio contains the value of the SEC opcod ($38 — the value in the source listing i necessary to get my assembler to cal culate the correct value). Entering a this third byte will set the carry an then fall into the common code. The en try points are Origin + $00 for addin and Origin + $02 for subtracting. (I fin close entry points easier to remembe than ones spaced farther apart.)

This bit of trickery saves one byte o code that could be crucial in a small ROM driver by compressing a sequence like

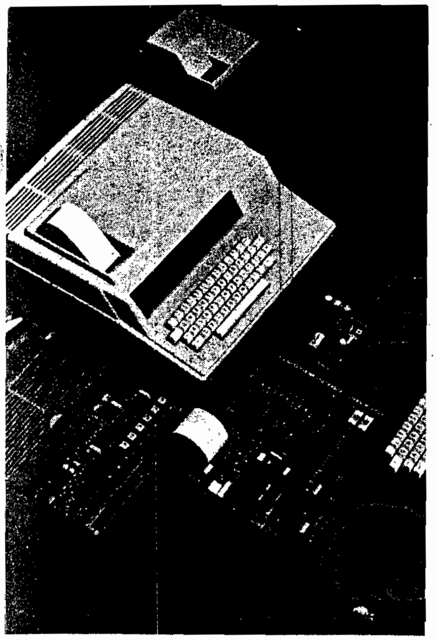
```
ENTRY1 CLC
       BCC MAIN
ENTRY2 SEC
MAIN   ...
       ...
```

of 4 bytes into 3 bytes. In addition, assuming the flag doesn't get modified by the main code, selective initialization or function selection is possible further down the road.

What We Have Gained

All of this is only of theoretical interest if there isn't some practical result. The clearest gain is a reduction of memory size from 101 bytes to 81 bytes without any loss of function and an increase in portability. There is also an improvement in speed but this isn't quite as clear-cut.

The test routines included in the listings were some of the code and conditions I used for quantifying the results. In the examples given, one of the worst case situations is executed. Two 128-byte zeros are added together, checked for a zero result, and the flags appropriately set. This is done 256 times before hitting the BRK's. With Golla's code, each of the 256 adds takes about



Listing 3

```
                             ********************************
                             *
                             * TESTING ROUTINES
                             *
                             ********************************
                             *
4200: A2 43                  SETPTRS   LDX  #$43
4202: 86 11                            STX  PTR1+1
4204: E8                               INX
4205: 86 13                            STX  PTR2+1
4207: E8                               INX
4208: 86 15                            STX  PTR3+1
420A: A0 00                            LDY  #$00
420C: 84 10                            STY  PTR1
420E: 84 12                            STY  PTR2
4210: 84 14                            STY  PTR3
4212: A9 7F                            LDA  #$7F       MAXIMUM PRECISION
4214: 85 16                            STA  PREC
4216: A9 00                            LDA  #$00
4218: A0 00                            LDY  #$00
421A: 91 10                  CLRLOOP   STA  (PTR1),Y
421C: 91 12                            STA  (PTR2),Y
421E: 91 14                            STA  (PTR3),Y
4220: C8                               INY
4221: 10 F7                            BPL  CLRLOOP
4223: 60                               RTS
                             *
4224: 20 00 42               ADDTST    JSR  SETPTRS    NULL EVERYTHING
4227: A2 00                            LDX  #$00
4229: 20 00 40               ADLP1     JSR  ADD
422C: CA                               DEX
422D: D0 FA                            BNE  ADLP1      ADD 0 TO ITSELF 256 TIMES
422F: 00                               BRK
                             *
4230: 20 00 42               ADDT2     JSR  SETPTRS
4233: A2 00                            LDX  #$00
4235: 20 00 41               ADLP2     JSR  MADD
4238: CA                               DEX
4239: D0 FA                            BNE  ADLP2      SAME AS ABOVE
423B: 00                               BRK
                             *

60 BYTES GENERATED THIS ASSEMBLY
```

.0059 seconds (5.9 milliseconds); with my code, each takes about .0049 seconds (4.9 milliseconds). (The multiple execution was to allow stopwatch timing to at least be in the ball park.) For these cases, all of the bytes of the result had to be examined before the zero flag could be properly set.

As a further test of the differences between the routines, I set them up to add zero and 1 (both 128-byte precision). Here the differences were much more substantial — Golla's code still took around 6 milliseconds per result while mine ran in about 3.3 milliseconds. This shows the effect of changing the algorithm because the code is almost identical except for checking the result for zero.

The rewritten code runs at times that are proportional to both the amount of precision and the result but the original code runs at speeds only proportional to the precision.

When and What to Optimize

As I said at the beginning, this article started out as a few thoughts about optimizing; obviously it's expanded considerably. Golla's routines seemed like a good place to illustrate some of the techniques and results of optimization. Not all code can be optimized in these ways and some shouldn't be. Saving three bytes and 15 microseconds is not important if you have 4K of extra RAM and the routine is dependent on user reaction time — the sweat just isn't worth it.

These math routines were good candidates though because the optimization worked on the loops where most of the execution time is spent. With the size of the code, tools should only be big enough to do their job (if they're too big, you may have to exclude another useful tool from your program). Tools like these routines should be optimized because they are likely to be used more often than their size would indicate. Number-crunching is slow enough as it is; the design of the code shouldn't impede it even more.

Some analysts estimate that 80% of the execution time is spent in 20% of the code. That 20% is where the optimization should be done.

Glenn R. Sogge is a 30 year old former composer with a degree in Art and 7½ years of retail business experience. He has become fascinated and infatuated with those electronic crossword puzzles that are called computers.

MICRO

Disassembling to Memory on AIM 65

This program lets you direct disassembled code to the AIM Editor's Text buffer for clean-up so that it can serve as input to the AIM Assembler.

Larry P. Gonzalez
Dept. of Physiology and Biophysics
University of Illinois Medical Center
P.O. Box 6998
Chicago, Illinois 60680

The disassemble command ("K") provided by the AIM 65 monitor is a useful aid to program debugging. This command disassembles object code from memory into mnemonic instruction codes, which are output to the display/printer (d/p) along with the instruction address, hex opcode, and any operand. The usefulness of instruction disassembly can be significantly increased by a modification of the monitor routines which allows the disassembled code to be stored in memory as well as output to the d/p. Since the output of the disassembler is in ASCII format, disassembly to memory provides the object code in a form accessible to both the AIM Text Editor and the Assembler.

Once the disassembled code can be accessed by the Editor, it can be modified with much greater ease. This is particularly advantageous when it is necessary to insert a new instruction into the main body of a set of object code. Normally this involves re-entering all of the code below the new instruction. If, however, the object program is disassembled to memory, the Editor can perform the insertion with relative ease; address modifications can also then be done with the Editor.

The idea for the program that I present here is from a program which appeared in the first issue of *The Target*.

***Figure 1:* Assembly listing: disassembling to memory.**

```
                 ;* DISASSEMBLING TO MEMORY
                 ;*
                 ;*    BY L.P. GONZALEZ
                 ;*
                 TOLO   EPZ $00
                 TOHI   EPZ $01
                 BOTLN  EPZ $E1          ;LAST ACTIVE LINE
                 TEXT   EPZ $E3          ;BEGIN TEXT BUFFER
                 END    EPZ $E5          ;TEXT BUFFER END
                 ;
                 COUNT  EQU $A419
                 ADDR   EQU $A41C
                 PRTBUF EQU $A460
                 M1     EQU $E000        ;MONITOR MSGS
                 M5     EQU $E01C        ;'MORE?'
                 EMSGA  EQU $E06C        ;'EDITOR'
                 EMSGB  EQU $E072        ;'.END'
                 ;
                 ;SUBROUTINE ADDRESSES
                 ;
                 START  EQU $E182        ;MONITOR ENTRY
                 DONE   EQU $E790
                 FROM   EQU $E7A3
                 TO     EQU $E7A7
                 KEP    EQU $E7AF
                 PSL1   EQU $E837
                 BLANK  EQU $E83E
                 KEPR   EQU $E970
                 CRLOW  EQU $EA13
                 CRCK   EQU $EA24
                 RD2    EQU $EA5D
                 ADDIN  EQU $EAAE
                 DISASM EQU $F46C
                 ;
0E00                    ORG $E00
0E00
0E00             ;
0E00             ;READ AND STORE PARAMETERS
0E00             ;
0E00 20A7E7             JSR TO           ;READ BUFFER START
0E03 AD1CA4             LDA ADDR
0E06 8500               STA TOLO
0E08 85E3               STA TEXT
0E0A AD1DA4             LDA ADDR+1
0E0D 8501               STA TOHI
0E0F 85E4               STA TEXT+1
0E11             ;
0E11             ;READ BUFFER END AND DEC TO
0E11             ;ALLOW FOR TEXT END CHARACTER
0E11             ;
0E11 2013EA             JSR CRLOW
0E14 A06C               LDY #EMSGA-M1
0E16 20AFE7             JSR KEP
0E19 203EE8             JSR BLANK
0E1C A072               LDY #EMSGB-M1
0E1E 20AFE7             JSR KEP
0E21 20AEEA             JSR ADDIN
0E24 AD1CA4             LDA ADDR
0E27 85E5               STA END
0E29 AD1DA4             LDA ADDR+1
```

(Continued)

The program sent disassembled instructions to a VIA port. Since I wanted to be able to edit and re-assemble the disassembled code, my program disassembles one-instruction-at-a-time, reads the print buffer, and writes the ASCII instruction code and operand to specified memory locations. Then, the Text Editor can be entered to allow listing or modification of the source code. The resulting file contains a source program which can serve as input to the Assembler.

The first line of the generated source file is an assembly language command which sets the program counter to the original location of the object code. The remainder of the file contains lines of the symbolic instruction codes and operands in Assembler-compatible format. The instruction address and hex opcode, contained in the original output of the disassembler, are deleted, while the mnemonic instruction code and any operands are retained. Each line is terminated with a carriage return character ($0D) and the entire file is terminated with the Assembler ".END" directive and the Editor's text-end character ($00).

Since the disassembler outputs operands in hexadecimal format without the hex symbol ($), this symbol is added where appropriate. Also, the accumulator addressing mode is indicated by ".A" on the initial disassembled output. The "." is removed from the final output file to allow subsequent input to the Assembler.

The assembly listing and symbol table for this program are presented in listings 1 and 2. The program can be relocated by simply changing the program origin.

Executing the Program

When the program is executed, "TO=" is displayed. The beginning location for storage of disassembled code should be entered; this will be the beginning of the Editor text buffer. The user is then requested by the program to enter the "EDITOR END" which is the ending address for the Editor text buffer. Next, the beginning location of the code to be disassembled is entered in response to the displayed message "FROM=". Finally, enter the number of instructions to be disassembled (two digit decimal number; return, space, or "." = 01 instruction). After disassembly of up to 99 (decimal) instructions, the message "MORE?" will be displayed. The user can enter "Y" to continue disassembling, or enter any other character to quit.

```
0E2C 85E6             STA END+1
0E2E 38               SEC
0E2F A5E5             LDA END
0E31 E901             SBC #$01
0E33 85E5             STA END
0E35 B002             BCS CNTINU
0E37 C6E6             DEC END+1
0E39 2013EA    CNTINU JSR CRLOW
0E3C 20A3E7           JSR FROM          ;DISASSEMBLE WHERE?
0E3F           ;
0E3F           ;SET UP PROGRAM ORIGIN
0E3F           ;
0E3F A92A             LDA '*
0E41 20100F           JSR ADINC
0E44 A93D             LDA '=
0E46 20100F           JSR ADINC
0E49 A924             LDA '$
0E4B 20100F           JSR ADINC
0E4E AD1DA4           LDA ADDR+1
0E51 20FC0E           JSR TOASCI
0E54 AD1CA4           LDA ADDR
0E57 20FC0E           JSR TOASCI
0E5A A90D             LDA #$0D
0E5C 20100F           JSR ADINC
0E5F           ;
0E5F 20D7E5           JSR $E5D7         ;SAVE ADDRESS FOR DISASSEMBLER
0E62           ;
0E62           ;READ # OF INSTRUCTIONS (DECIMAL 1-99)
0E62           ;
0E62 2037E8    HOWMNY JSR PSL1
0E65 205DEA           JSR RD2
0E68 B0F8             BCS HOWMNY
0E6A 48               PHA
0E6B 2024EA           JSR CRCK
0E6E           ;
0E6E           ;DISASSEMBLE ONE INSTRUCTION
0E6E           ;
0E6E A901      DIS1   LDA #$01
0E70 8D19A4           STA COUNT
0E73 206CF4           JSR DISASM
0E76           ;
0E76           ;SKIP PC AND OP CODE
0E76           ;
0E76 A209             LDX #$09
0E78 BD60A4    RDBUF  LDA PRTBUF,X
0E7B E00C             CPX #$0C
0E7D           ;PUT BLANK BETWEEN MNEMONIC AND ADDRESS--SKIP OTHER BLANKS
0E7D F018             BEQ STORE
0E7F B005             BCS SPACE
0E81 297F             AND #$7F          ;STRIP MSB FROM MNEMONIC
0E83 4C970E           JMP STORE
0E86 C920      SPACE  CMP #$20
0E88 F026             BEQ NEXTX
0E8A E00D             CPX #$0D          ;CHECK FOR ADDRESS FIELD
0E8C D009             BNE STORE
0E8E C923             CMP #$23          ;IF '#', STORE IT AND STORE HEX SYMBOL
0E90 D00B             BNE PAREN
0E92 20100F    HXSYM  JSR ADINC
0E95 A924             LDA '$
0E97 20100F    STORE  JSR ADINC
0E9A 4CB00E           JMP NEXTX
0E9D           ;
0E9D           ;IF '(' STORE IT AND STORE HEX SYMBOL
0E9D           ;
0E9D C928      PAREN  CMP '(
0E9F F0F1             BEQ HXSYM
0EA1 C92E             CMP #$2E          ;SKIP IF '.'
0EA3 F00B             BEQ NEXTX
0EA5           ;
0EA5           ;NOT'#','.', OR '('--
0EA5           ;MUST BE ADDRESS, SO
0EA5           ;STORE HEX SYMBOL FIRST.
0EA5           ;
0EA5 A924             LDA '$
0EA7 20100F           JSR ADINC
0EAA BD60A4           LDA PRTBUF,X
0EAD 4C970E           JMP STORE
0EB0 E8        NEXTX  INX
0EB1 E014             CPX #$14
0EB3 D0C3             BNE RDBUF
0EB5 A90D             LDA #$0D          ;OUTPUT CR AS LAST CHARACTER
0EB7 20100F           JSR ADINC
0EBA 202EE7           JSR $E72E
```

(Continued

```
0EBD                ;
0EBD                ;ARE WE DONE?
0EBD                ;
0EBD 68                     PLA
0EBE 8D19A4                 STA COUNT
0EC1 2090E7                 JSR DONE
0EC4 48                     PHA
0EC5 D0A7                   BNE DIS1
0EC7                ;
0EC7                ;DISASSEMBLE MORE?
0EC7                ;
0EC7 A01C                   LDY #M5-M1          ;MORE?
0EC9 2070E9                 JSR KEPR
0ECC C959                   CMP 'Y
0ECE D003                   BNE ADDEND
0ED0 4C620E                 JMP HOWMNY
0ED3                ;
0ED3                ;ADD '.END'
0ED3                ;
0ED3 2013EA         ADDEND JSR CRLOW
0ED6 A200                   LDX #$00
0ED8 BD3D0F         ENDING LDA MESG,X
0EDB 20100F                 JSR ADINC
0EDE E003                   CPX #$03
0EE0 F004                   BEQ FINISH
0EE2 E8                     INX
0EE3 4CD80E                 JMP ENDING
0EE6                ;
0EE6                ;CLOSE FILE, RECORD BOTTOM LINE
0EE6                ;AND ENTER MONITOR
0EE6                ;
0EE6 A90D           FINISH LDA #$0D
0EE8 20100F                 JSR ADINC
0EEB A900                   LDA #$00
0EED A000                   LDY #$00
0EEF 9100                   STA (TOLO),Y
0EF1 A500                   LDA TOLO
0EF3 85E1                   STA BOTLN
0EF5 A501                   LDA TOHI
0EF7 85E2                   STA BOTLN+1
0EF9 4C82E1                 JMP START
0EFC                ;
0EFC                ;CONVERT 2 HEX CHARACTERS TO ASCII
0EFC                ;
0EFC 48             TOASCI PHA
0EFD 4A                     LSR
0EFE 4A                     LSR
0EFF 4A                     LSR
0F00 4A                     LSR
0F01 20070F                 JSR CNVRT
0F04 68                     PLA
0F05 290F                   AND #$0F
0F07 18             CNVRT  CLC
0F08 6930                   ADC '0
0F0A C93A                   CMP #$3A            ;'9' + 1
0F0C 9002                   BCC ADINC
0F0E 6906                   ADC #$06
0F10                ;
0F10                ;STORE CHAR AND INC ADDRESS
0F10                ;
0F10 A000           ADINC  LDY #$00
0F12 9100                   STA (TOLO),Y
0F14 E600                   INC TOLO
0F16 D002                   BNE TEST
0F18 E601                   INC TOHI
0F1A A500           TEST   LDA TOLO
0F1C C5E5                   CMP END
0F1E D01C                   BNE RETURN
0F20 A501                   LDA TOHI
0F22 C5E6                   CMP END+1
0F24 D016                   BNE RETURN
0F26 2013EA                 JSR CRLOW
0F29 203EE8                 JSR BLANK
0F2C A06C                   LDY #EMSGA-M1
0F2E 20AFE7                 JSR KEP
0F31 203EE8                 JSR BLANK
0F34 A072                   LDY #EMSGB-M1
0F36 20AFE7                 JSR KEP
0F39 4CE60E                 JMP FINISH
0F3C 60             RETURN RTS
0F3D                ;
0F3D 2E454E         MESG   ASC '.END'
0F40 44
```

When disassembly is complete, or when the text buffer is filled, the buffer limits and last active line parameters are set up for the Editor, and the program control jumps to the AIM monitor. The user can then enter the Editor with the monitor "T" command to examine and edit the generated source file, and then use this file as input to the Assembler. If the text buffer becomes filled during disassembly, disassembly stops, the message "EDITOR END" is displayed, and the monitor is entered.

I have found this program to be particularly useful for accessing and editing sections of code from the AIM monitor ROM for inclusion in my programs. Listing 1 presents a sample run of my disassemble-to-memory program with the disassembly of a short monitor routine. The listing includes the output of the AIM disassembler during program execution, followed by an editor listing of the generated source file.

This program can be used any time it is necessary to alter a program which is available only in object code. As such, Disassembling-To-Memory is a useful utility for AIM microcomputer systems.

***Figure 2:* Sample run of the disassembling to memory program. Prior to execution the AIM printer was toggled to "ON", so that the listing includes the program dialogue and the output of the AIM disassembler. This is followed by an entry to the AIM Editor with the "T" command and a listing of the program generated source file.**

```
   *  =0E00
  0  /
TO=0000
EDITOR END=0D00
FROM=EA46
/10
 EA46 48 PHA
 EA47 4A LSR A
 EA48 4A LSR A "
 EA49 4A LSR A "
 EA4A 4A LSR A "
 EA4B 20 JSR EA51
 EA4E 68 PLA
 EA4F 29 AND #0F
 EA51 18 CLC
 EA52 69 ADC #30
MORE?Y 04
 EA54 C9 CMP #3A
 EA56 90 BCC EA5A
 EA58 69 ADC #06
 EA5A 4C JMP E9BC
MORE?N
  T
*=$EA46
=   L
```

(Continued)

Figure 2 *(Continued)*

```
/
OUT=R
*=$EA46
PHA
LSR A
LSR A
LSR A
LSR A
JSR $EA51
PLA
AND #$0F
CLC
ADD #$30
CMP #$3A
BCC $EA5A
ADD #$06
JMP $E9BC
END
```

Larry Gonzalez is an Assistant Professor of physiology and biophysics at the University of Illinois Medical Center. He has 12 years of programming experience in high-level languages and several years in the use of minicomputers for real-time data acquisition and signal analysis. During the last two years he has been developing a system using an AIM 65 in the collection and analysis of electrophysiological data.

MICRO

Sorting

An application of Quicksort to sort a file where the individual members cannot be moved. The indexes of the individual members are moved to implement the sort.

William R. Reese
6148 Persimmon Tree Court
Englewood, Ohio 45322

Listing 1

```
10   REM QUICKSORT FOR INDEXES
20   REM  QUICK SORT P26:28 MICRO JULY 1980
30   REM PRINT LINES 162 & 185 MAY BE REMOVED
90   INPUT "NUMBER TO BE SORTED:  ";N
94   DIM SK(20)
95   DIM V$(N + 1),SS%(N + 1)
99   REM  TEST FOR SORT FOR NINE DIGIT ZIP CODES
100   FOR I = 1 TO N
103 SS%(I) = I
105 V$(I1 = "4": FOR J = ⍺ TO 8:V$(I) = V$(I) +  STR$ ( INT (10 *  RN
D (1))): NEXT J
106  PRINT V$(I)
110  NEXT : PRINT
M112  REM SORT STARTS HERE
113  REM ALSO SeE LINES 94-95
Q15 V$(N + 1) = "999999999":SS%(N + 1) = N + 1
116 V$(0) = "  ":SS%(0) = 0: REM  THESE VALUES INCLUDED BECAUSE LINE
100 STARTS WITH I=1
120 P = 1:Q = N:ST = 0
130  IF P >  = Q THEN 170
135 K = Q + 1: GOSUB 1145
140  IF J - P < Q - J THEN 150
145  GOSUB 400: GOTO 160
150  GOSUB 500
160 ST = ST + 2
162  PRINT "TOP= ";ST; TAB( 10);"P= ";P; TAB( 17)"Q= ";Q
165  GOTO 130
170  IF ST = 0 THEN 200
180 Q = Sc(ST):P = SK(ST - 1)
185  PRINT "TOP= ";ST; TAB( 10);"P= ";P; TAB( 17)"Q= ";Q
190 ST = ST - 2: GOTO 130
200  FOR I J 0 TO N: PRINT I; TAB( 5);SS%(I); TAB( 10);V$(sSS%(I)): NE
XT
201  END
400 SK(ST + 1) = P:SK(ST + 2) = J - 1:P = J + 1: RETURN
500 SK(ST + 1) = J + 1:SK(ST + 2) = Q:Q = J - 1: RETURN
1145 VI = SS%(P):VH$ = V$(nI):I = P:J = K
1160 J = J - 1: IF V$(SS%(J)) <  = VH$ THEN 1170
1165  GOTO 1160
1170 I = I + 1: IF V$(SS%(I)) >  = VH$ THEN 1180
1175  GOTO 1170
1180  IF J <  = I THEN 1200
1190 GA = SS%(I):GB = SK%(J)
1195 SS%(I) = GB:SS%(J) = GA: GOTO Y160
1200 SS%(P) = SS%(J):SS%(J) = VI: RETURN
```

In the July 1980 issue of MICRO (26:13), the article on sorting by Richard Vile interested me. I was looking for a faster sort for my mailing list programs. That article assumed that you can move the numbers or names that you are sorting. In my mailing list programs, I cannot do that. I work with files of 200 to 400 names and addresses on several mailing lists that are on disk. However, I took the quicksort listed on page 28, changed it from Integer BASIC to Applesoft BASIC, and modified it to sort on an index rather than sort on the numbers and/or names themselves.

While I was doing this conversion, I remembered that the post office was planning to change zip codes from 5 to 9 digits. Since my mailing programs sorted by zip before printing the labels, I used nine-digit zip codes for testing during the conversion process.

When I want to sort a group, a Sort Sequence Index (see line 103 of listing 1 for SS%) is used. This way I can move these sequence numbers instead of moving the actual file on the disk. In modifying the Apple Quicksort in Mr. Vile's article, I tried to keep the line numbers the same for easy cross reference. This helped a lot while I debugged the program.

The finished conversion product is given in listing 1. (Figure 1 is a list of variables and purpose, and figure 2 is for those who are familiar with the article noted above.) Lines 90-110 are used to generate 9-digit zip codes that start with 4. The important difference between this program and Mr. Vile's is the subroutine starting at 1145. Notice that the comparisons are based on the Sort Sequence Index (SS%) instead of the numbers themselves. Compare figure 3 copied from the original article.

As you can see in the sample run (run 1) 20 numbers were randomly created. The numbers themselves were not moved, but the Sort Sequence Index was. The smallest zip code the sort found had an index of 17, and the largest had an index of 4. I then ran this program three times with 200, 300, and 400 numbers. The largest number TOP became was 12. Line 94 relflects this discovery.

My next project was to apply this Quicksort to handle multifield sorts, i.e., sorting a mailing list by last name, then the first name. In this example the last name is called the primary sort and the first name would be the secondary sort. In listing 2, V$ is the primary sort and W$ is the secondary sort.

The differences between listing 1 and listing 2 are in three areas:

1. The generation of the numbers to be sorted (lines 94-115),
2. the printout at the end of the program (line 200),
3. the comparisons in subroutine 1145 (lines 1160-1172).

In lines 94 through 115, I created a one-digit number V$ as the primary sort field and a 9-digit zip code for the secondary sort field. Line 200 was changed to print out both V$ and W$. Lines 1160-1162 and 1170-1172 are tricky. Compare lines 1160-1165 in listing 1 to those in listing 2. To understand this, just remember that you must go back to line 1160 whenever J is high, and go to 1170 when J is low or equal.

If you get to line 1162 then V$ (SS%(J)) = VH$ and you test your secondary sort field. If you have more than 2 sort fields, then you repeat the logic in 1160-1161 over until you get to your last sort field. Then the last sort field is handled just like W$ is, in line 1162.

In one of my applications I have 4 sort fields. If the program finds two records with all 4 sort fields equal, then the program stops, because in that application no two records should be exactly the same.

Lines 1170-1172 have been modified just like lines 1160-1162. A sample run with 20 pairs of numbers is given as an example of this program (run 2).

I hope that this article has helped you to sort out your problems with sorts when you cannot move the entries themselves in the sorting process.

Bill Reese has a Master of Mathematics from Cleveland State University. He is a computer specialist for the U.S. Air Force at Wright Patterson Air Force Base. He owns an Apple II which he uses to support a newsletter mailing list for his church's singles club. He has also computerized his model railroad's waybills and switching lists.

Figure 1

Vile's Article	My Listing	Purpose
TOP	ST	Point to top of stack
STACK	SK	STACK of partitions to sort
A(I)	V$(I)	Field to be sorted
V	VH$	Hold field for comparisons
TEMP	GA,GB	Temporary holders

Figure 2

Variable	Purpose
I,P	Local variable, low number of partition
N	Number of items
J,Q	Local variable, high number of partition
SK	StacK of partitions to sort
SS%	Sort Sequence Index (Integer Variable)
ST	Point to top of stack
V$	Primary sort field
VH$	Hold field for comparison
VI	Hold index for comparison
W$	Secondary sort field

Figure 3

```
1145 VH$=V$(P):I=P:J=K
1160 J=J-1:IF V$(J)< =VH$
     THEN 1170
1165 GO TO 1160
1170 I=I+1:IF V$(I)> =VH$
     THEN 1180
1175 GO TO 1170
1180 IF J < =I THEN 1200
1185 TEMP=V$(I)
1186 V$(I)=V$(J)
1188 V$(J)=TEMP
1199 GO TO 1160
1200 V$(P)=V$(J)
1202 V$(J)=VH$
1999 RETURN
```

Run 1

```
NUMBER TO BE SORTED: 20
457188910
469927789
469026711
495696624
493153727
451635537
461576650
459036737
448501656
429879597
```

(Continued)

(Run 1 continued)

```
459573279
476600802
440954747
408470923
450983254
486018953
402300858
475895981
408563191
490375116

TOP= 2    P= 1    Q= 8
TOP= 4    P= 7    Q= 8
TOP= 6    P=P9    Q= 8
TOP= 6    P= 7    Q= 7
TOP= 4    P= 1    Q= 5
TOP= 4    P= 6    Q= 5
TOP= 4    P= 1    Q= 4
TOP= 4    P= 1    Q= 1
TOP= 4    P= 3    Q= 4
TOP= 4    P= 3    Q= 2
TOP= 4    P= 4    Q= 4
TOP= 2    P= 10   Q= 20
TOP= 2    P= 10   Q= 9
TOP= 2    P= 11   Q= 20
TOP= 2    P= 11   Q= 10
TOP= 2    P= 12   Q= 20
TOP= 2    P= 17   Q= 20
TOP= 4    P= 17   Q= 16
TOP= 4    P= 18   Q= 20
TOP= 4    P= 21   Q= 20
TOP= 4    P= 18   Q= 19
TOP= 4    P= 18   Q= 17
TOP= 4    P= 19   Q= 19
TOP= 2    P= 12   Q= 15
TOP= 2    P= 12   Q= 12
TOP= 2    P= 14   Q= 15
TOP= 2    P= 14   Q= 13
TOP= 2    P= 15   Q 15
0    0
1    17   402300858
2    14   408470923
3    19   408563191
4    10   429879597
5    13   44095747
6    9    448501656
7    15   450983254
8    6    451635537
9    1    457188910
10   8    459036737
11   11   459573279
12   7    461576650
13   3    469026711
14   2    469927789
15   18   475895981
16   12   476600802
17   16   486018953
18   20   490375116
19   5    493153727
20   4    495696624
```

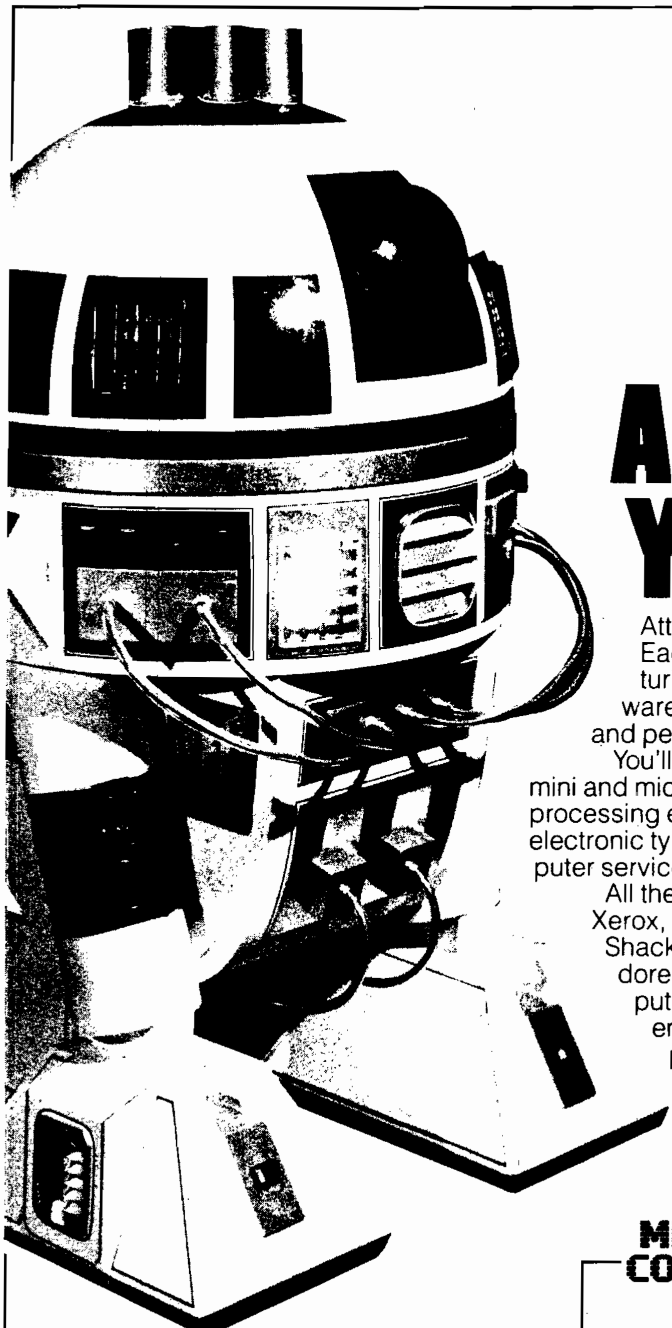
Listing 2

```
1   REM  *************
10  REM QU SORT IND 2 SORT FIELDS
20  REM  QUICK SORT P26:28 MICRO JULY 1980
30  REM PRINT LINES 162 & 185 MAY BE REMOVED
90  INPUT "NUMBER TO BE SORTED:  ";N
94  DIM SK(20)
95  DIM V$(N + 1),SS%(N + 1),W$(N + 1)
99  REM  TEST FOR SORT FOR NINE DIGIT ZIP CODES
100  FOR I = 1PTO N
103 SS%(Ii = I
104 V$(I) =  STR$ ( INT (10 *  RND (1)))
105 W$(I) = "4": FOR J = 1 TO 8:W$(I) = W$(I) +  STR$ ( INT (10 *  RN
D (10)): NEXT J
106P PRINT I; TAB( 5);SS%(I); TAB( 10);V$(SS%(I)); TAB( 20);W$(SS%(I)
)
110  NEXT : PRINT
112  REM SORT STARTS HERE
113  REM ALSO SEE LINES 94-95
115 V$(N + 1) = "9":SS%(N + 1) = N + 1:W$(N + 1) = "9":
116 V$(0) = "0":W$(0) = "0":SS%(0) = 0: REM THESE VALUES INCLUDED BEC
AUSE LINE 100 STARTS WITH I=1
120 P = 1:Q = N:ST = 0
130  IF P >  = a THEN 170
135 K = Q + 1: GOSUB 1145
140  IF J - P < Q - J THEN 150
145  GOSUB 400: GOTO 160
150  GOSUB 500
Q60 ST = ST + 2
162  PRINT "TOP= ";ST; TAB( 10);"P= ";P; TAB( 17)"Q= ";Q
165  GOTO 130
170  IF ST = 0 THEN 200
180 Q = SK(ST):P = SK(ST - 1)
185  PRINT "TOP= ";ST; TAB( 10);"P= ";P; TAB( 17)"Q= ";Q
190 ST@= ST - 2: GOTO 130
200  FOR I ] 0 TO N: PRINT I; TAB( 5);SS%(I); TAB( 10);V$(SS%(I)); TA
B( 20);W$(SS%(I)): NEXT
201  END
400 SK(ST + 1) = P:SK(ST + 2) = J - 1:P = J + 1: RETURN
500 SK(ST + 1) = J + 1:SK(ST + 2) = Q:Q = J - 1: RETURN
1145 VI = SS%(P):VH$ = V$(VI):I = P:J = K
1160 J = J - 1: IF V$(SK%(J)) < VH$ THEN 1170
1161  IF V$hSS%(J)) > VH$ GOTO 1160
1162  IF W$(SS%(J)) <  ] W$(VI) GOTO 1170
1165  GOTO 1160
1170 I = I + 1: IF V$(SS%(I)) > VH$ THEN 1180
1171  IF V$(SS%(I)) <PVH$ GOTO 1170
1172  IF W$(SS%(I)) >  = W$(VI) GOTO 1180
1175  GOTO 1170
1180  IF J <  = I THEN 1200
1190 GA = SS%(I):GB =PSS%(J)
1195 SS%(I) = GB:SS%(J) = GA: GOTO 116P
1200 SS%(P) = SS%(J):SS%(J) = VI: RETURN
```

Run 2

```
NUMBER TO BE SORTED: 20
1   1    4         404253628
2   2    1         402547722
3   3    5         434901450
4   4    8         479759823
5   5    8         486269585
6   6    7         414017862
7   7    2         419927548
8   8    4         444603652
9   9    8         409932506
10  10   1         443768300
11  11   9         499438847
12  12   8         482977184
13  13   9         435976469
14  14   7         483034670
15  15   8         407571009
16  16   4         476527172
17  17   8         455937055
18  18   6         421968942
19  19   8         449919376
20  20   0         491381959

TOP= 2   P= 1   Q= 4
TOP= 4   P= 5   Q= 4
TOP= 4   P= 1   Q=3
TOP= 4   P= 4   Q= 3
TOP= 4   P= 1   Q= 2
TOP= 4   P= 1   Q= 0
TOP= 4   P= 2   Q= 2
TOP= 2   P= 6   Q= 20
TOP= 2   `= 6   Q= 9
TOP= 4   P= 6   Q= 6
TOP= 4   P= 8   Q= 9
TOP= 4   P= 8   Q= 7
TOP= 4   P= 9   Q= 9
TOP= 2   P= 11  Q= 20
TOP= 2   P= 21  Q= 20
TOP= 2   P= 11  Q= 19
TOP= 2   P= 19  Q= 19
TOP= 2   P= 11  Q= 17
TOP= 2   P= 11  Q= 12
TOP= 4   P= 11  Q= 10
TOP= 4   P= 12  Q= 12
TOP= 2   P= 14  Q= 17
TOP= 2   P= 14  Q= 13
TOP= 2   P= 15  Q= 17
TOP= 2   P= 18  Q= 17
TOP= 2   P= 15  Q= 16
TOP= 2   P= 15  Q= 14
TOP= 2   P= 16  Q= 16
0   0    0         0
1   20   0         491381959
2   2    1         402547722
3   10   1         443768300
4   7    2         419927548
5   1    4         404253628
6   8    4         444603652
7   16   4         476527172
8   3    5         434901450
9   18   6         421968942
10  6    7         414017862
11  14   7         483034670
12  15   8         407571009
13  9    8         409932506
14  19   8         449919376
15  17   8    P    455937055
16  4    8         479759823
17  12   8    @    482977184
18  5    8         486269585
19  13   9         435976469
20  11   9         499438847
```

MICRO



On Buying a Printer

By Loren Wright

You've decided to buy a printer and are either impressed or overwhelmed with the number of choices available. To help you decide which printer best suits your needs, we'd like to familiarize you with printer features and manufacturers.

In researching this article, we tried to get information from every manufacturer of microcomputer-compatible printers selling for $2000 or less. The response was not 100%. Some manufacturers had moved, others had discontinued inexpensive models, others were out of business, and some simply failed to respond. Nevertheless, we compiled a substantial sample and will explain the many features printers offer. For more information see your local computer dealer or write the manufacturers. A list of addresses accompanies this article.

Probably the most important considerations are: "How much does it cost?" and, "Will it work with your computer?" However, there are many other features to consider. First, you should analyze your needs, both present and future. For instance, if you expect to be doing a lot of word processing, the quality of print would be an important feature. But if you expect to print large amounts of experimental data, then speed would be very important.

Characters

Most printers offer 96-character US ASCII character sets, which include both upper and lower case alphabets. Some of the less expensive printers print only upper case letters, however. This may be adequate for program listings and data printouts. Some printers allow substitution of the character ROM (Anadex, Base 2, Axiom IMP), and others allow at least one programmable character (Centronics 737 and 739, Base 2, C. Itoh Pro-Writer).

Print Quality

The best print quality is achieved with a formed character printer of the daisy wheel or ball (IBM) type. Most are priced well above our $2000 limit, but some of the less expensive ones are sold by C. Itoh, Vista, and NEC.

All others are dot matrix printers. The smallest matrix used is 5 × 7. The print head consists of a vertical row of seven printing needles which are controlled by seven solenoids. These solenoids lift and raise the needles at the appropriate moments as the head moves across the line. Because these characters usually appear grayish, rather than black, they are difficult to read — especially in photocopies or when reduced for publication. Lower case letters with descenders (the part of the character that normally extends below the line, as with g, j, p, q, and y) are crowded above the line. When extra needles are added (9 is a common total) these true descenders can be produced, and often an underline can be added. Centronics 737 and 739, Anadex DP-9610 RO, and Epson MX-80 are models with extra needles.

Another way of improving print quality is to stagger the needles in two rows. The Integral Data Systems Paper Tiger uses five needles interwoven with four. Other, considerably more expensive printers, use as many as 18, thereby largely eliminating the blank spaces between the needle imprints that cause the gray appearance mentioned above.

Yet another method is adopted by the Epson MX-80; in the *double print* mode, the characters are first printed normally, then the paper is advanced 1/216" and those characters are printed again. This fills in most of the space created between dots on the first pass. The MX-80 also has a *print enhancement* mode where the needles actually hit the ribbon harder. This mode is particularly useful for making multiple copies. Either of these special modes cuts the print speed in half and doubles the wear on the print head. Therefore these modes should be used judiciously.

Graphics Capability

Some printers allow individual control of every dot (Victor 5080, Base 2, Axiom, Centronics 739). This is useful in producing printouts of Apple Hi-Res screens. Even computers without high-resolution graphics can program these printers to produce high-resolution images. Base 2 offers an interface for Apple Hi-Res graphics. In this issue (39:44) a program is presented to dump the Apple Hi-Res screen to an Integral Data Systems Paper Tiger.

Line and Character Spacing

Some of the less expensive printers have a fixed number of characters per line, such as 21, 32, 40, 48, or 80. Be sure to get a line length that will suit your needs. Most other printers have line lengths variable from 40 to 132, selectable with either a program or switches.

Some printers (notably the Centronics 737 and 739) have a proportional spacing mode which produces copy like our typesetter prints this line. The narrower characters, such as 'I' and 'J,' take up less space than 'M' and 'W.' The overall effect is more pleasing than the 'monospace' copy produced by other printers.

With right-justification (also on the Centronics 737 and 739), the words line up at the right margin. Other printers produce what is called 'ragged right,' where alignment is achieved only at the left side of the page.

Variable line separation, subscripts, superscripts, and elongated characters are other extras to look for.

Paper Handling

Printer paper comes in a variety of forms and it is important to know which types your printer will take.

Fan fold is a continuous length of paper with holes on each edge. Usually the edges can be torn off and individual sheets separated. A wide variety of sizes and styles is available.

Roll is an inexpensive, long, continuous roll of paper. *Individual sheets* include stationery, letterhead, notebook paper, scrap paper, and special forms. Other papers available include *self-adhesive labels* and *multi-part forms.*

The most common method for advancing paper through a printer is with an adjustable tractor feed. Centronics and Epson models have a 'pin feed.' Both feed methods assure that paper can move quickly and precisely through the printer.

Self-adhesive labels and forms can be accommodated by tractor and pin feeds, but many of these feed mechanisms cannot handle the extra thickness and weight. Printer manufacturers usually specify the maximum thickness or number of plies that can be accommodated.

Individual sheets are handled by a friction feed mechanism (like a typewriter). These mechanisms will also handle roll paper, but a horizontal spindle of some sort for the roll is required.

Many printer models offer a combination of tractor and friction feeds.

Special Papers

Some of the less expensive printers require special paper. Thermal printers need special heat-sensitive paper. Instead of needles, the print head is composed of miniature heating elements which cause the paper to change color. Two cautions when using this paper are in order: 1) The blue-purple color commonly available does not photocopy well, and 2) the image tends to fade, particularly if transparent tape is applied over it.

The other kind of paper is electrosensitive. The standard needles are replaced with electrodes, which complete an electrical circuit when applied to the aluminum-coated paper. The normally shiny surface is turned black to form a character image. Handling this paper can be a very messy undertaking, as the metal coating rubs off easily.

Both of these special papers are considerably more expensive than the plain paper, and not as easily available.

The advantage of these kinds of papers is in the cost of the printer. No ribbon, or the associated feed mechanism, is required, nor are the seven or more individual solenoids to control the printing needles. Other economies such as fixed paper width, line length, and upper case only, are available to produce a truly bargain printer. At some point, however, the difference in the cost of the paper will add up to the difference in printer prices. This may take a few months, or many years, depending on how much you use your printer. Another advantage is that these printers tend to be quieter because they have fewer moving parts.

If you do decide to buy one of these non-impact printers, a useful feature is adjustable print darkness. A higher setting will make the copy more readable, while a lower one will extend the life of the print head. Also, as these print heads get older, the copy they produce gets lighter, so you will want to compensate for this aging.

Speed

The speed of a printer may be specified in characters per second (cps) or lines per minute. Formed character printers will typically do 25 to 50 cps, while dot-matrix printers are usually much faster. Typical values are 50 to 100 cps, while some print at 30 cps and others print faster than 200 cps. Sometimes there is a difference between the maximum or "burst" rate of printing and the average rate.

A number of printer features contributes to the overall speed. Bidirectional printing saves the time consumed by the extra carriage return required in unidirectional printing.

Logic seeking means the printer is able to look ahead and scout out the most efficient path for the print head. Both bidirectional printing and logic seeking require a buffer — an area of memory in the printer where it can inspect things before actually printing. Even without bidirectional printing or logic seeking, a buffer can add speed to the printing process. Until the buffer fills up, the printer will accept characters as fast as the computer sends them. Often, the computer is freed for other duty while the printer is still busy.

The use of special features, such as proportional spacing, right-justification, and print formatting may slow the printer down.

Several printers allow selection of the baud rate, either with switches or under program control.

Programmable Features

Some models allow extensive pro gramming of printer operations. W have already mentioned programmabl characters, elongated characters, bau rates, and line lengths. Other program mable features may include margins top-of-form, tabs, and print formattin (like print using).

Interfaces

Some printer models are sold a "designed for" a particular computer There are a number available for th Apple, several for the TRS-80, and a fev for the PET. Most, however, come witl either a standard parallel, or RS-232C serial interface, or both. Special inter faces for particular machines usuall cost extra. Most microcomputers however, will work with one of thes standard interfaces.

The most common parallel interfac is called "Centronics-compatible," which consists of seven data bits an three handshake bits. There are, how ever, 8-bit interfaces, and others whicl do not conform to the Centronics stan dard. Some additional circuitry or pro gramming may be required if there i not complete interface compatibility.

Other interfaces are 20 mA curren loop (or TTY) and IEEE-488. The 20 mA current loop is used with the AIM, SYM KIM, and other teletype-oriente machines. Adapting an RS-232C inter face to 20 mA current loop is fairly easy requiring only a few components IEEE-488 is generally used with th PET, but it is also used with Hewlett Packard and Tektronix controllers, an a wide variety of scientific tes equipment.

Two manufacturers (Base 2 and Vic tor Data Systems) include all four of th above interfaces as standard in thei printer models. Even the combination o a parallel and an RS-232C interface wil increase the flexibility of your printer making it easier to use with computer other than your own.

Other Features

With *self test*, the printer goe through a series of procedures testin some or all of the printer's functions This may be done on power-up or o demand.

An *out-of-paper signal* lets th printer detect when paper has run out stops printing, and usually sounds a audio alarm.

A Different Approach

The Axiom/Seikosha GP-80M does not use the standard needle/solenoid design for impact dot matrix printers. Instead, it uses a unihammer (single hammer) which rapidly strikes against splines on a freely rotating platen behind the paper. This model is one of the least expensive printers that do not require special thermal or electrosensitive paper. At 30 cps it is also one of the slowest.

Build Your Own

Heath and Coosol sell *printer kits*. The advantages of building a kit are: 1) you save money, 2) you know how well it was put together, 3) you get extensive documentation so you can usually fix it yourself if something goes wrong. The disadvantages are: 1) you may do a poor job of building it, 2) it takes time you may not have.

Generally, prices are going down while capabilities increase. Most of the major computer manufacturers offer one or more printers as parts of their "systems." Often you pay a premium price for relatively little power. You do know these printers will work with the specified computer, however, while it may take some effort to get a non-system printer working.

Whether you choose to buy the 'system' printer or opt for another, you certainly won't be saying, "I had no choice!"

Anadex, Inc.
9825 De Soto Avenue
Chatsworth, California 91311

Axiom Corporation
1014 Griswold Avenue
San Fernando, California 91340

Base 2
P.O. Box 3548
Fullerton, California 92634

Centronics Data Computer Corp.
Hudson, New Hampshire

Computer Devices, Inc.
25 North Avenue
Burlington, Massachusetts 01803
Mini Term 1201

Coosol, Inc.
P.O. Box 743
Anaheim, California 92805

Epson America, Inc.
23844 Hawthorne Boulevard
Torrance, California 90505

Heath Company
Benton Harbor, Michigan 49022

Integral Data Systems
Milford, New Hampshire 03055
Paper Tiger

C. Itoh Electronics, Inc.
5301 Beethoven Street
Los Angeles, California 90066

Microtek, Inc.
9514 Chesapeake Drive
San Diego, California 92123
Bytewriter-1

NEC Information Systems, Inc.
5 Militia Drive
Lexington, Massachusetts 02173

United Systems Corporation
918 Woodley Road
P.O. Box 458
Dayton, Ohio 45401
DigiTec 6430/6470 Non-impact

Victor Business Products
3900 North Rockwell Street
Chicago, Illinois 60618

Vista
1317 E. Edinger Avenue
Santa Ana, California 92705

MICRO

Using a TTY Printer with the AIM 65

While Rockwell provided both the hardware and software to permit TTY I/O on the AIM 65, output to a TTY while retaining AIM keyboard input is not allowed. The programs presented in this article provide for output to a teletype printer without restricting use of the AIM keyboard for input.

Larry P. Gonzalez
Dept. of Physiology and Biophysics
University of Illinois Medical Center
P.O. Box 6998
Chicago, Ilinois 60680

I recently obtained a TTY printer for use with my AIM 65 microcomputer. Since the AIM contains a hardware TTY interface, and TTY I/O routines are provided in the monitor, I expected little difficulty getting my TTY printer up and running. While the hardware interface posed no problem, a closer look at the monitor I/O routines revealed that TTY output is allowed only when the TTY/KB switch is in the TTY position. This is because the monitor routine OUTPUT ($E97A) tests the TTY/KB switch, instead of checking OUTFLG ($A413) before sending a character to the TTY, or to the on-board Display/Printer. Thus, entering "L" to indicate TTY output only works with this switch in the TTY position. Since I want to retain use of the AIM keyboard while sending output to my TTY printer, the TTY/KB switch must be in the KB position. This prevents my use of the OUTPUT routine (called by OUTALL at $E9BC).

Listing 1

```
                     ;*
                     ;* OUTPUT HANDLER FOR TTY PRINTER
                     ;*
                     ;*      BY LARRY P. GONZALEZ
                     ;*
                     ;NOTE: LOAD PROGRAM START ADDRESS INTO UOUT ($10A)
                     ;      BEFORE CALLING THIS ROUTINE. FOR THIS
                     ;      ASSEMBLY, SET $10A=$00 AND $10B=$02
                     ;
                     OUTTTY EQU $EEA8          ;OUTPUT CHARACTER TO TTY
                     ;
                            ORG $200

                     ;
0200 B00D                   BCS SECND          ;TEST FOR FIRST ENTRY
0202                 ;
0202 A925            FIRSTM LDA #$25           ;SET TRANSMISSION SPEED
0204 8D17A4                 STA $A417
0207 A900                   LDA #$00
0209 8D18A4                 STA $A418
020C 4C1C02                 JMP END
020F                 ;
020F                 ;ALL SUBSEQUENT ENTRIES MADE HERE
020F                 ;
020F                 ;NOTE: ACC PLACED ON STACK IN OUTALL
020F                 ;
020F 68              SECND  PLA
0210 C90D                   CMP #$0D
0212 D005                   BNE OUT
0214                 ;
0214 20A8EE          TCRLF  JSR OUTTTY         ;CR AND LF TO TTY
0217 A90A                   LDA #$0A
0219 20A8EE          OUT    JSR OUTTTY
021C 60              END    RTS
                            END
```

Listing 2

```
;*
;* OUTPUT HANDLER AND FORMATTER
;*        FOR TTY PRINTER
;*
;*     BY LARRY P. GONZALEZ
;*
;MONITOR ADDRESSES
;
COMIN  EQU $E182              ;MONITOR ENTRY
PSLS   EQU $E7DC              ;BACK UP CURSOR
RED2   EQU $E976              ;DISPLAY CHARACTER
OUTPUT EQU $E97A              ;OUTPUT ONE CHAR TO D/P
CRCK   EQU $EA24              ;CLEAR POINTERS AND OUTPUT PRINT BUFFER
OUTTTY EQU $EEA8              ;OUTPUT ONE CHAR TO TTY
CUREAD EQU $FE83              ;READ ONE CHAR FROM THE KEYBOARD
;
PRTADR EPZ $00                ;ADDRESS OF MSG TO PRINT
PRTPOS EQU $34F               ;PRINT HEAD POSITION
PGCNT  EQU $350               ;PAGE COUNT
LINCNT EQU $352               ;LINE COUNT
PAGING EQU $353               ;PAGE FLAG
PFLAG  EQU $354               ;PRINT FLAG
```

(Continued)

Listing 2

```
                  TITLE  EQU $355         ;TITLE
0392                     ORG $392
0392 57414E       WANT   ASC 'WANT PAGING (Y/N)?;'
0395 542050
0398 414749
039B 4E4720
039E 28592F
03A1 4E293F
03A4 3B
03A5 20454E       TITLES ASC ' ENTER PAGE TITLE:;'
03A8 544552
03AB 205041
03AE 474520
03B1 544954
03B4 4C453A
03B7 3B
03B8 504147       PAGES  ASC 'PAGE ;'
03BB 45203B
0200                     ORG $200
0200
0200 A90D         INIT   LDA #START       ;INITIALIZE UOUT
0202 8D0A01              STA $10A
0205 A902                LDA /START
0207 8D0B01              STA $10B
020A 4C82E1              JMP COMIN
020D B07A         START  BCS SECND
020F A925         FIRSTM LDA #$25         ;SET TRANSMISSION SPEED FOR TTY
0211 8D17A4              STA $A417
0214 A900                LDA #$00
0216 8D18A4              STA $A418
0219 2024EA              JSR CRCK
021C A203                LDX /WANT        ;WANT PAGING?
021E A092                LDY #WANT
0220 20FC02              JSR PRINT
0223 2083FE              JSR CUREAD
0226 8D5303              STA PAGING
0229 C959                CMP 'Y
022B D065                BNE TCRLF
022D 2024EA              JSR CRCK
0230 A901                LDA #$01         ;INITIALIZE PAGE COUNT AND LINE NUMBER
0232 8D5203              STA LINCNT
0235 8D5003              STA PGCNT
0238 A900                LDA #$00
023A 8D5103              STA PGCNT+1
023D A203                LDX /TITLES      ;GET PAGE TITLE
023F A0A5                LDY #TITLES
0241 20FC02              JSR PRINT
0244 2024EA              JSR CRCK
0247 A200                LDX #$00
0249 2083FE       TTLIN  JSR CUREAD
024C C97F                CMP #$7F         ;DELETE?
024E D00B                BNE CHARIN
0250 E000                CPX #$00
0252 F0F5                BEQ TTLIN
0254 CA                  DEX              ;BACK UP POINTER
0255 20DCE7              JSR PSL6         ;BACK UP DISPLAY
0258 4C4902              JMP TTLIN
025B C90D         CHARIN CMP #$0D
025D F00B                BEQ TTLEND
025F 2076E9              JSR RED2
0262 9D5503              STA TITLE,X
0265 E8                  INX
0266 E030                CPX #$30         ;IS BUFFER FULL (60 CHARS)?
0268 D0DF                BNE TTLIN
026A A93B         TTLEND LDA #$3B         ;STORE ';' TO END TITLE
026C 9D5503              STA TITLE,X
026F              ;TITLE OUTPUT ROUTINE
026F 209202       TTLOUT JSR TCRLF
0272 209202              JSR TCRLF
0275 20EF02              JSR LINE
0278 A203                LDX /TITLE
027A A055                LDY #TITLE
027C 200003              JSR TPRINT
027F 20B302              JSR PGNUM
0282 A902                LDA #$02
0284 8D5203              STA LINCNT
0287 D009                BNE TCRLF
```

(Continued)

A TTY Output Handler

The program presented in listing 1 is a short user output handler which replaces the AIM OUTPUT subroutine to allow TTY output while retaining input from the AIM keyboard. This program tests the carry bit to determine if this is the first entry to this routine. The first entry usually occurs with execution of the monitor WHEREO ($E871) subroutine, which clears the carry bit upon first entry to a user output handler. If the carry is clear (first entry), the baud rate ($A417) and delay ($A418) are initialized and an RTS (Return from Subroutine) is executed. I found that the parameters suggested by Rockwell (page 9-31 of the user's guide) did not work well with my printer; the values I used were determined by trial and error.

For subsequent entry, the carry bit should be set prior to jumping to this program, as is done by the monitor OUTALL routine. OUTALL places the character to be output onto the stack, so this character is pulled into the accumulator upon subroutine entry. If the character is a carriage return ($0D), it is sent to the TTY and is followed by a linefeed ($0A). Otherwise, the character is output to the TTY, using the monitor OUTTTY routine ($EEA8), and an RTS instruction is executed.

Output is directed to the TTY printer by loading the start address of this program (here $0200) into the vector to the user output handler (UOUT = $010A, $010B) and specifying "U" as the output device. This can be used with any of the AIM routines which permit a selection of the output device.

Providing Page Titles and Numbers

A fancier output handler is presented in listing 2. This program requires more memory, but is easier to use (it loads the start address into UOUT) and provides for optional page headings and page numbers.

To use this program, first run the program at $0200 to enter the routine start address into UOUT. Output can then be directed to the TTY from the AIM monitor, from the Text Editor, or from the Assembler (but not from the AIM disassembler) by specifying "U" as the output device. The message "WANT PAGING (Y/N)?" will be displayed, to which a response of "N" will result in unformatted (no paging) output to the TTY. A response of "Y" is followed by the message "ENTER PAGE

TITLE:" The user can then enter a title of up to 60 characters, terminated by a carriage return, which will be output as a header on each page of output, along with the page number.

The program listings presented in this article were prepared on my TTY printer using this program.

Directing Disassembled Output to the TTY

As noted above, the programs in listings 1 and 2 may be used by the AIM monitor, the Text Editor, or the Assembler. The AIM disassembler, however, sends output to the AIM printer without an optional output device. Since I often save disassembled listings as part of my program documentation, I also wanted the capability of directing the output of the disassembler to my TTY printer. Listing 3 presents a program which provides this ability.

This program is very similar to the AIM disassembler, but it has OUTFLG set to "U" to permit TTY output, and has calls to the monitor routine CRCK ($EA24) changed to CRLF ($E9F0). Using CRLF allows sending carriage return characters to the TTY printer while retaining AIM keyboard input. Run this program (* = $8D00) and respond to the prompts as for the AIM disassembler. Output is directed to the TTY printer.

With these programs my TTY printer is a useful addition to my AIM 65 system.

Larry P. Gonzales is an Assistant Professor of Physiology and Biophysics at the University of Illinois Medical Center. He has 12 years experience programming in high level languages and several years in the use of minicomputers for real-time data acquisition and signal analysis. During the last two years he has been developing a system using an AIM 65 in the collection and analysis of electrophysiological data.

MICRO

Listing 2

```
0289                ;SECOND & SUBSEQUENT ENTRY TO TTOUT
0289                ;ACC WAS PUSHED IN OUTALL
0289 68        SECND  PLA
028A C90D             CMP #$0D
028C F004             BEQ TCRLF
028E 20A8EE           JSR OUTTTY
0291 60        RETRN  RTS
0292 A90D      TCRLF  LDA #$0D          ;CR AND LF TO TTY PRINTER
0294 20A8EE           JSR OUTTTY
0297 A90A             LDA #$0A
0299 20A8EE           JSR OUTTTY
029C AD5303           LDA PAGING
029F C959             CMP 'Y
02A1 D00F             BNE RTRN
02A3 A900             LDA #$00
02A5 8D4F03           STA PRTPOS
02A8 EE5203           INC LINCNT
02AB AD5203           LDA LINCNT
02AE C93F             CMP #$3F
02B0 F0BD             BEQ TTLOUT
02B2 60        RTRN   RTS
02B3                ;OUTPUT PAGE NUMBER TO TTY PRINTER
02B3 AD4F03    PGNUM  LDA PRTPOS
02B6 C93F             CMP #$3F
02B8 F00A             BEQ PGOUT
02BA A920             LDA #$20
02BC 20A8EE           JSR OUTTTY
02BF EE4F03           INC PRTPOS
02C2 10EF             BPL PGNUM
02C4 A203      PGOUT  LDX /PAGES
02C6 A0B8             LDY #PAGES
02C8 200003           JSR TPRINT
02CB AD5103           LDA PGCNT+1
02CE 203703           JSR THXOUT
02D1 AD5003           LDA PGCNT
02D4 203703           JSR THXOUT
02D7 209202           JSR TCRLF
02DA F8               SED               ;UPDATE 2-BYTE DECIMAL PAGE COUNTER
02DB 18               CLC
02DC A901             LDA #$01
02DE 6D5003           ADC PGCNT
02E1 8D5003           STA PGCNT
02E4 D008             BNE CLEAR
02E6 A900             LDA #$00
02E8 6D5103           ADC PGCNT+1
02EB 8D5103           STA PGCNT+1
02EE D8        CLEAR  CLD
02EF                ;OUTPUT LINE TO TTY PRINTER
02EF A249      LINE   LDX #$49
02F1 A92D             LDA #$2D
02F3 20A8EE    PRNT   JSR OUTTTY
02F6 CA               DEX
02F7 D0FA             BNE PRNT
02F9 4C9202           JMP TCRLF
02FC                ;PRINT MESG OR TITLE TO DISP/PR OR TO TTY PRINTER
02FC A9FF      PRINT  LDA #$FF
02FE D002             BNE TPRIN2
0300 A900      TPRINT LDA #$00
0302 8D5403    TPRIN2 STA PFLAG         ;SAVE ZERO PAGE DATA
0305 A500             LDA PRTADR
0307 48               PHA
0308 A501             LDA PRTADR+1
030A 48               PHA
030B 8601             STX PRTADR+1
030D 8400             STY PRTADR
030F AE5403           LDX PFLAG
0312 A000             LDY #$00
0314 B100      PRINT3 LDA (PRTADR),Y
0316 C93B             CMP ';            ;DONE?
0318 D009             BNE CHROUT
031A 68               PLA               ;RESTORE ZERO PAGE DATA
031B 8501             STA PRTADR+1
031D 68               PLA
031E 8500             STA PRTADR
0320 4C9102           JMP RETRN
0323 E000      CHROUT CPX #$00
0325 F006             BEQ TTY
0327 207AE9           JSR OUTPUT
032A 4C3003           JMP INCR
032D 20A8EE    TTY    JSR OUTTTY
0330 C8        INCR   INY
0331 EE4F03           INC PRTPOS
0334 4C1403           JMP PRINT3
```

(Continued)

Listing 2

```
0337                ;OUTPUT 2 HEX CHARACTERS TO TTY PRINTER
0337 48             THXOUT PHA
0338 4A                    LSR
0339 4A                    LSR
033A 4A                    LSR
033B 4A                    LSR
033C 204203                JSR CNVRT
033F 68                    PLA
0340 290F                  AND #$0F
0342 18             CNVRT  CLC
0343 6930                  ADC #$30
0345 C93A                  CMP #$3A
0347 9002                  BCC CHRPRT
0349 6906                  ADC #$06
034B 20A8EE         CHRPRT JSR OUTTTY
034E 60             END    RTS
                           END
```

Listing 3

```
                 ;* DISASSEMBLING TO TTY
                 ;* BY LARRY P. GONZALEZ
                 ;REPLACES CRCK IN AIM'S DISASSEMBLER WITH CRLF
                 COMIN  EQU $E182       ;MONITOR ENTRY
                 CGPCO  EQU $E5D7       ;ALTER PROGRAM COUNTER
                 GCNT   EQU $E785       ;GET NUMBER OF LINES
                 DONE   EQU $E790       ;CHECK COUNT
                 PSL1   EQU $E837       ;PRINT '/'
                 RCHEK  EQU $E907       ;CHECK FOR STOP COMMAND
                 OUTPUT EQU $E97A       ;OUTPUT TO TTY OR TO D/P
                 CRLF   EQU $E9F0       ;OUTPUT CR AND LF
                 ADDIN  EQU $EAAE       ;GET FOUR BYTE ADDRESS
                 OUTTTY EQU $EEA8       ;OUTPUT ONE CHARACTER TO TTY
                 DISASM EQU $F46C       ;DISASSEMBLE ONE INSTRUCTION
                        ORG $8D00
8D00 A95C               LDA #UOUT       ;INITIALIZE JUMP TO USER OUTPUT HANDLER
8D02 8D0A01             STA $10A
8D05 A98D               LDA /UOUT
8D07 8D0B01             STA $10B
8D0A A925               LDA #$25        ;SET TRANSMISSION SPEED
8D0C 8D17A4             STA $A417
8D0F A900               LDA #$00
8D11 8D18A4             STA $A418
8D14 AD13A4             LDA $A413       ;SAVE OUTFLG
8D17 48                 PHA
8D18 A955               LDA 'U          ;SET OUTFLG="U"
8D1A 8D13A4             STA $A413
8D1D A92A        KDISA  LDA #$2A        ;GET START ADDRESS
8D1F 207AE9             JSR OUTPUT
8D22 20AEEA             JSR ADDIN
8D25 B0F6               BCS KDISA
8D27 20D7E5             JSR CGPCO
8D2A 2037E8             JSR PSL1
8D2D 2085E7             JSR GCNT        ;GET COUNT OF INSTRUCTIONS
8D30 20F0E9             JSR CRLF
8D33 4C3E8D             JMP JDB
8D36 2007E9      JDA    JSR RCHEK
8D39 2090E7             JSR DONE        ;ARE WE DONE?
8D3C F017               BEQ JDD
8D3E 206CF4      JDB    JSR DISASM      ;GO TO DISASSEMBLER
8D41 AD25A4             LDA $A425       ;UPDATE PROGRAM COUNTER
8D44 38                 SEC
8D45 65EA               ADC $EA
8D47 8D25A4             STA $A425
8D4A 9003               BCC JDC
8D4C EE26A4             INC $A426
8D4F 20F0E9      JDC    JSR CRLF
8D52 4C368D             JMP JDA
8D55 68          JDD    PLA             ;RESTORE OUTFLG
8D56 8D13A4             STA $A413
8D59 4C82E1             JMP COMIN       ;RETURN TO MONITOR
8D5C             ;TTY OUTPUT HANDLER
8D5C 68          UOUT   PLA
8D5D C90D               CMP #$0D
8D5F D005               BNE OUT
8D61 20A8EE      TCRLF  JSR OUTTTY
8D64 A90A               LDA #$0A
8D66 20A8EE      OUT    JSR OUTTTY
8D69 60          END    RTS
```

A $200 Printer for C1P & Superboard

Hardware modifications are presented to interface the C1P to a Radio Shack Quick Printer II. Software considerations are discussed and demonstration programs are included.

Louis A. Beer
P.O. Box 705
Portola, California 96122

If you write programs, a near must for your computer is a printer. The Radio Shack Quick Printer II is relatively fast (32-character 120 lines per minute), reliable, quiet, and inexpensive (approximately $200). It is easy to interface to the Ohio Scientific C1P or Superboard. This article explains how.

There are three problems to handle, and all are quite easily overcome:

1. The Quick Printer operates at 600 baud. The C1P normally operates at 300 baud.

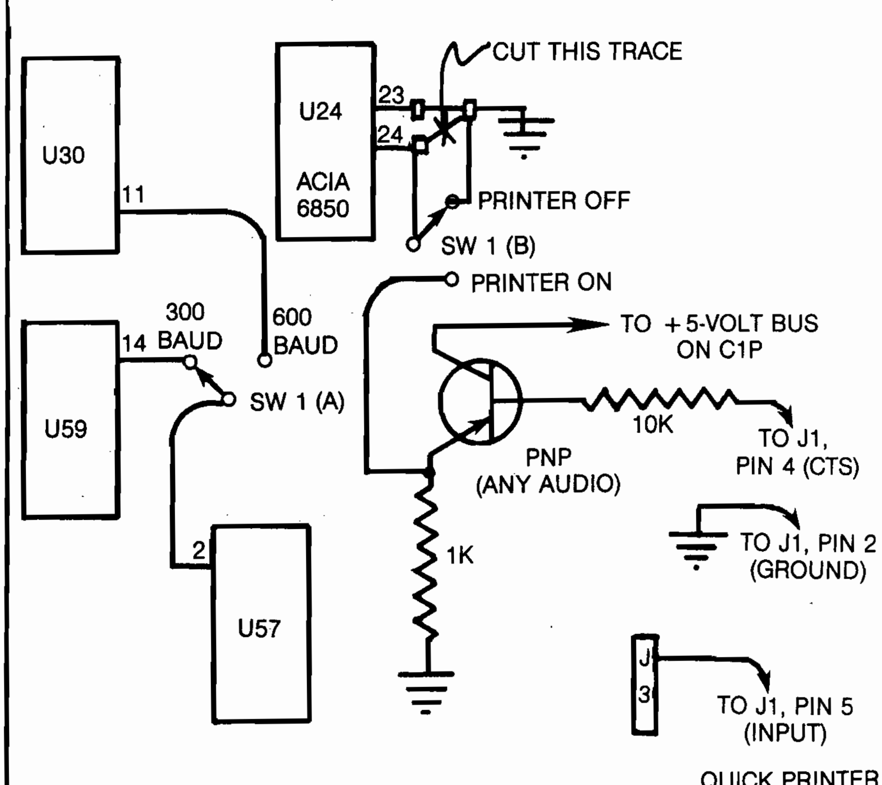
2. The Quick Printer sends a +5 volt signal on the CTS (clear-to-send) line to indicate it is ready to receive data, and −6.2 volts to indicate not ready. The C1P serial interface (ACIA, U-24 on the OSI schematic) takes +5 volts on its CTS line to inhibit sending data, and ground potential on this line to enable sending data.

3. The C1P character output program in ROM outputs ten nulls at the beginning of each print line. The Quick Printer does not recognize nulls ($00), and therefore locks up and sends a 'not clear-to-send' signal when these are encountered. Some previous fixes for solving this problem have merely eliminated the nulls, but this makes reliable saving on tape impossible once the fix is in memory. The loss of the ten nulls at the beginning of each line causes reading errors when reading the tape back. My system eliminates this problem by substituting the ASCII 'SOH' (start of heading) for the nulls. The Quick Printer recognizes this character, discards it, and waits for a printable character. The C1P treats it as a null.

Let's take these problems in order and give the solutions. First, to make the C1P switchable for 300 or 600 baud, locate pin 2 of U57 and cut the trace (which goes to pin 14 of U59) so that it can be switched to either pin 14 of U59 for normal 300 baud operation, or pin 11 of U30 for 600 baud operation. One half of a double-pole double-throw switch is used. (See wiring diagram.)

Second, to make the CTS (clear-to-send) switchable between normal C1P operation and Quick Printer operation, again refer to wiring diagram. Cut the trace at W3 (on the C1P) from pin 24 of U24 (ACIA) to ground. Use the other half of the double-pole double-throw switch to switch the CTS line (pin 24) between ground (normal, 300 baud, printer-off) and emitter of an audio transistor, which will effectively provide ground for 'clear-to-send' and +5 volts for 'not clear-to-send' signals being received from the printer.

***Figure 1:* C1P Modifications to Operate Quick Printer II**



I soldered the transistor collector directly to the +5-volt bus on the C1P, and the emitter through the 1K ohm ¼-watt resistor to the ground bus so that it is mechanically self-supporting. Any 3-wire connector can be used to connect the cable from the Quick Printer. I used a couple of RCA jacks. The RS-232 (out) port on the C1P must be populated per the diagram in the user's manual if you have not already done so. This takes four resistors and one PNP transistor and is rather easy to do. The schematic is in the user's manual and labelled "sheet 6 of 13." Only R72, R63, R64, R65 and Q1 are required. Any PNP audio transistor will do for Q1.

Third, the 8-line program given here will take care of the null problem. The BASIC support for outputting characters is in ROM $FF69 to $FF95 (65385 to 65429 dec). What we will do is lift this entire routine and put it in unused RAM, then replace the null at $FF80 with the SOH ($10). We do this by reading these 44 bytes and POKEing them into unused RAM starting at $0222 (546 dec). This is all done by lines 2 and 8. Lines 3, 4, 6 and 7 set the output vector and warm start pointers so that any output will use the routine starting at $0222 rather than the one in ROM $FF69. To set up your machine, LOAD this program, then RUN. It takes about a second to run. Next, hit BREAK and W (warm start) and you are in business.

You should next clear this BASIC program by typing NEW and hitting RETURN, or (in case you have another BASIC program already in memory and don't want to lose it) by typing 1 through 8 with RETURN to eliminate each line. The reason for clearing the program is that the DATA statements can confuse another program using DATA statements. Warm start will continue to work, but after any cold start the program will have to be loaded and run again to use the printer.

Here is the general operating procedure: when you want to list a program in the computer on the printer, start with the switch you installed in the normal (300 baud, no print) position. Type SAVE, hit RETURN, type LIST (and line numbers to be listed, if desired). Now turn on the printer mainline switch and put its PRINT switch to the on-line (up) position. The printer INPUT SELECT switch should *always* be in SERIAL (down) position. The printer will now print "PRINTER READY." Now put the double-pole switch you installed in the 600 baud/print position. Hit RETURN, and out comes your program listing. You can have the printer "on-line" when running a program which has printed output (a disassembler, for example) but watch out for excessive use of paper by PRINT statements used for screen clearing, etc.

```
1 REM:QUICK PRINTER FIX BY
LOU BEER
2 M=546:FORN=65385TO
65429:P=PEEK(N):POKEM,P
:M=M+1:NEXTN
3 DATA169,34,141,26,2,169,2,
141,27,2,76,116,162
4 DATA76,216,0
6 FORN=216TO228:READP:
POKEN,P:NEXTN
7 FORN=0TO2:READP:POKEN,
P:NEXTN
8 POKE569,16:END
OK
```

The whole modification is simpler than it sounds. If you have any problems in getting it to work, I will be glad to assist if you send a S.A.S.E.

MICRO

C1P to Epson MX-80 Printer Interface

A circuit is presented to interface the C1P to the popular Epson MX-80 printer.

Gary E. Wolf
227 Grove Street
Clifton, New Jersey 07013

There have been several articles written on interfacing the C1P with a printer, but it seems that each printer needs its own instructions. The Epson MX-80 is no exception.

Other sources have detailed the installation of the RS-232C, and figure 1 shows the schematic. By cutting the W10 trace, a negative 9 VDC can be applied at this point, via J3 pin 7. I used a simple transistor radio battery eliminator for this. Important: remember correct polarity. Positive on this source is *ground*.

Next, cut the trace that connects the ACIA (U14) pin 24 to ground. (See figure 2.) Solder a jumper from pin 24 to the CTS trace. Then mount a SP2T (single pole double throw) switch somewhere on the computer enclosure to put ground back on pin 24 when you use a cassette. The cassette won't operate properly if this pin is floating.

I mounted a DB-25 connector in the rear of my cabinet. Since only three pins will be used, almost any connector will do. Solder the cross connections between the DB-25 and a Molex connector, which fits into J3 on your computer board. (See figure 4.) Now to the printer.

I assume you have bought the MX series option for your printer, since it will not interface to a C1P without one. If the board has been installed, you may

Figure 1

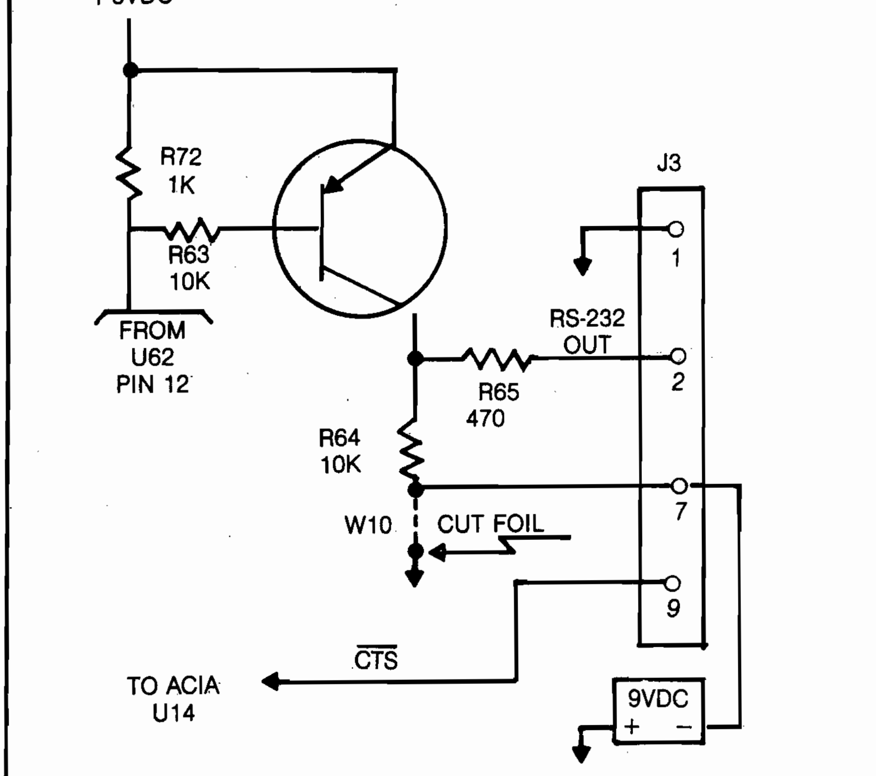
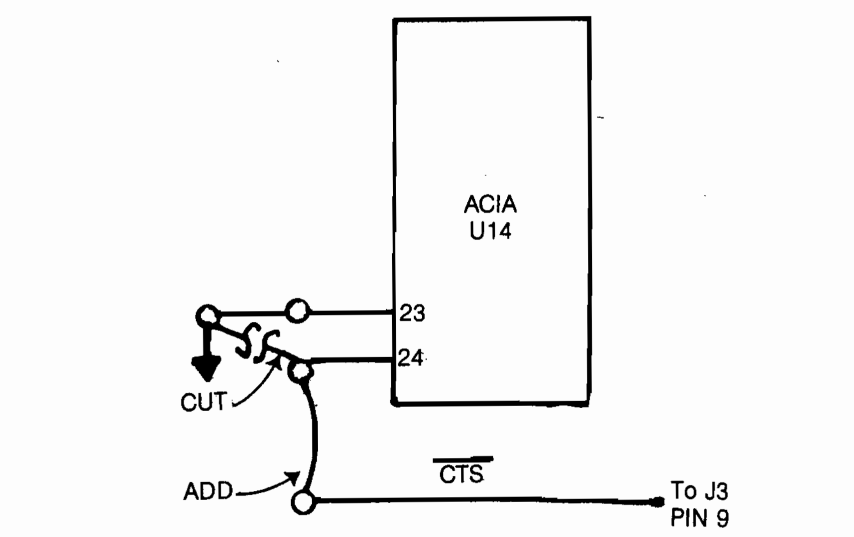


Figure 2



be ready to plug in the cable and be off and running, but don't count on it! Go to the series option manual and follow the instructions for removal of the printer cover. Check the settings on DIP switch 8141. See table 2 on page 4 of your manual. Settings should agree with table 1 (shown here).

***Table 1:* Setting of DIP SW (8141)**

Pin	Setting for 300 B.P.S.
1	Off
2	Off
3	On
4	On
5	N/A
6	Off
7	Off
8	N/A

The board comes from the factory with jumper JNOR connected. It should be cut and jumper JREV should be installed. This adds another inverter to the output at pin 11.

Pin 11 ultimately connects to the CTS lead at your computer. This is the handshake. A high signal on CTS inhibits ACIA output. With JREV on and JNOR off the C1P will send out data only when the printer is ready for it. Note also that ground from the computer is connected to pin 7 of the printer, *not* pin 1. They are not the same.

I have included a simple address and label program to get you started. The Epson MX-80 is a great printer, and although there are a few spots in the manuals that are confusing, most of the information is clear and helpful. With these tips you should have no problem with the interface.

MICRO

Figure 3

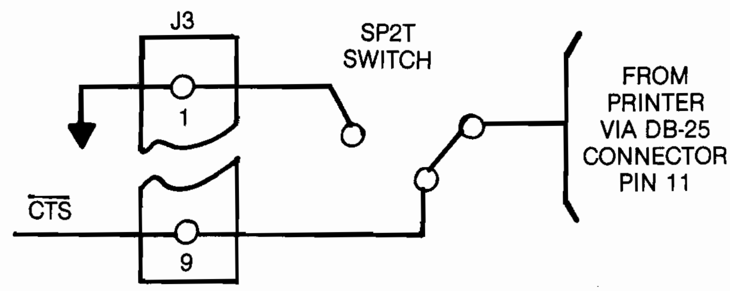
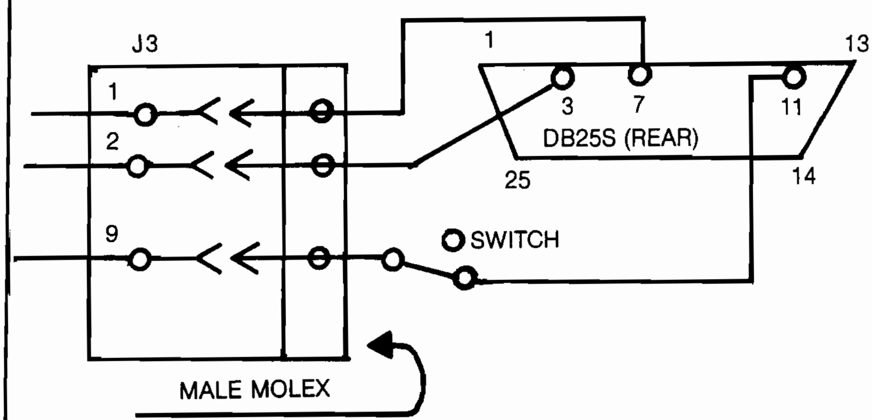


Figure 4



Utilities for the Paper Tiger 460

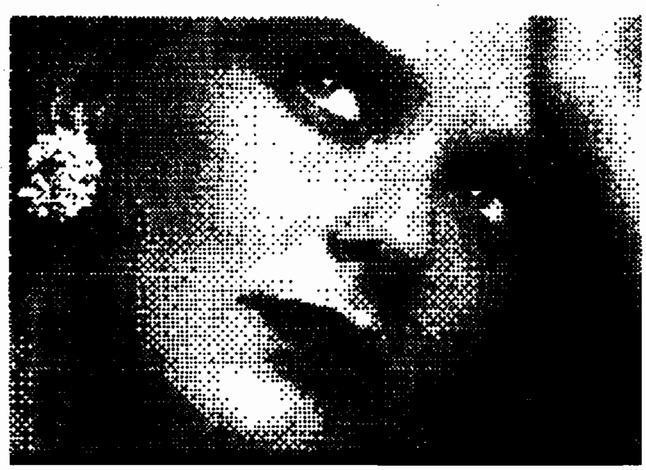
Here are two utilities for the Paper Tiger 460 printer for use with the Apple II. The Applesoft BASIC program lets you set all the programmable features of the Paper Tiger by choosing from a menu. The machine language program dumps the Apple Hi-Res graphics screen buffer to the printer.

Terry L. Anderson
Dept. of Physics & Computer Science
Walla Walla College
College Place, Washington 99324

The Paper Tiger 460 is an exciting addition to the group of printers available to the personal computer user. This dot matrix printer uses paper up to 10.5 inches wide and prints at a mode-dependent speed of up to 150 characters per second. It has a graphic option with 84 dots to the inch resolution (both vertical and horizontal). This is nearly double the resolution of most other printers with dot graphic modes. But the most unique feature is the use of overlapping dots. Most printers use a single row of print head wires allowing dots that nearly touch but cannot overlap. The 460 uses two side-by-side rows of four and five wires, respectively, which are staggered so that the resulting dots overlap about 30%. Thus a vertical line such as is used to print an 'L' or an 'I' is solid without distinct dots and has very little raggedness. The result is type quality nearly as good as fully-formed character printers such as Diablo and IBM Selectric, and adequate for many word processing applications.

The overlapping dots also allow solid black areas in graphics. With non-overlapping dot graphic printers, four dots in a square pattern leave a little white in the center of the pattern. This results in a slightly gray effect. But the overlapping dots of the 460 filling in the center of a four-dot square pattern completely, result in very solid blacks. This is important if you want to use the printer to construct bar code patterns for use with readers such as the new HEDS-3000 bar code wand from Hewlett-Packard. Areas of white in the middle of a bar can result in false readings. The Paper Tiger 460 should be very useful as a bar code printer, which is one of my next projects.

The high resolution of the 460's printing allows more options than most dot matrix printers have. These options include six character sizes, variable vertical line spacing, fractional line spacing up and down for sub- and super-scripts, and fully right- and left-justified text using variable character spacing, not just extra spaces between words. The firmware allows all of these features to be used under program control. This results in a great deal of flexibility, such as mixing type sizes on one line, and sub- and super-scripts. But choosing a feature requires sending special control characters, even if one feature is to be used for an entire print job. Some of these are hard to remember and some are difficult to send from the Apple keyboards, which cannot generate all 128 ASCII characters. Several important functions on the 460 require characters not available on Apple's keyboard. The one that enables auto-justification (control-D) conflicts with Apple's DOS use of that character, so some can only be sent using a program.



Tiger Setup

To make configuring the Paper Tiger 460 easy we need a configuring program. The first program, TIGER SETUP, allows you to choose the features you want from a menu. This reminds you which features are available and you don't have to remember all the special characters. When you exit with 'Q' (for quit) all the special characters to program the printer are sent.

The menu shows the options with the currently-selected value indicated by inverse video. Many selections are made with a single keystroke to toggle the state of the printer, such as between auto-justify mode and normal, or between six and eight lines per inch. The key is indicated by inverse video. Some selections require a single keystroke followed by a value for a parameter. The single keystroke will place the cursor just in front of the old value and allow a new value to be typed over the old. Choosing length of a form is an example. A few selections require two keystrokes; one to choose the category and the second a subcategory, such as horizontal or vertical for tabs, and right or left for margins. After the first keystroke the cursor is moved in front of the secondary choices to indicate the required action. After the second keystroke the value is entered. If, at any time, an invalid keystroke is entered the program simply returns to the main menu cursor location. In the case of tabs, up to eight tab locations can be entered, separated by commas.

Listing 1

```
10   REM ************************
12   REM *                      *
14   REM *     TIGER SETUP      *
16   REM *                      *
18   REM *         BY           *
20   REM *                      *
22   REM *   TERRY L ANDERSON   *
24   REM *                      *
26   REM *  WALLA WALLA COLLEGE *
28   REM *                      *
30   REM * BEGUN    1981 FEB 03 *
32   REM * LAST MOD 1981 FEB 19 *
34   REM *                      *
36   REM * MENU DRIVEN PROG TO  *
38   REM * CONFIGURE PAPER      *
40   REM * TIGER 460 FEATURES   *
42   REM *                      *
44   REM * CHANGE LINE 110 TO   *
46   REM * PRINTER SLOT#        *
48   REM *                      *
50   REM ************************
99   REM
100   REM
   INITIALIZE
110  SL = 1: REM PRINTER SLOT#
120  T$ = "TIGER SETUP"
130  V$ = "VER 81-FEB-19"
140  D$ =  CHR$ (4): REM CTRL-D
150  E$ =  CHR$ (27): REM <ESC>
160  N$ = "," +  CHR$ (0): REM END
     ING FOR ESC FUNC'S
170  FL% = 528
180  PS% = 48
190  AL% = 8:AG% = 4:AO% =  - 4
200  VT$ = "0,0,0,0,0,0,0,0":HT$ =
     VT$
210  REM
MENU

220  HOME : VTAB 1: HTAB 20 -  LEN
     (T$) / 2: PRINT T$
230  VTAB 3: HTAB 20 -  LEN (V$) /
     2: PRINT V$
240  VTAB 5
250  INVERSE : PRINT "G";: NORMAL
     : PRINT "RAPHIC MODE ";
260  IF G% THEN  PRINT "OFF/";: INVERSE
  : PRINT "ON": NORMAL : GOTO 280
270  INVERSE : PRINT "OFF";: NORMAL
     : PRINT "/ON"
280  INVERSE : PRINT "J";: NORMAL
     : PRINT "USTIFY MODE ";
290  IF J% THEN  PRINT "OFF/";: INVERSE
  : PRINT "ON": NORMAL : GOTO 310
300  INVERSE : PRINT "OFF";: NORMAL
     : PRINT "/ON"
310  INVERSE : PRINT "P";: NORMAL
     : PRINT "ROPORTIONAL SPACING
     ";
320  IF P% THEN  PRINT "OFF/";: INVERSE
  : PRINT "ON": NORMAL : GOTO 340
330  INVERSE : PRINT "OFF";: NORMAL
     : PRINT "/ON"
340  INVERSE : PRINT "L";: NORMAL
     : PRINT "INE SPACING ";
350  IF AL% = 6 THEN  PRINT "6/";
     : INVERSE : PRINT "8";: NORMAL
     : PRINT " LPI": GOTO 380
360  IF AL% = 8 THEN  INVERSE : PRINT
     "6";: NORMAL : PRINT "/8 LPI
     ": GOTO 380
370  PRINT "6/8 LPI ";: INVERSE :
     PRINT "SPECIAL SEE ADV LF":
     NORMAL
380  INVERSE : PRINT "A";: NORMAL
     : PRINT "DVANCE  ";: INVERSE
     : PRINT "L";: NORMAL : PRINT

     "F ";: INVERSE : PRINT AL%;:
     NORMAL : PRINT "           /
    48TH INCH"
390  HTAB 10: INVERSE : PRINT "G"
     ;: NORMAL : PRINT "RAPHIC LF
     ";: INVERSE : PRINT AG%: NORMAL

400  HTAB 10: INVERSE : PRINT "O"
     ;: NORMAL : PRINT "THER ";: INVERS
E : PRINT AO%: NORMAL
410  INVERSE : PRINT "C";: NORMAL
     : PRINT "HAR SPACING ";
420  IF C% = 0 THEN C% = 4
430  IF C% = 1 THEN  INVERSE : PRINT
     "5";: NORMAL : PRINT ",6,8.4
     ,10,12,16.8";: GOTO 490
440  IF C% = 2 THEN  PRINT "5,";:
     INVERSE : PRINT "6";: NORMAL
     : PRINT ",8.4,10,12,16.8";: GOTO
     490
450  IF C% = 3 THEN  PRINT "5,6,"
     ;: INVERSE : PRINT "8.4";: NORMAL
     : PRINT ",10,12,16.8";: GOTO
     490
460  IF C% = 4 THEN  PRINT "5,6,8
     .4,";: INVERSE : PRINT "10";
     : NORMAL : PRINT ",12,16.8";
     : GOTO 490
470  IF C% = 5 THEN  PRINT "5,6,8
     .4,10,";: INVERSE : PRINT "1
     2";: NORMAL : PRINT ",16.8";
     : GOTO 490
480  IF C% = 6 THEN  PRINT "5,6,8
     .4,10,12,";: INVERSE : PRINT
     "16.8";: NORMAL : GOTO 490
490  PRINT " CPI"
500  INVERSE : PRINT "I";: NORMAL
     : PRINT "NTERCHAR SPACING ";
     : INVERSE : PRINT IS%;: NORMAL
     : PRINT " /24TH CHAR WIDTH"
510  INVERSE : PRINT "M";: NORMAL
     : PRINT "ARGIN  ";: INVERSE
     : PRINT "L";: NORMAL : PRINT
     "EFT  ";: INVERSE : PRINT ML
     %;: NORMAL : PRINT "  /120TH
      INCH"
520  HTAB 8: INVERSE : PRINT "R";
     : NORMAL : PRINT "IGHT ";: INVERSE
 : PRINT MR%: NORMAL
530  INVERSE : PRINT "F";: NORMAL
     : PRINT "ORMS LENGTH ";: INVERSE
     : PRINT FL%;: NORMAL : PRINT
     " /48TH INCH  (=";FL% / 48;"
      INCH)"
540  INVERSE : PRINT "S";: NORMAL
     : PRINT "IZE PAGE SKIP ";: INVERSE
 : PRINT PS%;: NORMAL : PRINT " /4
     8TH INCH  (=";PS% / 48;" INC
     H)"
550  INVERSE : PRINT "T";: NORMAL
     : PRINT "ABS  ";: INVERSE : PRINT
     "H";: NORMAL : PRINT "ORIZ "
     ;: INVERSE : PRINT HT$;: NORMAL
     : PRINT " /120TH INCH"
560  HTAB 7: INVERSE : PRINT "V";
     : NORMAL : PRINT "ERT  ";: INVERSE
 : PRINT VT$;: NORMAL : PRINT " /4
     8TH INCH"
570  PRINT : INVERSE : PRINT "Q";
     : NORMAL : PRINT "UIT AND CO
     NFIGURE PRINTER"
580  PRINT : PRINT "SET: ";
590  POKE 49168,0: REM CLR KB STR
     OBE
600  GET A$
610  REM
   NOTE REQUESTED CHANGE
620  IF A$ = "G" THEN G% =  NOT G
     %: GOTO 960
630  IF A$ = "J" THEN J% =  NOT J
     %: GOTO 960
640  IF A$ = "P" THEN P% =  NOT P
     %: GOTO 960
650  IF A$ <  > "L" THEN 680
660  IF AL% = 8 THEN AL% = 6: GOTO
     960
670 AL% = 8: GOTO 960
680  IF A$ <  > "A" THEN 740
690  VTAB 09: HTAB 8: GET A$
700  IF A$ = "L" THEN  HTAB 12: INPUT
     AL%: GOTO 960
710  IF A$ = "G" THEN  VTAB 10: HTAB
     20: INPUT AG%: GOTO 960
720  IF A$ = "O" THEN  VTAB 11: HTAB
     15: INPUT AO%: GOTO 960
730  GOTO 960
740  IF A$ <  > "C" THEN 760
750 C% = C% + 1: IF C% > 6 THEN C
     % = 1: GOTO 960
760  IF A$ <  > "I" THEN 780
770  VTAB 13: HTAB 18: INPUT IS%:
      GOTO 960
780  IF A$ <  > "M" THEN 830
790  VTAB 14: HTAB 7: GET A$
800  IF A$ = "L" THEN  VTAB 14: HTAB
     14: INPUT ML%: GOTO 960
810  IF A$ = "R" THEN  VTAB 15: HTAB
     14: INPUT MR%: GOTO 960
820  GOTO 960
830  IF A$ <  > "T" THEN 930
840  VTAB 18: HTAB 5: GET A$
850  IF A$ <  > "H" THEN 890
860  VTAB 18: HTAB 12:HT$ = ""
870  GET A$: IF A$ <  >  CHR$ (13
     ) AND A$ <  >  CHR$ (141) THEN
     HT$ = HT$ + A$: PRINT A$;: GOTO
     870
880  GOTO 960
890  IF A$ <  > "V" THEN 920
900  VTAB 19: HTAB 12:VT$ = ""
910  GET A$: IF A$ <  >  CHR$ (13
     ) AND A$ <  >  CHR$ (141) THEN
     VT$ = VT$ + A$: PRINT A$;: GOTO
     910
920  GOTO 960
930  IF A$ = "F" THEN  VTAB 16: HTAB
     13: INPUT FL%: GOTO 960
```

(Continued)

Listing 1 *(Continued)*

```
940   IF A$ = "S" THEN  VTAB 17: HTAB
      15: INPUT PS%: GOTO 960
950   IF A$ = "Q" THEN 970
960   GOTO 210
970   REM
CONFIGURE TIGER

980  CS$ = ""
990   IF P% THEN CS$ = CS$ +  CHR$
      (16)
1000   IF  NOT P% THEN CS$ = CS$ +
       CHR$ (6)
1010   IF J% THEN CS$ = CS$ +  CHR$
       (4)
1020   IF  NOT J% THEN CS$ = CS$ +
       CHR$ (5)
1030   IF C% < 4 THEN CS$ = CS$ +
       CHR$ (1)
1040   IF C% =  > 4 THEN C% = C% -
      3:CS$ = CS$ +  CHR$ (2)
1050  CS$ = CS$ +  CHR$ (28 + C%)
1060  CS$ = CS$ + E$ + ",B," +  STR$
      (AL%) + N$
1070  CS$ = CS$ + E$ + ",C," +  STR$
      (AG%) + N$
1080  CS$ = CS$ + E$ + ",D," +  STR$
      (AO%) + N$
1090  CS$ = CS$ + E$ + ",E," + VT$
      + N$
1100  CS$ = CS$ + E$ + ",F," + HT$
      + N$
1110  CS$ = CS$ + E$ + ",J," +  STR$
      (ML%) + "," +  STR$ (MR%) +
     N$
1120  CS$ = CS$ + E$ + ",L," +  STR$
      (FL%) + "," +  STR$ (FL% - P
      S%) + N$
1130  CS$ = CS$ + E$ + ",P," +  STR$
      (IS%) + N$
1140   IF C% THEN CS$ = CS$ +  CHR$
      (3)
1150   PRINT
1160   PRINT D$;"PR#";SL
1170   PRINT CS$
1180   PRINT D$;"PR#0"
1190   HOME : PRINT "TIGER CONFIGU
     RED": END
```

The only change an Apple owner with a Paper Tiger 460 may need to make is to change the variable SL in line 110 to indicate the slot number of his printer interface. For 460 owners with other computers, the program should be fairly easy to adapt. If you do not have reverse video through a function like Apple's 'INVERSE,' a different method of indicating the chosen option must be substituted. Also, the single keystroke method is only possible if a single key input function such as GET is available. Note that GET was also used to input the string for the tabs. On the Apple, a comma in a string INPUT results in multiple strings, not a single string, unless the entry is typed with quotes (a nuisance to be avoided).

The program consists of four parts: documentation and initialization (lines 10-200), the menu printer (lines 210-590), the keystroke interpreter (lines 600-960), and the command character transmitter (lines 970-1190). The menu printer portion looks very complicated because of the difficulties in turning inverse on and off and in maintaining the current value and state of each option.

I did find one error in my copy of the Paper Tiger 460 manual. My copy is marked 'preliminary' — hopefully it will be fixed in the permanent manual. On page 3-14 and 3-15 where it describes the 'form size' feature, table 3-4 indicates two parameters required while the description and example discuss only one. Two is the correct required number (the second one is *not* optional), so the example given will cause the printer to simply ignore the command and keep the old value of form size. The first parameter should be the total form size in 48ths of an inch as in the example. The second parameter should be the printed portion exclusive of the desired skip, also in 48ths of an inch. For example, if you want a 4.5 inch form with a one-half inch skip (thus 4 inches used for print) the correct command is:

<ESC> ,L,216,24,<CR>

TIGER SETUP allows you to indicate the skip size rather than the printed portion size, a method I find easier.

It appears that some modes of the printer interfere with others. For example, auto-justify and proportional modes cannot be used simultaneously; the proportional mode takes precedence and overrides the auto-justify mode.

Listing 2

```
:ASM
                1000 ********************************
                1005 *                              *
                1010 *          TIGER DUMP          *
                1015 *                              *
                1020 *              BY              *
                1025 *                              *
                1030 *       TERRY L ANDERSON       *
                1035 *                              *
                1040 * BEGUN           1981 JAN 09  *
                1045 * LAST MODIFIED   1981 FEB 22  *
                1050 *                              *
                1055 * WILL DUMP 8192 BYTES OF DATA *
                1060 * (USUALLY HI-RES SCREEN 1 OR  *
                1065 * 2) TO THE IDS460 PAPER TIGER *
                1070 *                              *
                1075 * USEFUL INTERNAL ADDR:        *
                1080 *                              *
                1085 *  6000 24576 ENTRY            *
                1090 *                              *
                1095 *  6001 24577 HPAG-HI BYTE OF  *
                1100 *               BUFFER TO PRINT *
                1105 *               (DEF $20)      *
                1110 *                              *
                1115 *  6040 24640 NUMLIN-# OF HI-  *
                1120 *               RES HORIZ LINES *
                1125 *               (DEF $C0=192)  *
                1130 *                              *
                1135 *  609D 24733 INVMSK-MASK BYTE *
                1140 *               $00-NORMAL (DEF)*
                1145 *               $7F-INVERSE VID *
                1150 *                              *
                1155 *  6124 24868 SLOT OFFSET--    *
                1160 *               SLOT# * $10    *
                1165 *               $10-SLOT 1 (DEF)*
                1170 *                              *
                1175 *  6125 24869 EXPANDED PLOT -- *
                1180 *               $00-NORMAL (DEF)*
                1185 *               $80-EXPANDED *2 *
                1190 *                              *
                1195 ********************************
                1200
                1300 ***  ZERO PAGE LOC'S  ***
                1310
0026-           1320 HBASL   .EQ $26         BASE ADDR FOR
0027-           1330 HBASH   .EQ $27          CUR HIRES LN
0050-           1340 ZSTOR   .EQ $50         ZPAGE STORAGE
                1350 *                        NEEDS $10 BY
0052-           1360 BASL0   .EQ ZSTOR+$2 HGR BYT PTR
0053-           1370 BASH0   .EQ ZSTOR+$3  TBL(ROW7..1)
00E6-           1380 HPAG    .EQ $E6         HBYTE OF ADDR
                1390 *                        GRAF BUFFER
                1400 ***  MON CALLS  ***
                1410
                1420
                1430 ***  A/S BASIC CALLS  ***
                1440
F411-           1450 HPOSN   .EQ $F411       SETS HBASL,H
                1460
                1470
                1480 ***  HARDWR ADDR  ***
                1490 *                       ACIA REG'S
C084-           1500 STATUS  .EQ $C084       STATUS REG
C084-           1510 CNTRL   .EQ $C084       CONTROL REG
C085-           1520 OUTPRT  .EQ $C085       OUTPORT REG
                1530
                1540 ***  OTHER CONST  ***
                1550
0020-           1560 GRBUF   .EQ $20         HBYTE OF ADDR
                1570 *                        GRAF BUFFER
00C0-           1580 NUMLIN  .EQ 192         # OF GR ROWS
000F-           1590 SAVSIZ  .EQ $F          #BYTES-1 TO
                1600 *                        SAVE FOR TBL
                1610 *  PAPER TIGER CONTROL CODES
                1620 *
0003-           1630 GRMODE  .EQ $03         CTRL-C
0003-           1640 GRPFIX  .EQ $03         CMD PREFIX
                1650 *                        WHILE IN GR
000E-           1660 GRAFLF  .EQ $0E         GRAPHIC LF
0002-           1670 NORMOD  .EQ $02         NORMAL MODE
0009-           1680 HTAB    .EQ $09         CTRL-I
0000-           1690 INVMSK  .EQ $00         INVERSE MASK
                1700 *                        USE $7F TO
                1710 *                        INVERSE PIX
0010-           1720 SLOT1   .EQ $10         SLOT OFFSET
                1730 *                        FOR SLOT 1
                1740
                1750
                1900 ***  ORIGIN  ***
                1910
                1920         .OR $6000       CALL 24576
                1930         .TA $6000
                1940
6000- A9 20     2000 DUMP    LDA #GRBUF      NORMAL ENTRY
6002- 85 E6     2010         STA HPAG        INIT HPAG
6004- A9 00     2020         LDA #00
6006- 8D 1A 61  2030         STA RONUMH      INIT ROW
6009- 8D 19 61  2040         STA RONUML
                2050
                2060 * RESET ACIA & INIT FOR DEFAULT
                2070 * 8 BIT;2 STOP; /16 CLOCK
                2080 * DISABLE INTERUPTS
                2090
600C- AC 24 61  2100         LDY SLOT        GET SLOT#
600F- A9 03     2110         LDA #$03
6011- 99 84 C0  2120         STA CNTRL,Y     RESET ACIA
```

(Continued)

Listing 2 *(Continued)*

```
6014-  A9 11           LDA #$11
6016-  99 84 C0        STA CNTRL,Y    SET DEFAULTS
6019-  A2 00    SAVEZP LDX #00        SAVE CONT OF
601B-  B5 50    SAVE1  LDA ZSTOR,X     TABLE SPACE
601D-  48              PHA             TO RESTORE
601E-  E8              INX
601F-  E0 10           CPX #SAVSIZ+1 DONE?
6021-  D0 F8           BNE SAVE1      NO,NEXT
6023-  20 E1 60        JSR PUTSTR     SEND CONTROLS
6026-  0E              .DA #GRAFLF     DUMP BUFF
6027-  03              .DA #GRMODE     GRAPHIC MODE
6028-  00              .HS 00          END STRING
                *** PRINT LINE OF 7 ROWS
6029-  A9 0E    PRLINE LDA #$0E       INIT ROW INDX
602B-  8D 1B 61        STA ROWDEX      1 OF 7 *2
602E-  A2 00    PRLIN1 LDX #$00       INIT COLUMN L
6030-  A0 00           LDY #$00              COLUMN H
6032-  AD 1A 61        LDA RONUMH     GET ROWNUM/2
6035-  4A              LSR
6036-  AD 19 61        LDA RONUML
6039-  2C 25 61        BIT DOUBLE     CHK EXPAND?
603C-  10 01           BPL PRLIN2     NO,OK
603E-  6A              ROR            YES, /2
603F-  C9 C0    PRLIN2 CMP #NUMLIN    DONE NUMLINS?
6041-  D0 06           BNE PRLIN3     NO,OK
6043-  A9 80           LDA #$80       YES,SET BIT7
6045-  8D 1A 61        STA RONUMH      INDIC DONE
6048-  38              SEC            SET DONE
6049-  CE 1B 61 PRLIN3 DEC ROWDEX     DEC 1OF7
604C-  CE 1B 61        DEC ROWDEX      TWICE
604F-  30 1C           BMI PRLIN5     ROWDEX<0;DONE
6051-  B0 1A           BCS PRLIN5     =NUMLIN;DONE
6053-  20 11 F4        JSR HPOSN      SET HBAS
6056-  AC 1B 61        LDY ROWDEX     SAVE HBAS
6059-  A5 26           LDA HBASL       FOR EACH
605B-  99 52 00        STA BASL0,Y     ROW
605E-  A5 27           LDA HBASH
6060-  99 53 00        STA BASH0,Y
6063-  EE 19 61        INC RONUML     INC ROWNUM
6066-  D0 03           BNE PRLIN4
6068-  EE 1A 61        INC RONUMH
606B-  D0 C1    PRLIN4 BNE PRLIN1     ALWAYS TAKEN
606D-  A0 27    PRLIN5 LDY #$27       INIT COLUMN
606F-  8C 1C 61        STY COLBYT      BYTE CNT
                *** PRINT 7 COLUMNS OF 7
6072-  A0 06    PR7COL LDY #$6        INIT ROW INDX
6074-  A2 0C           LDX #$0C       INIT ROW *2
6076-  A1 52    PR7C1  LDA (BASL0,X) GET GRAF BYT
6078-  99 1D 61        STA ROWBYT,Y  STO FOR PRINT
607B-  F6 52           INC BASL0,X   SET FOR NXT
607D-  88              DEY           DEC ROW
607E-  CA              DEX            & X TWICE
607F-  CA              DEX
6080-  EC 1B 61        CPX ROWDEX    =ROWDEX?
6083-  D0 F1           BNE PR7C1     NO,NXT BYTE
6085-  A0 07           LDY #$07      8 BITS,0 1ST
6087-  AD 1B 61 PR1COL LDA ROWDEX    ROWS ROWDEX.7
608A-  4A              LSR           /2
608B-  AA              TAX           USE AS INDEX
608C-  E8              INX           +1
608D-  A9 00           LDA #$00      CLR ACC
608F-  7E 1D 61 PR1C1  ROR ROWBYT,X   EACH BYTE
6092-  2A              ROL            INTO ACCUM
6093-  E8              INX
6094-  E0 07           CPX #$07      GOT 7?
6096-  D0 F7           BNE PR1C1     NO,NEXT
6098-  C0 00           CPY #$00      Y,BIT0'S?
609A-  F0 11           BEQ PR1C3     Y,DON'T PRINT
609C-  49 00           EOR #INVMSK   NO,APPLY MASK
609E-  29 7F           AND #$7F      KEEP BIT7=0
60A0-  2C 25 61        BIT DOUBLE    CHK EXPANDED?
60A3-  10 05           BPL PR1C2     NO,OUT ONCE
60A5-  48              PHA           YES,SAVE ACC
60A6-  20 D4 60        JSR OUTBYT     EXTRA OUT
60A9-  68              PLA            RESTOR ACC
60AA-  20 D4 60 PR1C2  JSR OUTBYT    MAIN OUT
60AD-  88       PR1C3  DEY           NXT COL OF 7
60AE-  10 D7           BPL PR1COL    DONE?NO, NEXT
60B0-  CE 1C 61        DEC COLBYT    NXT 7 COL'S
60B3-  10 BD           BPL PR7COL    DONE?N,NXT 7
60B5-  20 E1 60        JSR PUTSTR     Y,SEND <CR>
60B8-  03              .DA #GRPFIX     & <GR LF>
60B9-  0E              .DA #GRAFLF
60BA-  00              .HS 00           END STR
60BB-  2C 1A 61        BIT RONUMH    LAST 7 ROWS?
60BE-  30 03           BMI DONE      LAST?Y,DONE
60C0-  4C 29 60        JMP PRLINE    N,NEXT LINE
60C3-  20 E1 60 DONE   JSR PUTSTR    EXIT GRAF
60C6-  03              .DA #GRPFIX    MODE
60C7-  02              .DA #NORMOD
60C8-  0E              .DA #GRAFLF    &2<GR LF>'S
60C9-  0E              .DA #GRAFLF
60CA-  00              .HS 00         END STR
60CB-  A2 0F    RESTZP LDX #SAVSIZ   RESTORE
60CD-  68       REST1  PLA            ZPAGE USED
```

(Continued)

TIGER DUMP

We also need a way to print graphic material which has been developed on Apple's Hi-Res screen. The preliminary manual gives no information about the graphic mode except how to get into it (not even how to get out). Fortunately, I had had some experience with the Paper Tiger 440 and suspected they would be similar. The only significant differences are that the 460 prints seven dot rows (not all nine) in each head pass across the page instead of six and that <SO> or control-N is used as a 'graphic' line feed (move paper exactly seven dot rows) rather than a <VT> or control-K as on the 440.

TIGER DUMP takes data stored in Apple's RAM in Hi-Res screen buffer format and reorganizes the information to construct bytes consisting of seven dots in a column, one for each of the seven rows. A one indicates a dot that is 'on' and a zero indicates a dot that is 'off.' It then sends 280 such seven-dot columns to form one print head pass, printing seven horizontal rows. It repeats with another seven rows until all the data is printed. Unfortunately, seven does not go evenly into 192, the number of rows in Apple's Hi-Res screen. The last seven rows only have four rows of data, so zeros are assumed for the other rows and they are printed. This means that another Hi-Res screenful cannot be printed immediately, adjoining the previous one. Three blank lines will separate them. It's difficult to print larger pictures when you use multiple screenfuls. I wish the 460 would use eight print wires and use all eight bits of the data bytes. It would then run 14% faster and not have extra lines left over.

TIGER DUMP includes several features I have not seen in other graphic dump programs. These features are chosen by POKEing new values for any of five parameters. You can specify the number of lines to print, allowing only a part of the Hi-Res buffer to be printed (the part must be at the top as viewed, i.e. at beginning of buffer). You can specify the location of the buffer allowing use of Hi-Res screen two or any other 8K bytes of memory as long as it is in Hi-Res buffer format. Hi-Res buffers are organized so that lines that appear adjacent on the screen are not stored next to each other. Any data to be printed with this program must be stored exactly like a Hi-Res buffer, but it need not be in Hi-Res page one or two. This would allow several screenfuls to be BLOADed into memory wherever there is free room, and then printed.

An inverse or reverse video mask is used so you can invert a picture while printing, but the stored picture is not affected as in the programs I have seen for the 440. Several of them EOR (exclusive-or) all the bytes of the Hi-Res page before printing. TIGER DUMP simply applies the mask to each constructed byte before sending it, but does not affect the stored bytes. Each of the first seven bits of the mask byte affect one of the seven rows; a zero leaves it unaffected, a one inverts it. The mask byte $7F or $FF would invert the entire picture and $00 would print it normally. A stripped effect can be obtained by experimenting with other mask bytes. For example, $55 = 01010101 and $2A = 00101010 would invert alternate rows.

The inversion feature is particularly helpful when printing nearly 'photographic' pictures such as those in the Apple Software Bank Contributed Program Slide Shows. On the Apple screens, one-bits result in a light dot on a dark background, but on the printer, a one normally yields a black dot on white paper. The result is a print which looks like a negative. This is desirable for a line drawing. Inverting a picture gives it a more satisfying result.

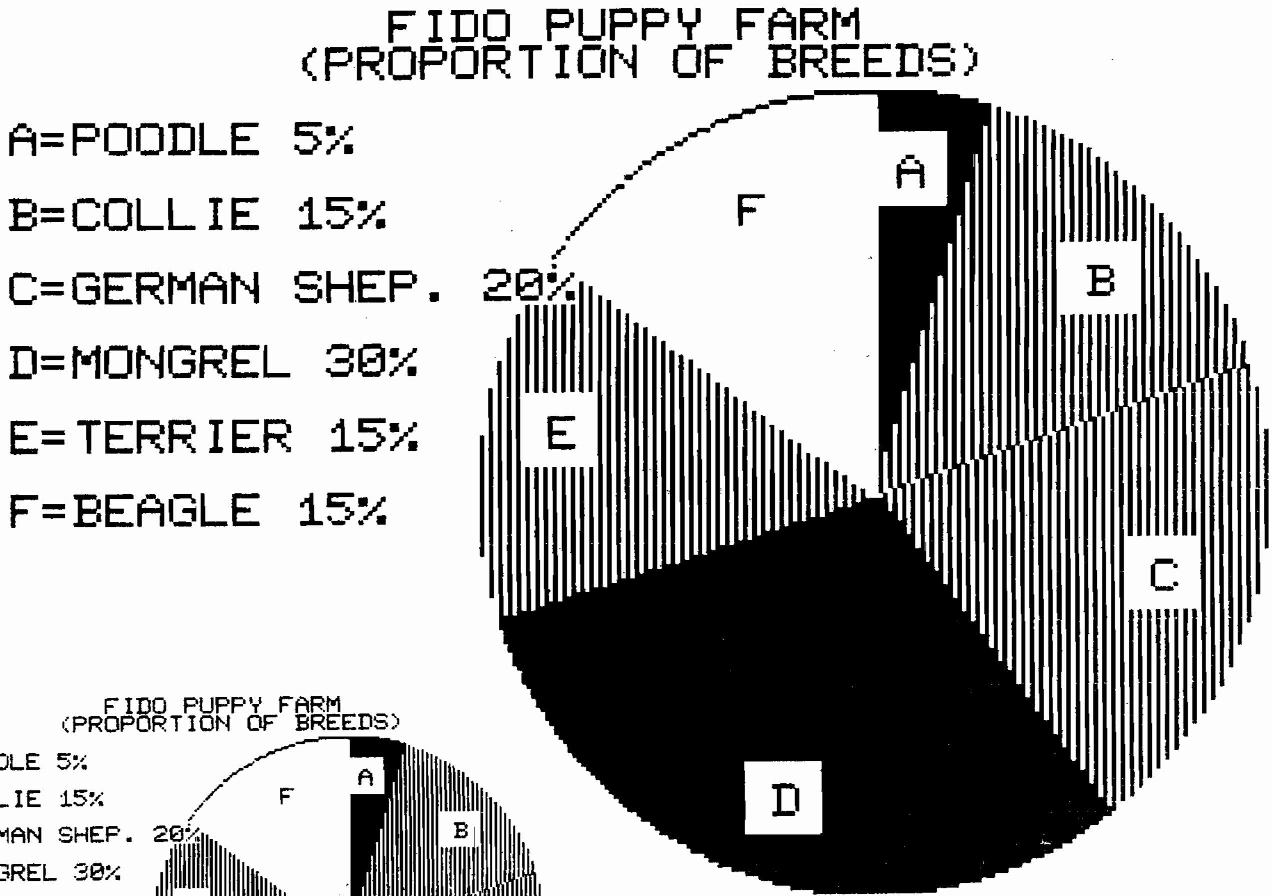
The higher resolution of the 460 compared to the 440 results in much smaller prints if you use the minimum dot spacing (84/inch) for each Hi-Res dot. The total print for 280 dots by 192 dots is only 3.33 by 2.29 inches. This is nice for some applications but often a larger print size is desirable. You could use alternate dot locations on the printer, resulting in 42 dots/inch and a print doubled in size, but that would result in white spaces between dots causing black regions to appear gray.

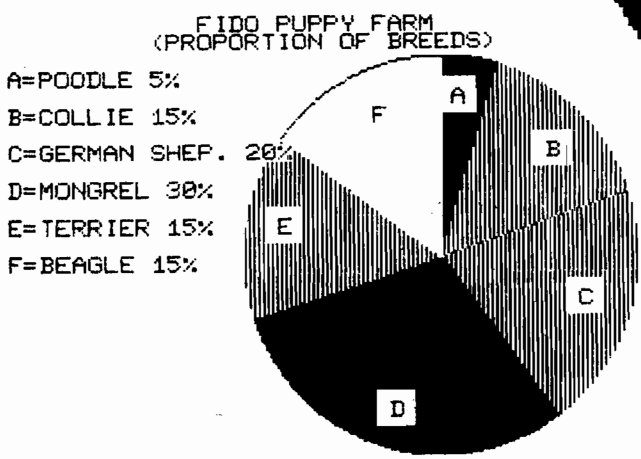
A better method is to map each Hi-Res dot into a 2 by 2 pattern of dots; each Hi-Res dot becomes a big dot. Then the dots still overlap, allowing solidly printed regions, but the image is twice as large. No additional detail is allowed though the print is larger, because no smaller detail information can be stored in Apple's Hi-Res buffer. TIGER DUMP allows the user to choose between the small size print or the expanded print with the default being the small size.

To use TIGER DUMP simply prepare the Hi-Res buffer or BLOAD a stored picture and BRUN TIGER DUMP.

Listing 2 *(Continued)*

```
60CE- 95 50              STA ZSTOR,X
60D0- CA                 DEX
60D1- 10 FA              BPL REST1
60D3- 60         RETURN  RTS                RTN FORM DUMP
                 *******************************
                 * SUBR OUTBYTE                *
                 *  OUTPUTS BYTE CHECKING FOR  *
                 *  GRAPHIC PREFIX & DBL'ING   *
                 *******************************
60D4- C9 03      OUTBYT  CMP #GRPFIX        CHK
60D6- D0 05              BNE OUTB1          NO,OUT ONCE
60D8- 20 00 61           JSR COUT           Y,OUT TWICE
60DB- A9 03              LDA #GRPFIX         FOR SECOND
60DD- 20 00 61   OUTB1   JSR COUT           PRINT IT
60E0- 60                 RTS                RETURN
                 *********************************
                 *                               *
                 *       SUBR PRINT STRING       *
                 *                               *
                 *          1981 JAN 11          *
                 *                               *
                 * SUBR WILL PRINT THE STRING    *
                 * THAT IMMEDIATELY FOLLOWS THE  *
                 * JSR AND ENDS WITH A NULL OR   *
                 * ASCII 00.                     *
                 * NOTE: USES $FE,FF FOR TEMP    *
                 *  STORAGE OF RETURN ADDR.      *
                 *                               *
                 *********************************
                 *** ZERO PAGE LOC'S
0050-            TEMPL   .EQ ZSTOR          TEMP STORAGE
0051-            TEMPH   .EQ TEMPL+1         FOR RTN ADDR
60E1- 68         PUTSTR  PLA                SAVE RTN ADDR
60E2- 85 50              STA TEMPL
60E4- 68                 PLA
60E5- 85 51              STA TEMPH
60E7- A0 00      PUTST1  LDY #$00           OFFSET
60E9- E6 50              INC TEMPL          INC POINTER
60EB- D0 02              BNE PUTST2
60ED- E6 51              INC TEMPH
60EF- B1 50      PUTST2  LDA (TEMPL),Y LOAD CHR
60F1- F0 06              BEQ PUTST3         0?Y,DONE
60F3- 20 00 61           JSR COUT           N,PRINT
60F6- 38                 SEC
60F7- B0 EE              BCS PUTST1         ALWAYS TAKEN
60F9- A5 51      PUTST3  LDA TEMPH          RESTORE
60FB- 48                 PHA                 UPDATED
60FC- A5 50              LDA TEMPL           RETURN
60FE- 48                 PHA                 ADDR
60FF- 60                 RTS
                 ***  END SUBR PUT STRING  ****
                 *************
                 * SUBR COUT *
                 *************
                 * PUTS CHAR OUT THRU ACIA
                 *  DIRECTLY
6100- 8D 18 61   COUT    STA ACCSAV         SAVE ACC
6103- 98                 TYA                 & Y REG
6104- 48                 PHA
6105- AC 24 61           LDY SLOT           INDEX BY SLOT
6108- B9 84 C0   COUT1   LDA STATUS,Y GET ACIA STAT
610B- 29 02              AND #$02            CHK READY
610D- F0 F9              BEQ COUT1           NOT? LOOP
610F- AD 18 61           LDA ACCSAV         RESTOR ACC
6112- 99 85 C0           STA OUTPRT,Y        & PUT OUT
6115- 68                 PLA                RESTORE
6116- A8                 TAY                 Y
6117- 60                 RTS                RETURN
                 *** END SUBR COUT ***
                 ***  LOCAL DATA  ***
6118-            ACCSAV .BS $1              SAVE ACCUM
6119-            RONUML .BS $1              ROWNUM
611A-            RONUMH .BS $1               HI BYTE 0..1
                 *                           $80-INDIC
                 *                           REACH NUMLIN
611B-            ROWDEX .BS $1              0..13 OR $D
611C-            COLBYT .BS $1              COLUMNBYT CNT
611D-            ROWBYT .BS $7
6124- 10         SLOT   .DA #SLOT1          SLOT OFFSET
                 *                           $NO
6125- 00         DOUBLE .DA #00             EXPANDED PLOT
                 *                          =$80;NORM =00
                 ***  END OF TIGER DUMP  ***
                 ZEND    .EN

SYMBOL TABLE

6118- ACCSAV
0053- BASH0
0052- BASL0
C084- CNTRL
```





If you wish to use any of the options:

1. BLOAD TIGER DUMP.
2. Modify $6001 or 24577 to the high byte of the buffer location if it is not Hi-Res page one; for example, for Hi-Res page two, change it to $40 or 64.
3. Modify $6040 or 24640 to the number of lines to be printed if less than 192.
4. Set the inverse mask at $609D or 24733 if you want any lines inverted; a $7F or 127 will invert the whole picture.
5. Change $6125 or 24869 from $00 to $80 or 128 if you want an expanded print.
6. Call $6000 or 24576 to run.

BSAVE TIGER DUMP INVERTED, A$6000, L$126 if you want a copy of this new version.

TIGER DUMP is located just above Hi-Res page two. If it is to be used with a BASIC program you should protect the Hi-Res pages and TIGER DUMP by setting LOMEM: 24870 or greater. This will cause variable storage to begin above TIGER DUMP. If you have an assembler, TIGER DUMP can easily be relocated to any other unused location such as just below DOS (then HIMEM should be moved to below it).

I use slot one for my printer interface. If yours is in another slot change $6110 or 24848 to $N0 or N*16 where N is your slot number.

TIGER DUMP contains its own I/O driver in a subroutine called COUT. This saves the necessity of a PR#n call to the monitor. But more importantly, the I/O driver contained in the firmware of many printer interface cards contains options which are selected by control characters. These often interfere with the 460's use of these characters. The disadvantage of providing my own I/O driver is that the TIGER DUMP is not as universal.

TIGER DUMP was written for use with the serial interface on the AIO serial/parallel interface board by SSM. For other interfaces you might have to change the locations for the output port OUTPRT and the status and control registers, STATUS and CNTRL at $C085, $C084, and $C083 respectively. Apparently some other serial interfaces are compatible. I tried the program with an unmodified California Computer Systems Asynchronous serial card and with no modifications and it worked fine at 1200 baud, but seemed to have some errors (displaced columns) at 9600 baud.

If your interface's I/O routine does not trap any of the control characters, you could eliminate my COUT. This would then allow the use of the standard driver. Simply change calls to COUT to call the monitors standard COUT at $FDED. Then you can do a PR#N before running TIGER DUMP.

The serial interface should be run at as high a baud rate as possible. Any rate of 1200 or above will allow the printer to print at near its maximum rate in the text mode. In the graphic mode at least five times as many bytes must be sent per inch of head motion (maximum of 16.8 bytes/inch in text and 84 bytes/inch in graphic). Thus even at 1200 bits/sec the printer must wait at the end of each seven row head pass for more data to be transmitted. At 9600 bit/sec, however, there is little delay; the printer is kept busy.

PUTSTR

TIGER DUMP uses a subroutine called PUTSTR that machine language programmers might find useful in other programs. It will print the string that immediately follows the JSR instruction. The string must end with a <null> or ASCII 00. I have found this a very handy way to print strings for messages and prompts in machine language programs. It takes much less memory than loading each character into the accumulator with a LDA-immediate. The subroutine gets the address of the first byte of the string from the return address on the stack. Then it loads and prints each character until a $00 is found. Then it pushes a return address on the stack that points to the first instruction beyond the string and does a return from subroutine. This routine will even print strings longer than one page, 256 bytes.

I would like to thank Dr. Claude C. Barnett, who helped me develop many of the ideas in these programs and helped test them on some of his students.

Terry Anderson is Professor of Physics and Computer Science at Walla Walla College. He teaches an introductory physics laboratory course using eight Apples for data acquisition and analysis. He also has an Apple at home which he uses for text editing, program development and, with a DC Hayes modem, as a terminal to the college's HP3000 minicomputer.

PET/CBM IEEE 488 to Parallel Printer Interface

The author presents an interface that allows a parallel printer to be connected to PET's IEEE-488 port. This maintains compatibility with PET BASIC CMD and PRINT# commands.

Alan Hawthorne
611 Vista Drive
Clinton, Tennessee 37716

Wouldn't it be nice to avoid shelling out between $65 and $150 for an interface board, plus another $50 for an IEEE 488 interface cable just to be able to interface a non-CBM printer to your PET/CBM? Well, that was the question I was faced with recently after purchasing a new CBM 8032 and 8050 disk drive along with an Integral Data System 460 Paper Tiger, which promised to provide letter-quality printout at dot-matrix speed (and price). An alternative was to use the PET/CBM parallel port for the printer and write a machine language program to output the characters to the printer. However, this solution wasn't too promising since I would not be able to use the BASIC PRINT# statement nor would I be able to list programs, which would be a considerable sacrifice. I was convinced that with a little thought, a few simple logic ICs, and a couple of spare connectors, I could make a functional IEEE-parallel printer interface, and, in addition to the challenge of the project, I could save up to $150 and still have the output features I wanted. Having been successful in the design and implementation of this project, I will describe it in the event there are other PET/CBM owners with the same need. No guarantee is made as to the conformity of the interface to IEEE standards or as to the validity of your PET/CBM warranty with the interface. However, I have successfully operated the printer interface with my CBM 8032 and 8050 disk drive, as well as a PET 2001, with no detrimental effects.

Figure 2: Simple IEEE-printer interface for use when no other IEEE-488 devices are on the bus.

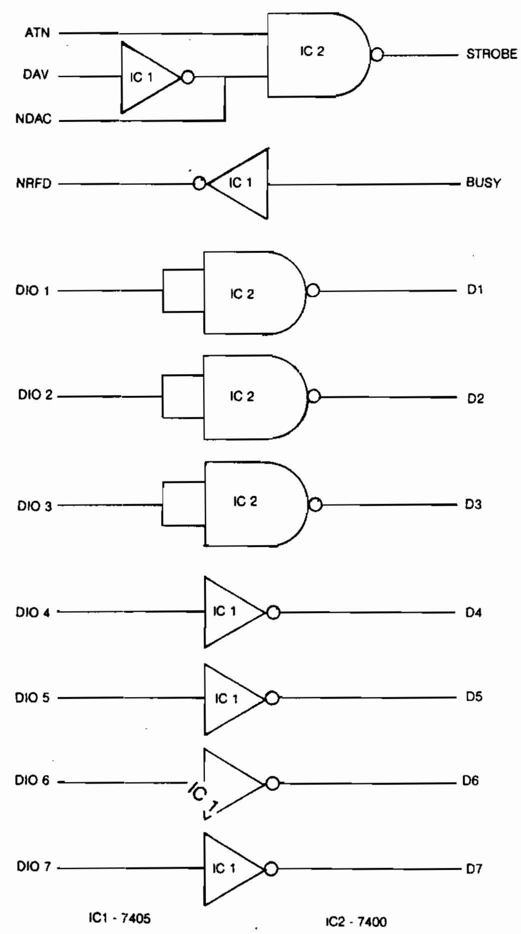
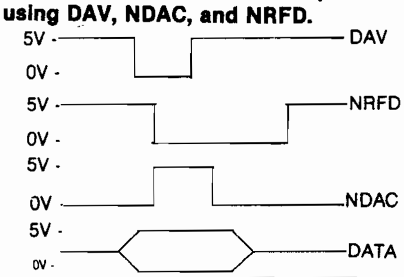


Figure 1: IEEE-488 handshake protocol using DAV, NDAC, and NRFD.



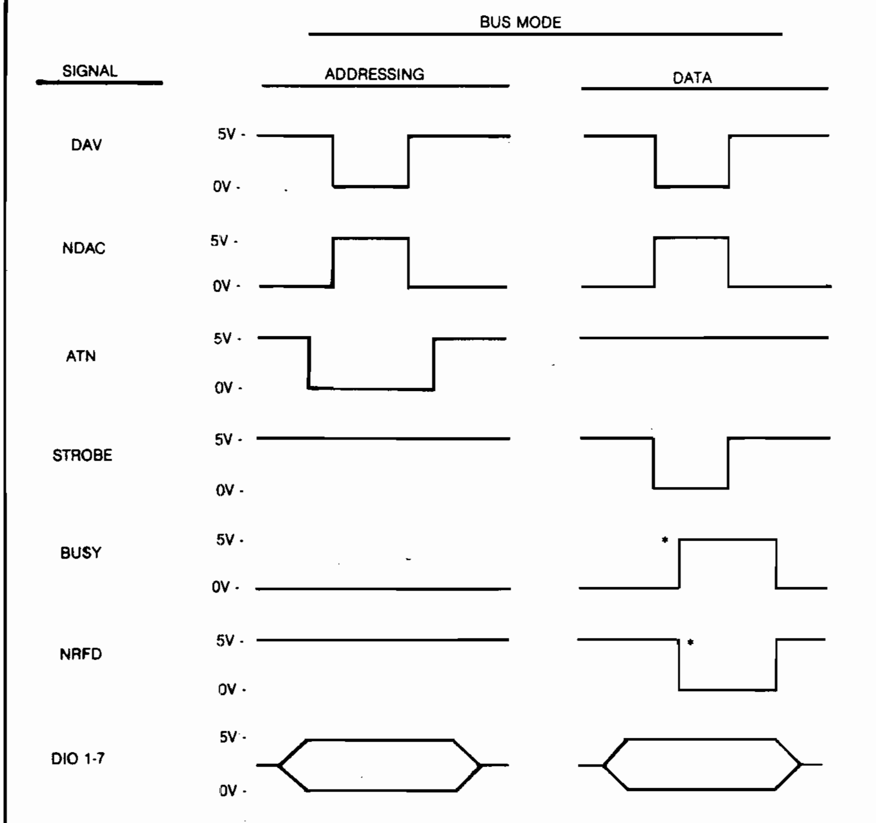
The IEEE bus consists of three types of signals: data, transfer, and management. Each device on the bus is either a talker or a listener. There are eight data lines which provide the parallel transfer of data from a talker to a listener, and also provide address information to the devices on the bus, depending on the state of the management signals. The transfer lines implement the handshaking protocol between the talkers and the listeners on the bus. There are three such signals: DAV = data valid, NRFD = not ready for data, and NDAC = data not accepted. The DAV signal originates from the talker, while NRFD and NDAC signals are provided by the listeners. Figure 1 illustrates the handshaking protocol implemented with these transfer signals.

The final group of signals consists of the management lines. There are five of these lines: IFC = interface clear, SRQ = service request, ATN = attention, REN = remote enable, and EOI = end or identify. The management signals control and indicate whether data or device addressing information is on the bus. Not all of these management signals are implemented in the PET/CBM. All bus signals are implemented as negative logic; i.e., a high level corresponds to a zero or false state, while a low level corresponds to a one or true state.

When a BASIC OPEN command is performed, the operating system tells the specified device to listen. Optionally, the secondary address and the file name may be transmitted at the same time. Likewise, a CLOSE command instructs the device associated with that logical unit to unlisten.

A PRINT# command first sends a device listen instruction, then transfers the ASCII characters of the print statement indicating the last character. Thus if a circuit could be designed which would enable data transfer to the printer when a PRINT# statement begins, and disable it at the end of the statement while not listening to other devices' data or addressing instructions, the interface would be achieved.

Figure 3: Timing diagram associated with the interface circuit in figure 2.



* Busy only if printer buffer is full.

Interface Design

Figure 2 shows a simple interface which will work with the PET/CBM IEEE port when no other device (including a disk drive) is on the bus. The associated timing diagram is presented in figure 3. The interface is implemented with only two ICs, a 7400 quad dual-input nand gate and a 7405 hex inverter with open-collector outputs. Open-collector outputs are used in order for the NDAC and NRFD handshake signals to be wire ORed with other devices. If your printer will operate with negative logic, then the inverting of the data lines will not be necessary. When addressing information is on the data bus, the ATN line will be held low; while data is on the bus the ATN line remains high. The arrangement in figure 2 will strobe the printer on when ATN*DAV is true, thus providing the needed decoding to distinguish between data and addressing information on the IEEE bus.

When the printer buffer is full, the printer BUSY line provides the necessary handshake signal to NRFD to allow the computer to wait until the printer is no longer busy. This circuit indicates to the PET/CBM that data is accepted as soon as the IEEE DAV goes low. This requires the printer to latch the data within the time that DAV is low, whereas if implemented as a true IEEE device, the computer would wait until the printer acknowledged receipt of the data. This should not be a limitation for most parallel printers but may be a point to test if the interface doesn't work for you.

If another IEEE device, such as a disk drive, is present, then the simple two-chip circuit of figure 2 will not be adequate to interface the printer. Additional circuitry will be required to decode device addressing. The address decoding is accomplished with a 7470, which is an AND-gated J-K positive-edge-triggered flip-flop with preset and clear. Figure 4 shows the function table for this IC.

For the PET/CBM peripherals, the normal IEEE device addresses are an 8 for the disk drive and a 4 for the printer. These device addresses are assumed in the printer interface design shown in figure 5. As shown in figure 4, Q will be set high on the positive edge of the clock pulse if the J input is high and the K input is low. Likewise, Q will be set low on the positive edge of the clock pulse if the J input is low and the K input is high. Also Q is set low if the clear input is brought low. These three functions allow the address decoding to be accomplished with only this one IC when the Q output is NANDed with the DAV and ATN bus signals. The appropriate clocking pulse is obtained by NANDing the ATN and DAV signal so that a clock pulse occurs when valid addressing signals are on the IEEE bus. The clock does not function when valid data is on the bus.

When the PET/CBM outputs data to the IEEE port via a PRINT# statement, the following address bytes (ATN low) are output first: a \$2*x*, where *x* is the device address, and a \$6*y*, where *y* is the secondary address specified in the OPEN statement. An OPEN statement gives a

Figure 4: Functions of a 7470 and-gated J-K positive-edge-triggered flip-flop.

SET	CLR	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
H	H	↑	L	L	Q	$\overline{Q}$
H	H	↑	H	L	H	L
H	H	↑	L	H	L	H
H	H	↑	H	H	TOGGLE	
H	H	L	X	X	Q	$\overline{Q}$

↑ — Positive transition.
X — Either level.

Figure 5: IEEE-printer interface with address decoding capability.

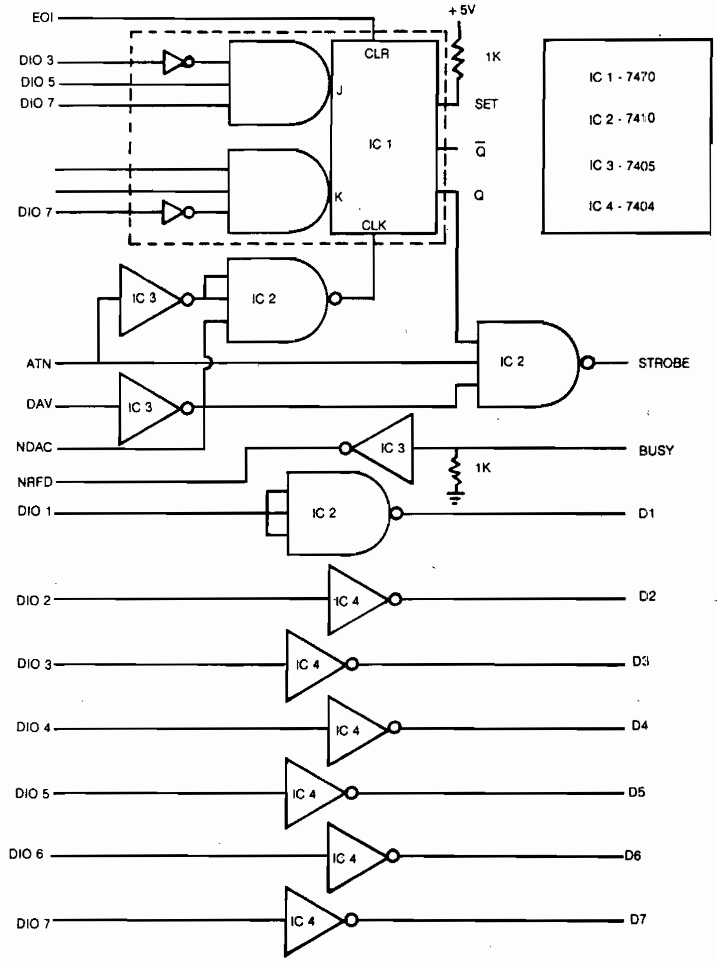
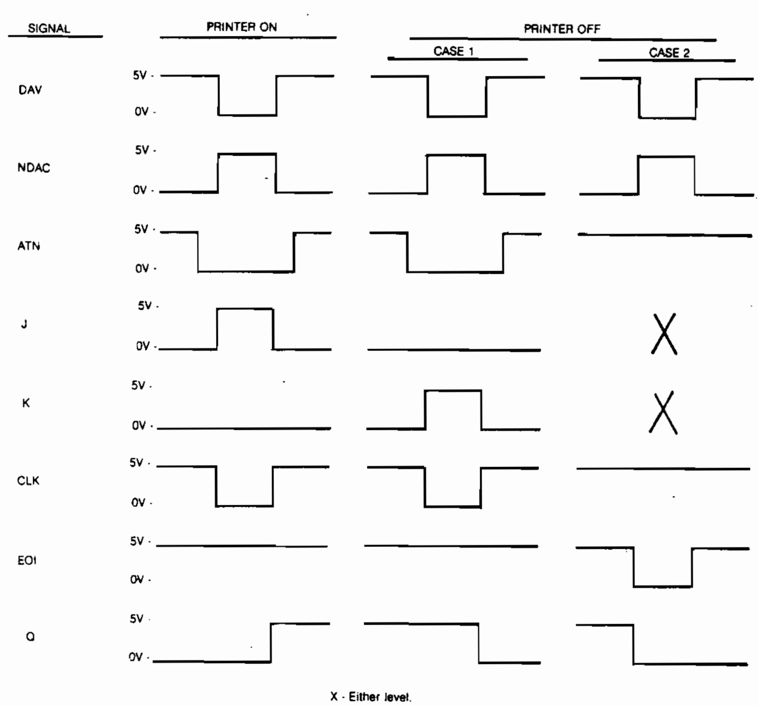


Figure 6: Timing diagram associated with the interface circuit in figure 5.



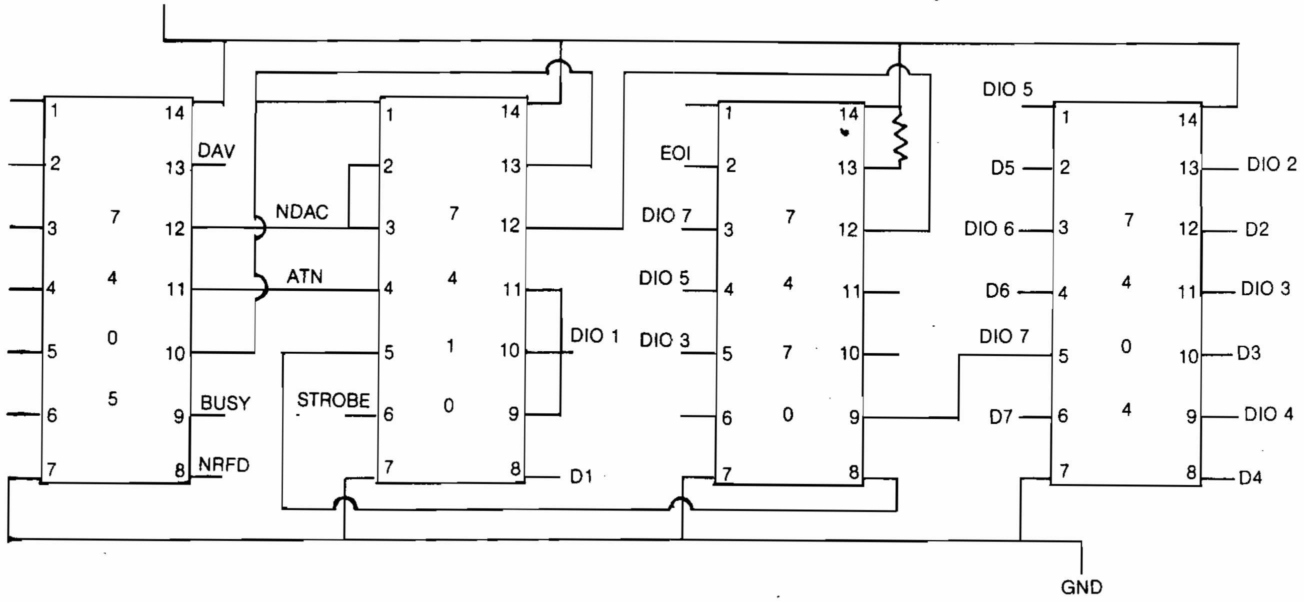
$2*x* (*x* is device address) followed by a $F*y* (*y* is secondary address), and a CLOSE statement gives a $2*x* followed by a $E*y*. The EOI line is brought low concurrent with the last transmitted data byte. Complete address decoding is not accomplished with the 7470 but sufficient lines are decoded to allow the interface to recognize a $24 (printer address) and to gate the printer on (i.e., set Q high). When the last data character of a PRINT# statement is transmitted, the EOI signal gates the printer off (i.e., set Q low). Since the $24 address code is also transmitted when an OPEN or a CLOSE statement is executed, bit 6 is used to toggle the flip-flop back low and gate the printer off once again. This is necessary in order to prevent the printer from remaining on line after an OPEN or CLOSE, which can certainly give strange behavior when communicating with a disk drive. Figure 6 illustrates the timing diagram for the interface and should make the functional operation of the interface easier to understand.

Figure 7 is a wiring layout for the printer interface. The circuit is constructed on a small piece of PC board with one side being a 24-pin edge duplicating the physical IEEE port of the PET/CBM. The other side of the interface box contains a 24-pin edge connector which plugs onto the computer IEEE port. The IEEE bus signals are passed through the box, allowing the PET-IEEE cable to be used with the interface as it was used with the computer. I used a spare 15-pin D connector for attachment to the printer. The 5-volt supply to operate the circuit was obtained from the cassette interface at the rear of the PET/CBM.

Final Comments

As I mentioned, this PET/CBM IEEE to parallel printer has worked well for me using an Integral Data System Paper Tiger with my CBM system, as well as with a PET. However, let me warn of some potential problems and limitations. First of all, the interface does not transmit the last character of the data to be printed. This is not a particularly troublesome problem if the computer transmits a carriage return and a line feed, and the printer functions with only a carriage return. The PET I have sends both a carriage return and a line feed. However, the CBM 8032 sends the line feed only if the file number is 128 or greater. This could lead to some editing of existing programs to change file numbers so that a line feed is sent. Alternatively, additional hardware could be added so that the 7470 clear

Figure 7: Wiring layout for interface circuit in figure 5.



line is set low on the positive-going edge of the EOI signal. You must decide if the inconvenience is worth the additional hardware.

An additional area where a problem might arise is the device address decoding. Should additional IEEE devices such as a modem be attached to the bus, care must be exercised to ensure that none of the addresses are decoded by this circuit. For instance, any device whose address contains bit 2 will output to the printer; thus 4, 5, 6, and 7 are device addresses which will gate the printer on. Once again, additional hardware can be added to provide complete decoding.

One final point of caution concerns the handshake implementation. The pull-down resistor on the busy line allows the IEEE bus to operate with the printer turned off or disconnected from the interface. However, this implementation rather defeats the benefits of having handshaking, in that complete handshaking with the computer occurs even when the printer is not present. I much prefer to be able to use my disk drive with the printer turned off and don't consider it much of a shortcoming.

MICRO

An Inexpensive Printer for Your Computer

Even the very low budget computer hobbyist can have a printer to list his programs and data. Described here is an inexpensive printer mechanism and how it works. A simple circuit and software are included that will allow this printer to be interfaced to your 6502's parallel I/O port.

Michael J. Keryan
713 Locust Drive
Tallmadge, Ohio 44278

Many computer hobbyists have no hard copy output device. The main reason is the price of printers; all but a few cost nearly as much as the computer itself. This is a shame, since much time is wasted copying programs and data back and forth from paper to keyboard, to CRT display, to paper. In this article, a printer, interface circuitry, and 6502 driver software are described. Assuming you have a microcomputer with a PIA and 768 bytes of spare memory, you can add this printer to your microcomputer for about fifty dollars.

The printer mechanism is a Sharp DC-1606A, recently offered by an electronics surplus dealer, (John Meshna Jr., of Lynn, Mass.), for $20. The printer uses aluminized paper and gives printed copy similar to Radio Shack's $219.00 Quick Printer II. Although not acceptable for some applications, the print is readable and useful for program and data documentation and output of programs such as checkbook balancers. The software given will print 96 characters (upper and lower case) in a five by eight matrix. The character widths are variable from five or less characters per line (for headings) to a maximum of forty-two characters per line.

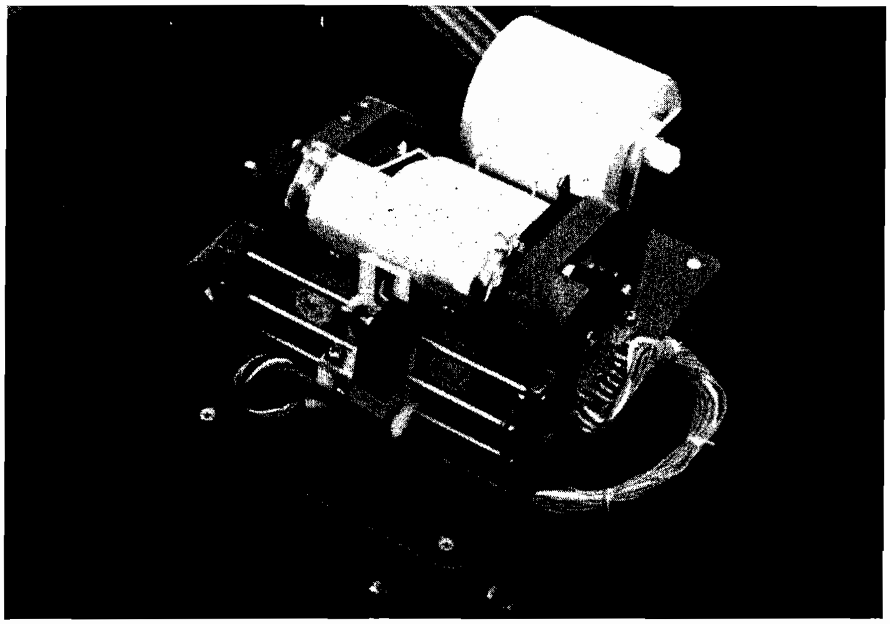


Photo 1: **The printer mounted on the box containing the interface circuit and power supply.**

How the Printer Works

The paper is coated with a very thin layer of aluminum, which can be burned away by electric current, leaving an almost black surface. The print head consists of a vertical column of eight elements that are in physical contact with the paper as the head traverses from left to right. If a sufficient current source is applied to the conductive aluminum surface of the paper, providing a ground through which the elements will burn away the coating, a black dot or line will be produced. Any desired character can thus be formed by turning each of the eight elements on and off at the right times.

An open loop system with character widths being a function of a timing pulse would be the simplest way to get the dots to form characters, but this is not practical. The horizontal speed of the print head is not constant and an open loop system would give unequal character widths. However, a feedback system is extremely simple to interface, using the strobe systems in the printer mechanism. Assuming the motor is turned on and the print head is in the process of printing a line of text, the head travels from left to right across the paper. At the right margin, the print head is automatically lifted from the paper surface and the head then moves from right to left. During this motion, the platen also indexes the paper to the next horizontal line position. Therefore, carriage return and line feed occur after each line. At the left margin, the print head is lowered to the paper surface to begin the left to right scan for the next line. This is shown in figure 1.

Within the print mechanism are two strobe wheels, which can block light paths between lamps and associated

***Figure 1:* Motion of the print head to form a printed line with automatic carriage and line feed.**

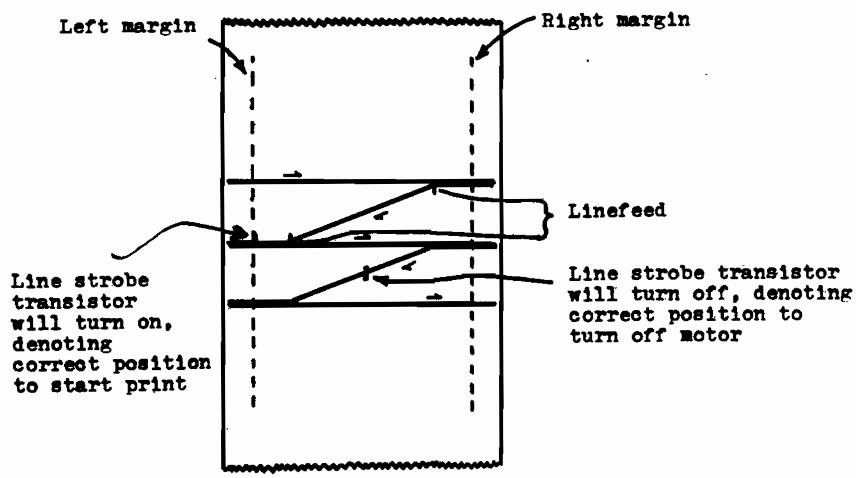


photo-transistors. The *line strobe* wheel begins to allow the light to turn on the line strobe transistor at the left margin, as the print head is moving toward the right. A transition in this transistor from off to on denotes the beginning of a line. The transistor remains on until mid-position in the carriage return. A transition of the transistor from on to off denotes the proper position to turn off the motor when printing one line. The print head will then remain in this position until another line is ready to be printed.

The *character strobe* is similar to the line strobe but contains many more lugs on a faster spinning wheel. The character strobe photo-transistor outputs a square wave of approximately 126 pulses as the print head moves to the right between margins. Although the pulse width is not constant as a function of time, it is constant as a function of movement of the print head. Therefore, turning the printhead elements on and off at the right time is merely a matter of synchronizing the output signals to the character strobes. Character widths can be varied by allowing varying integral half-cycles of the character strobe to represent a vertical column. Horizontal spacing between characters can be varied similarly. The right margin is located by counting the character strobe pulses, or alternately by counting the number of character spaces and adjusting the maximum number of characters for the pulse count per character.

Line character width will be determined by vertical column width and spacing between characters. Using five-by-eight matrices for the characters (five vertical columns, each eight segments high) and assuming that the column widths for spacing are equal to the printed column width, the maximum number of characters per line can be represented as a function of width and spacing:

$$C = INT \frac{253 + WS}{W(5 + S)}$$

where C = number of characters/line,

W = number of half cycles of character strobe per vertical column, and

S = number of blank vertical columns after each character.

Some examples of print size are shown in table 1. In general, line lengths of from sixteen to twenty-one characters can be considered normal. Line lengths shorter than sixteen might be used for headings, while those larger than twenty-one would result in narrow, closely spaced characters, which are difficult to read without inserted spaces. The print mechanism also contains several microswitches and other features, best described in conjunction with the interface circuit.

The Interface Circuit

The interface circuitry is shown in figure 2. It can be used to interface the printer to a PIA, VIA, or TTL input/output port. (A PIA was used in the prototype.) Eight output bits are required for the print head and one output bit drives the motor control circuit. Also required are four input bits for feedback to the computer. The numbers shown at the connection points between the printer mechanism and the interface circuit refer to the numbered pins on the edge connector provided with the printer.

***Table 1:* Variation in print size. Listed are values of C (Number of characters/line), W (Width = number of half cycles of character strobe/vertical column), and S (Space = number of blank vertical columns following character), followed by one line of text at that spacing.**

C	W	S	Text
42	1	1	ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghij
36	1	2	ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcd
32	1	3	ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'
21	2	1	ABCDEFGHIJKLMNOPQRSTU
18	2	2	ABCDEFGHIJKLMNOPQR
16	2	3	ABCDEFGHIJKLMNOP
14	3	1	ABCDEFGHIJKLMN
12	3	2	ABCDEFGHIJKL
10	3	3	ABCDEFGHIJ
10	4	1	ABCDEFGHIJ
9	4	2	ABCDEFGHI
8	4	3	ABCDEFGH
8	5	1	ABCDEFGH

As already described, a positive voltage is applied to the paper surface. A return to ground through the transistors will result in a printed dot or line. The transistors are driven by inverter sections of IC1 and IC2 (4049's). These CMOS IC's are ideal for this use since they are compatible with five volt MOS or TTL levels, and are virtually indestructable.

The positive voltage at the paper surface is sampled by two elements. When the paper runs out, the voltage at these pins will drop to zero. These pins are connected through protection and noise elimination networks to pins 11 and 12

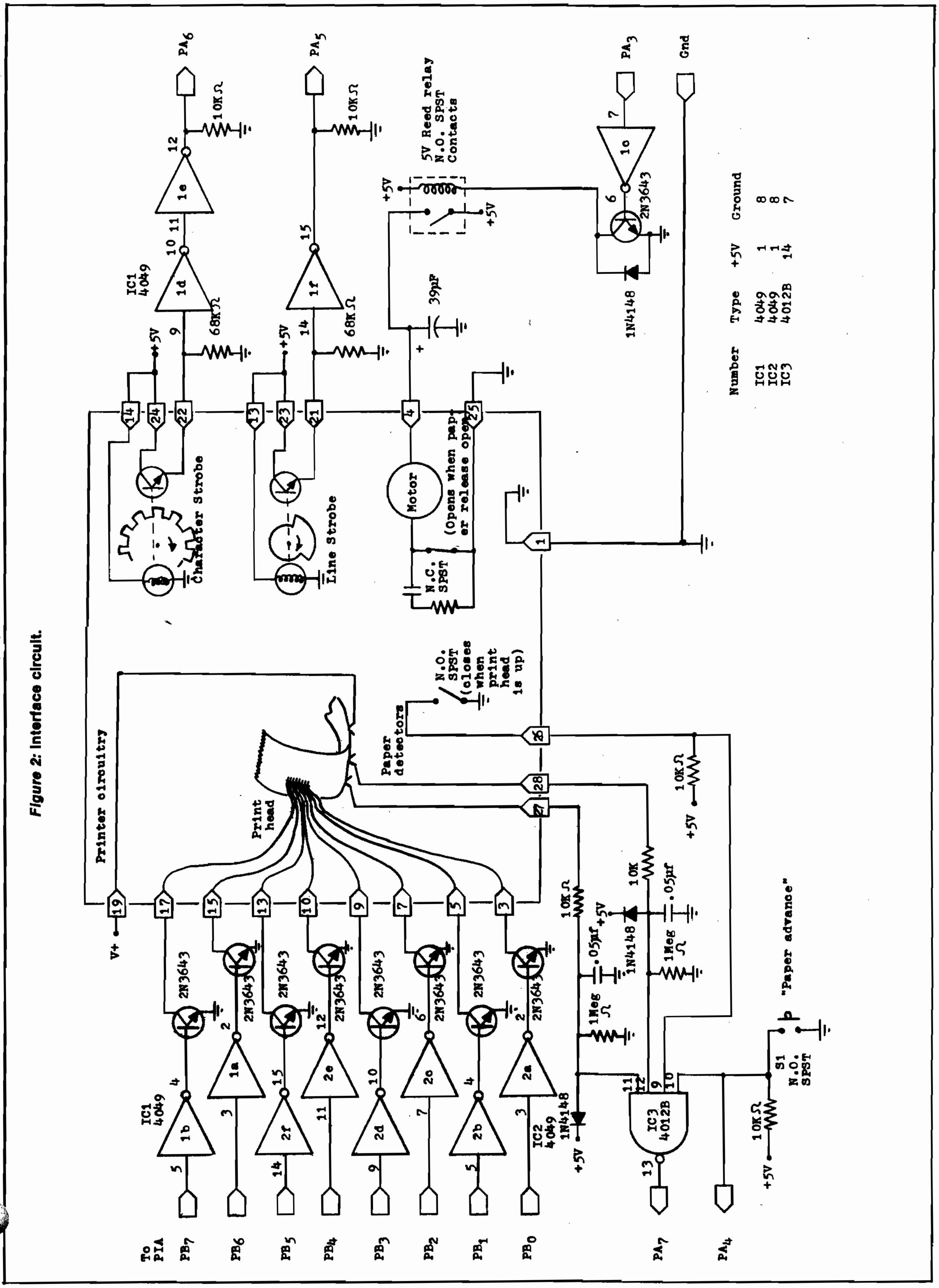


Figure 2: Interface circuit.

of IC3. This nand gate has two more inputs. Pin 9 is connected to a normally open switch within the printer, that closes a circuit to ground when the print head is manually lifted from the paper by sliding back the plastic guard. Pin 10 is connected to S1, a normally open SPST switch added to the interface. Zero volts on any of these inputs will cause the output of the nand gate, connected to PA7, the status bit, to go high, indicating some sort of problem. S1 is also connected, to PA4, useful as a paper advance (line feed) request.

The motor runs well at 5 volts, but not at 4.5 volts. Therefore, a reed relay is used to switch the 5 volts to the motor. An electrolytic capacitor is added to the motor connection to slightly slow down the transition from five to zero volts, removing the need for noise elimination near the cross-over point of the line strobe. PA3 drives the motor control circuit, buffered through an inverter and transistor. A zero volt level on PA3 will turn on the motor.

The lamps and the collectors of the strobe transistors are connected to the +5 volts. The emitters are brought to ground through 68Kohm resistors. The voltages generated across these resistors are buffered by CMOS inverters. The outputs of these inverters are pulled to ground through 10K resistors and are connected to PA6/PA5 for the character/line strobes, respectively. These resistors ensure the outputs to be at a zero volt level when no power is applied to the interface circuit.

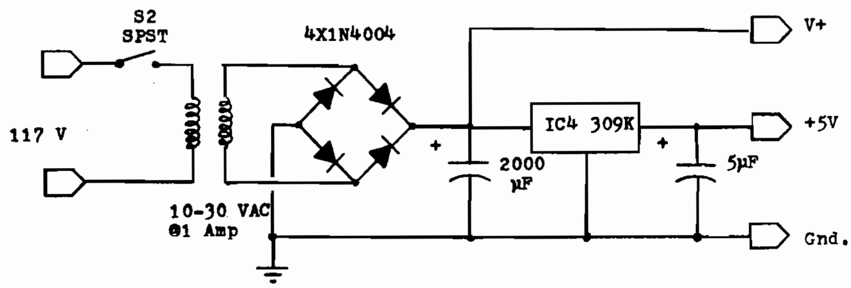
The power supply, shown in figure 3, is very simple and needs little explanation. The transformer can have an output voltage of ten to thirty volts. Higher voltage will give darker print but will require a higher voltage rating for the 2000uF capacitor and more heat sinking for IC4, the voltage regulator. The prototype circuit used a twelve volt, one amp transformer.

PA0, PA1, and PA2 are not used. If desired, they could be configured for increased input/output control. One use would involve circuitry to control the power supply, by replacing S2 (the power switch) by a relay or solid-state switch.

The Software

The software shown in listing 1 was written for a 6502-based OSI C2-4P, but will require only minor modifications for other 6502 computers. A buffer area of programmable memory is required to hold one line of characters before printing. The beginning of the buffer is set to

Figure 3: **Power supply for printer and interface.**



Listing 1: **6502 matrix print routine.**

```
                ;* INEXPENSIVE PRINTER DRIVER
                ;*
                ;*      BY M.J. KERYAN
                ;*
                LINLEN EPZ $E0           ;NO. CHARACTERS/LINE
                WIDTH  EPZ $E1           ;NO. PULSES/COLUMN
                SPACE  EPZ $E2           ;BLANK COL./CHAR.
                CHRCNT EPZ $E3           ;CHARACTER COUNT
                STROBE EPZ $E4           ;CHAR. STROBE FLAG
                TEMP   EPZ $E5           ;TEMP. REGISTER
                TABLEA EPZ $E6           ;COLUMN 1 VECTOR (LO,HI)
                TABLEB EPZ $E8           ;COLUMN 2 VECTOR
                TABLEC EPZ $EA           ;COLUMN 3 VECTOR
                TABLED EPZ $EC           ;COLUMN 4 VECTOR
                TABLEE EPZ $EE           ;COLUMN 5 VECTOR
                ;
                BUFFER EQU $D3C4         ;CHARACTER STORAGE
                BUFEND EQU $D3FF         ;END OF BUFFER
                ;
                PIADA  EQU $F700         ;DATA REGISTER A
                PIACA  EQU $F701         ;CONTROL REGISTER A
                PIADB  EQU $F702         ;DATA REGISTER B
                PIACB  EQU $F703         ;CONTROL REGI8TER B
                ;
                       ORG $8000
                ;
8000 48         PRTOUT PHA               ;SAVE COLUMN
8001 A5E1              LDA WIDTH         ;IS WIDTH =0?
8003 F011              BEQ SUBOUT        ;YES, RETURN
8005 85E5              STA TEMP          ;SAVE WIDTH
8007 AD00F7     WAIT   LDA PIADA         ;CHECK CHAR. STROBE
800A 2940              AND #%01000000    ;MASK
800C C5E4              CMP STROBE        ;SAME?
800E F0F7              BEQ WAIT          ;YES, LOOP AND WAIT
8010 85E4              STA STROBE        ;NO, RESET FLAG
8012 C6E5              DEC TEMP          ;WIDTH=WIDTH-1
8014 D0F1              BNE WAIT          ;LOOP IF <>0
8016 68         SUBOUT PLA               ;RECALL COLUMN
8017 8D02F7            STA PIADB         ;OUTPUT IT
801A 60                RTS               ;RETURN
801B AD00F7     LINDET LDA PIADA         ;CHECK LINE STROBE
801E 2920              AND #%00100000    ;MASK FOR LINE STROBE
8020 60                RTS               ;RETURN
8021 85E5       PRINTM STA TEMP          ;SAVE CHARACTER
8023 48         PRINTB PHA               ;SAVE FOR RETURN
8024 8A                TXA               ;SAVE X REGISTER
8025 48                PHA
8026 98                TYA               ;SAVE Y REGISTER
8027 48                PHA
8028 A2FF              LDX #$FF
802A A900              LDA #$00
802C 8D01F7            STA PIACA         ;DATA DIRECTION A
802F 8D03F7            STA PIACB         ;DATA DIRECTION B
8032 85E4              STA STROBE        ;INIT. STROBE FLAG
8034 A908              LDA #%00001000
8036 8D00F7            STA PIADA         ;MOTOR BIT=OUT
8039 8E02F7            STX PIADB         ;PRINTHEAD=OUT
803C A904              LDA #$04
803E 8D01F7            STA PIACA         ;DATA REGISTER A
```

Listing 1 *(Continued)*

```
8041 8D03F7          STA PIACB          ;DATA REGISTER B
8044 8E00F7          STX PIADA          ;MOTOR OFF
8047 8E02F7          STX PIADB          ;PRINTHEAD OFF
804A A20A            LDX #$0A           ;TRANSFER TABLE
804C BDF680   TRANSF LDA ROMTAB,X       ;POINTERS TO PAGE
804F 95E6            STA TABLEA,X       ;ZERO MEMORY
8051 CA              DEX
8052 D0F8            BNE TRANSF         ;DONE? IF SO,
8054 201B80          JSR LINDET         ;IS POWER ON?
8057 F004            BEQ OUT            ;IF NOT, RETURN
8059 A5E0            LDA LINLEN         ;O.K.,IS LINELEN=0?
805B D002            BNE CHKSTP         ;NO, CONTINUE
805D F027     OUT    BEQ RETRUT         ;OTHERWISE RETURN
805F AD00F7   CHKSTP LDA PIADA          ;CHECK STATUS
8062 101A            BPL BUILD          ;IF O.K., BRANCH
8064 2910            AND #%00010000     ;PAPER ADVANCE?
8066 D0F7            BNE CHKSTP         ;NO, THEN WAIT TILL O.K.
8068 A900            LDA #$00
806A 8D00F7          STA PIADA          ;TURN ON MOTOR
806D 201B80   LNFDA  JSR LINDET         ;CHECK LINE STROBE
8070 D0FB            BNE LNFDA          ;WAIT IF <>0
8072 201B80   LNFDB  JSR LINDET         ;CHECK LINE STROBE
8075 F0FB            BEQ LNFDB          ;WAIT IF =0
8077 A908            LDA #%00001000     ;LINEFEED COMPLETE,
8079 8D00F7          STA PIADA          ;STOP MOTOR
807C D0E1            BNE CHKSTP         ;RECHECK STATUS
807E A5E5     BUILD  LDA TEMP           ;GET ASCII CHAR.
8080 297F            AND #%01111111     ;MASK OFF HIGH BIT
8082 C90D            CMP #$0D           ;CARRIAGE RETURN?
8084 F013            BEQ FILL           ;YES, FILL BUFFER
8086 C920     RETRUT CMP #$20           ;LEGITIMATE CODE?
8088 3063            BMI RETURN         ;IF NOT, RETURN
808A A6E3            LDX CHRCNT         ;CURRENT BUFFER LEN.
808C 9DC4D3          STA BUFFER,X       ;ADD CHAR. TO BUFFER
808F E6E3            INC CHRCNT         ;NEW BUFFER LENGTH
8091 A5E0            LDA LINLEN         ;MAXIMUM LENGTH
8093 C5E3            CMP CHRCNT         ;IS BUFFER FULL?
8095 F00E            BEQ LINOUT         ;YES, OUTPUT LINE
8097 D054            BNE RETURN         ;NO, THEN RETURN
8099 A6E3     FILL   LDX CHRCNT         ;CURRENT BUFFER LEN.
809B A920            LDA #$20           ;ASCII SPACE
809D 9DC4D3   LOOPFL STA BUFFER,X       ;PLACE IN BUFFER
80A0 E8              INX                ;NEXT LOCATION
80A1 E4E0            CPX LINLEN         ;LAST?
80A3 D0F8            BNE LOOPFL         ;NO, CONTINUE
80A5 A200     LINOUT LDX #$00           ;START LINE OUTPUT
80A7 8E00F7          STX PIADA          ;TURN ON MOTOR
80AA A003            LDY #$03           ;LOOP COUNTER
80AC 201B80   LNDT   JSR LINDET         ;CHECK LINE STROBE
80AF D0FB            BNE LNDT           ;WAIT TILL=0
80B1 88              DEY                ;REPEAT 3 TIMES
80B2 D0F8            BNE LNDT           ;SO YOU ARE SURE
80B4 BCC4D3   LOOPOU LDY BUFFER,X       ;GET CHARACTER
80B7 B1E6            LDA (TABLEA),Y     ;COLUMN 1 CODE
80B9 200080          JSR PRTOUT         ;OUTPUT IT
80BC B1E8            LDA (TABLEB),Y     ;COLUMN 2 CODE
80BE 200080          JSR PRTOUT         ;OUTPUT IT, ETC.
80C1 B1EA            LDA (TABLEC),Y
80C3 200080          JSR PRTOUT
80C6 B1EC            LDA (TABLED),Y
80C8 200080          JSR PRTOUT
80CB B1EE            LDA (TABLEE),Y
80CD 200080          JSR PRTOUT
80D0 A4E2            LDY SPACE          ;NO. OF BLANK COLUMNS
80D2 A9FF     LOOPBL LDA #$FF           ;BLANK CODE
80D4 200080          JSR PRTOUT         ;OUTPUT BLANK COL.
80D7 88              DEY                ;DONE?
80D8 D0F8            BNE LOOPBL         ;NO, DO IT AGAIN
80DA E8              INX                ;NEXT CHARACTER
80DB E4E0            CPX LINLEN         ;IS THAT ALL?
80DD D0D5            BNE LOOPOUT        ;NO, LOOP & CONTINUE
80DF 201B80   CRLF   JSR LINDET         ;CHECK LINE STROBE
80E2 F0FB            BEQ CRLF           ;IF =0, WAIT
80E4 A908            LDA #%00001000
80E6 8D00F7          STA PIADA          ;STOP MOTOR
80E9 A900            LDA #$00
80EB 85E3            STA CHRCNT         ;RESET CHAR. COUNTER
80ED 68       RETURN PLA                ;RESTORE REGISTERS
80EE A8              TAY
80EF 68              PLA
80F0 AA              TAX
80F1 68              PLA
80F2 60              RTS
80F3 EA              NOP
80F4 EA              NOP
80F5 EA              NOP
80F6          ;
80F6 E080     ROMTAB ADR $80E0          ;TABLE POINTERS WILL BE
80F8 4081            ADR $8140          ;TRANSFERRED TO PAGE ZERO
80FA A081            ADR $81A0          ;MEMORY BY PROGRAM
80FC 0082            ADR $8200
80FE 6082            ADR $8260
8100          ;
```

$D3C4, which in my OSI system corresponds to the unused lower two lines of the video refresh memory. This allows the buffer to be viewed on the CRT prior to printing. Also needed are sixteen bytes of page zero programmable memory, located at hexadecimal locations 00E0-00EF, also not used by OSI routines. Three of these must be set up prior to calling the print subroutine. They can be changed between lines if desired, but must all be greater than zero:

$00E0: (C) = number of characters/line,
$00E1: (W) = width of vertical column,
$00E2: (S) = spacing, number of blank columns/character.

Locations $00E3-$00E5 are temporary registers. Locations $00E6-$00EF are pointers to the character decoding tables. These are written from the upper ten bytes of the 256-byte program each time the program is called, so they can be used for other purposes between callings of the print subroutine. The PIA is configured at locations $F700-$F703, as on the OSI 500 CPU board. The program itself is located at $8000-$80FF, and the character decode tables start at $8100, shown in listing 2. There are actually five of these tables, each 96 bytes long, the first table corresponding to vertical column one of ASCII characters $20-$7F, the second table corresponding to the second vertical column, etc. To fill out the last page, a screen clear program starts at $82E0; this is useful only for OSI systems.

The main program is commented and therefore little explanation is necessary. There are two entry points. If the character to be printed is in the accumulator, enter the program by a JSR $8021. If the character is not in the accumulator, it should be written into $00E5 by either a machine language routine or a BASIC POKE statement, then the program entered by a JSR $8023. The subroutine will restore all registers before returning. To modify the program to other 6502 configurations, only the three-byte instructions and the table pointers (upper 10 bytes) will need to be changed.

When entered, the program initializes the PIA and the strobe flag, then copies the table pointers to page zero. It then checks to see if the power to the printer is on and if the carriage is in the correct position. If not, it will then return. Next, it checks the status bit. If not OK, it will then check to see if paper advance is requested (by a closure of S1). If so, it will line feed until S1 opens. If not, it will wait until the status is OK.

The high bit of the character is then masked off and it is checked. If it is a carriage return ($0D), the remainder of the buffer will be filled with blanks ($20) and a line of text will be output. If the ASCII code is not legitimate (less than $20), it will then return. Otherwise, it will add the character to the buffer and check to see if the buffer is full. If full, it will output a line; otherwise, it will return. Note that nothing is printed unless the buffer is full or the character is a carriage return.

In my system, the printer routine is called every time a character is output to the cassette tape port. This was accomplished by a jumper from the UART TDS (pin 23) to the NMI bus line. The following code is entered at the NMI vector: $0130.

```
$0130 20 21 80  JSR $8021
$0133 40        RTI
```

For a C1P, the same thing can be accomplished by merely changing the output routine vector (located at $021A-$021B) to point at the following code:

```
20 21 80 JSR $8021
4C 69 FF JMP $FF69
```

The printer routine will then be executed prior to the normal output routine. In either case, a change in $E0 from a zero to a non-zero value will enable the print routine. When in the SAVE mode, everything on the CRT will be printed. Alternately, a BASIC USR call can print selected material.

Either programmable memory or erasable read-only memory can be used for program storage, but read-only memory is much more convenient. There is an additional benefit to having the character code conversion table in memory. All your other programs can then have access to the codes, for large titles on your CRT, or whatever.

The program in listing 3, written in OSI BASIC, will demonstrate the 96 characters on the CRT display; these codes are illustrated in table 2.

Notes on Construction

The prototype was built on a small breadboard with a dual 22-pin edge connector, available at Radio Shack. After cutting a few notches on this connector, it will fit the edge connector of the printer perfectly. Since all signals are fairly low frequency, parts placement on the board is not critical. I used point-to-point wiring using pre-cut wire-wrapping wire. Use a low wattage

***Listing 2:* Hexadecimal character code conversion table.**

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
8100	FF	FF	FF	D7	DB	3B	93	FF	C7	FF	BB	EF	FF	EF	FF	FB
8110	83	FF	B9	7B	E7	1B	C3	7F	93	9D	FF	FF	EF	D7	FF	BF
8120	83	C1	01	83	01	01	01	83	01	FF	FB	01	01	01	01	83
8130	01	83	01	9B	7F	01	07	01	39	3F	79	01	BF	7D	F7	FD
8140	FF	E3	01	E3	E3	E3	FF	C7	01	FF	FF	01	FF	C1	C1	E3
8150	80	C7	C1	ED	DF	C3	CF	C3	DD	FF	DD	FF	EF	7D	F7	00
8160	FF	FF	1F	01	AB	37	6D	FF	BB	FF	D7	EF	FD	EF	FF	F7
8170	75	BD	75	7D	D7	5D	AD	71	6D	6D	FF	FD	D7	D7	7D	7F
8180	7D	B7	6D	7D	7D	6D	6F	7D	EF	7D	FD	EF	FD	BF	DF	7D
8190	6F	7D	6F	6D	7F	FD	FB	FB	D7	DF	75	01	DF	7D	EF	FD
81A0	BF	DD	EB	DD	DD	D5	EF	BA	DF	ED	FE	F7	7D	DF	DF	DD
81B0	D7	BB	EF	D5	DF	FD	F3	FD	EB	C2	D9	EF	EF	7D	EF	00
81C0	FF	05	FF	D7	01	EF	95	1F	7D	7D	01	83	F3	EF	FD	EF
81D0	6D	01	6D	6D	B7	5D	6D	6F	6D	6D	D7	D3	BB	D7	BB	65
81E0	45	77	6D	7D	7D	6D	6F	7D	EF	01	FD	D7	FD	CF	EF	7D
81F0	6F	75	67	6D	01	FD	FD	E7	EF	E1	6D	7D	EF	7D	DF	FD
8200	5F	DD	DD	DD	DD	D5	81	BA	DF	A1	FE	E7	01	E1	DF	DD
8210	BB	BB	DF	D5	81	FD	FD	FB	F7	FA	D5	93	AB	93	EF	00
8220	FF	FF	1F	01	AB	D9	FB	FF	FF	BB	D7	EF	FF	EF	FF	DF
8230	5D	FD	6D	4D	01	5D	6D	5F	6D	6B	FF	FF	7D	D7	D7	5F
8240	65	B7	6D	7D	7D	6D	6F	75	EF	7D	FD	BB	FD	BF	F7	7D
8250	6F	7B	6B	6D	7F	FD	FB	FB	D7	DF	5D	7D	F7	01	EF	FD
8260	BF	EB	DD	DD	EB	D5	6F	D6	DF	FD	A1	DB	FD	DF	DF	DD
8270	BB	D7	DF	D5	DF	FD	F3	FD	EB	FA	CD	7D	EF	EF	EF	00
8280	FF	FF	FF	D7	B7	B9	F5	FF	FF	C7	BB	EF	FF	EF	FF	BF
8290	83	FF	9D	33	F7	63	73	3F	93	87	FF	FF	FF	D7	EF	BF
82A0	8D	C1	93	BB	83	7D	7F	71	01	FF	03	7D	FD	01	01	83
82B0	9F	85	9D	B3	7F	01	07	01	39	3F	3D	7D	FB	01	F7	FD
82C0	FF	C1	E3	DD	01	E7	FF	81	E1	FF	FF	BD	FF	E1	E1	E3
82D0	C7	80	FF	DB	DF	C1	CF	C3	DD	C1	DD	7D	EF	FF	DF	00
82E0	48	98	48	A0	00	A9	20	99	00	D3	99	00	D2	99	00	D1
82F0	99	00	D0	C8	D0	F1	68	A8	68	60	4B	65	72	79	61	6E

***Listing 3:* Character demonstration program in BASIC.**

```
10  REM  CHARACTER
15  REM  DEMO
20  REM  BY M.J. KERYAN
25 :
30 IS = 53612: REM  CORNER
35 TA = 32992: REM  TABLE-32
40 CU = 54116: REM  CURS LOC
45 B1 = 32:B2 = 127
50  FOR C = IS - 66 TO IS - 58
55  POKE C,B2: POKE C + 32,B1
60  POKE C + 320,B1: POKE C + 352,B2
65  NEXT : POKE IS - 34,B2: POKE IS - 26,B2
70  FOR C = IS - 2 TO IS + 224 STEP 32
75  POKE C,B2: POKE C + 1,B1
80  POKE C + 7,B1: POKE C + 8,B2
85  NEXT : POKE IS + 254,B2: POKE IS + 262,B2
90  FOR CR = 32 TO 127
95  POKE CU,CR
```

(Continued)

soldering iron and sockets for the CMOS IC's. None of the resistor or capacitor values is very critical. All transistors should be high gain, high current types, such as 2N3643, 2N4401, etc. The unused input pins of IC3 should be brought to either 5 volts or ground.

The circuit board, switches, and transformer were mounted in a Radio Shack plastic box (item #270-224). The printer was mounted on top, using rubber stand-offs. The paper holder was made from a piece of aluminum formed into a U-shape. A cut-down toilet tissue holder was mounted on the support. Before connecting the interface to the printer, the interface should be powered up and checked out by bringing all inputs to 5 volts or ground, and monitoring the corresponding outputs. Then connect the printer, turn it on, and check out the motor by switching the line marked PA3 to ground.

Comments on Use

If out of paper, pull the plastic guard up and lock the metal lever up to loosen the platen. Feed the end of a new roll from the back, release the metal lever to tighten the platen against the paper, and close S1. The paper will then advance as long as S1 is closed. After opening S1, flip the plastic guard back into position and the printer will continue normal operation.

The printer should only be turned on after the computer is powered up. Likewise, the printer should be turned off before the computer. Failure to follow this sequence will turn on the motor, due to a low voltage at PA3. The reason for this configuration is that before the PIA is initialized, all outputs will be high.

When printing tables, it is sometime advantageous to change the spacing parameters between lines. This was done in table 2, in which three different configurations were used.

Michael Keryan has a Master of Science degree in Chemical Engineering, and has used computers in school, and for the last eleven years in industrial applications. His hobby has been electronics and, most recently, microcomputers; his interests are equally divided between hardware, software, and systems. To keep the cost of his hobby within reason, he prefers to build everything himself. This article is the result of one such project.

MICRO

Listing 3 *(Continued)*

```
100  GOSUB 125
105  FOR DE = 1 TO 250: NEXT DE
110  NEXT CR
115  END
120 : REM SUBROUTINE
125 : REM  PLOTS CHAR'S
130  FOR J = 0 TO 4
135 JN = J * 96
140 X =  PEEK (CR + TA + JN)
145  FOR N = 7 TO 0 STEP  - 1
150 P = 2 ^ N
155 L = IS + J + 32 * (7 - N)
160  IF (X AND P) > .5 THEN  POKE L,B1: GOTO 170
165  POKE L,B2
170  NEXT N
175  NEXT J
180  RETURN
```

***Table 2:* Character set. The tables in listing 2 define 96 characters. These are the standard ASCII symbol, upper case, and lower case characters, except for a degree symbol (for hexadecimal 60) and a divide symbol (for hexadecimal 7C).**

HEX-ASCII TABLE

Hexadecimal Code:

X	2X	3X	4X	5X	6X	7X
0		0	@	P	°	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	[	k	{
C	,	<	L	\	l	÷
D	—	=	M	]	m	}
E	.	>	N	^	n	~
F	/	?	O	_	o	▒

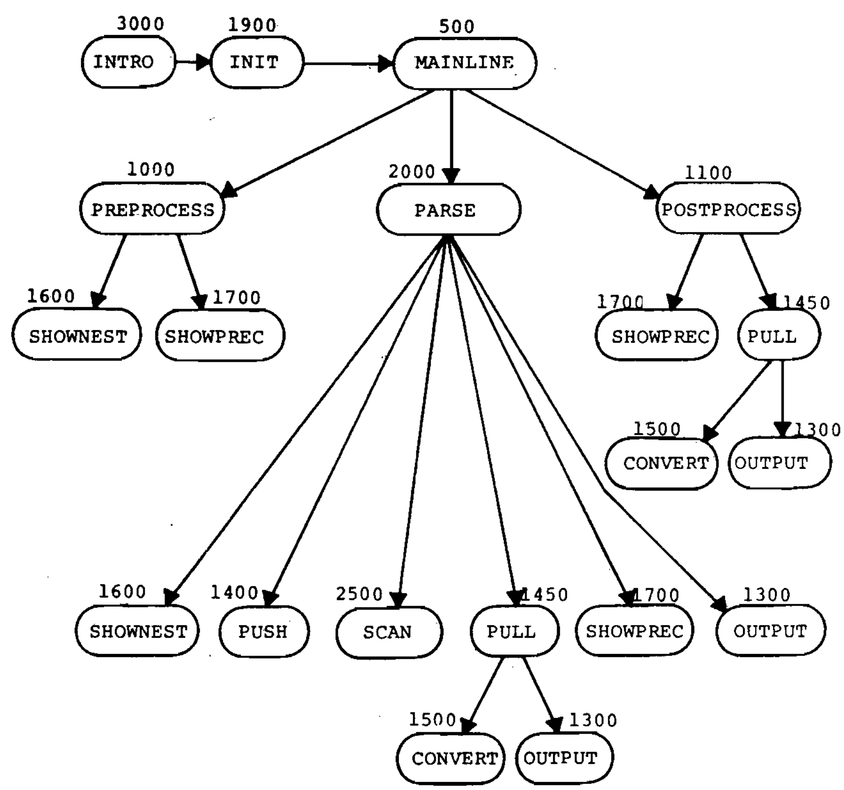
Expressions Revealed,

Part 2

In this, the final part of the series, the author presents and discusses BASIC and Pascal versions of a program demonstrating the translation process.

Richard C. Vile, Jr.
3467 Yellowstone Dr.
Ann Arbor, Michigan 48105

Figure 1: **Subroutine Hierarchy of Listing 2.**



Expression Translation Implemented

Listings 1 and 2 present two demonstration programs, both of which implement the infix to postfix translation algorithm. They allow the user to view the process as it is carried out, by displaying various information used by the algorithm on the Apple II screen. The program in listing 1 is written in Integer BASIC, while that in listing 2 is written in Pascal. We shall conclude the article with a few comparisons between the implementations and an elucidation of the operation of the demonstrations.

The demonstration programs expect a partially parenthesized expression as input. The allowable operators in the expression are as follows:

& ! ' = # < > + - * / ↑

where the logical operators AND, OR, and NOT have been replaced by the single characters &, !, and ', respectively. This makes the operation of the scanner much simpler and removes detail from our discussion that is not strictly relevant to the translation algorithm.

The translation algorithm discussed last month in part 1 is executed directly upon the screen. As each character is scanned, it is highlighted in reverse video. (Note: if your Apple II has been modified to display lower case, this probably will not work.) The output string, which is the RPN translation of the original expression, grows dynamically on a separate line as the scan progresses, and the stack of operators grows and shrinks on yet another line. In addition, other information is displayed on the lower portion of the screen:

```
NESTING LEVEL = = = = = = = >
CURRENT PRECEDENCE = = = >
LAST PRECEDENCE = = = = = >
TOKEN = = = = = = = = = = = >
STACK DEPTH= = = = = = = = >
```

Each piece of information so displayed is updated on the screen whenever it is modified by any portion of the translation algorithm. As the translation proceeds, there are pauses to allow the viewer to absorb the significance to the translation of the changes that have taken place. To cause the translation to continue after one of these pauses, simply press any key on the Apple II keyboard. A more detailed version of the demonstration in which the routines of the translation algorithm "talk" to the user, i.e. print explanations of their operation, is available from the author (see note at end of the article).

Figure 1 shows the calling heirarchy of the routines used in the BASIC implementation of the translation algorithm (see listing 1). It is suggested that the user study the Pascal implementation given in listing 2 and construct a similar diagram. This will give an opportunity to compare the inner details of the two implementations.

Some Comparisons

There are some noteworthy points concerning the style of the two programs presented in listings 1 and 2 which bear directly on the differences between the two languages BASIC and Pascal. The following discussion is not intended to be complete, but rather to prompt the reader into further thoughts and investigations along the same lines.

Length: The Pascal version is longer than the BASIC version, at least in pages of text (I did not count individual characters). There are several reasons for this: Pascal encourages and indeed requires the programmer to provide more information about the program, and Pascal is much easier to read if it is written in a "spread out" fashion. Even though the following code would be "legal":

```
IF TOKEN = OPERAND THEN
RPNOUT(NEXTCHAR) ELSE IF
TOKEN = LPAREN THEN BEGIN
NEST := NEST + 1; GOTOXY
(25,NESTLINE); SCREEN(CLREOL);
WRITE(NEST); END ELSE IF
TOKEN = RPAREN THEN BEGIN
NEST := NEST - 1; GOTOXY
(25,NESTLINE); SCREEN(CLREOL);
WRITE(NEST); END ELSE BEGIN
NOWP := NEST*10 +
PRECEDENCE[TOKEN]; SHOW
PRECEDENCE; POPSTACK(NOWP);
PUSHSTACK(TOKEN,NOWP); END;
```

it is extremely difficult to read and would be considered poor Pascal style. See listing 2 for the "acceptable" version of the same code (in PROCEDURE PARSE). What is the underlying reason for this? In Pascal, statements may continue on for many lines. This example is actually *one* Pascal IF statement. In BASIC this is not the case; statements are limited to a single line. The consequence is that you don't have to be as careful when formatting your BASIC source programs as you do when formatting your Pascal programs.

The practical consequences of the differences in length seem to be:

1. Pascal programs tend to be easier to read, understand and modify, but they are more difficult in some ways to write.
2. BASIC programs, especially shorter ones, tend to be easier to write than the corresponding Pascal programs. They are more difficult to read, understand, and modify, especially as they become longer.

Structure: The Pascal language provides many more structuring facilities than does the BASIC language. This applies not only to the procedural portion of programs in which Pascal provides:

named procedures with parameters
if-then-else statement
while-do statement
repeat-until statement
for statement

but also in the *declarative* portion of programs in which Pascal provides explicit structuring mechanisms to reveal the logical relationships between various pieces of data used. Pascal gives us not only variables and arrays, but also:

sets
records
pointers

as well as the ability to *nest* instances of these facilities, one within the other. This leads to a notational clarity in the *representation* of data, especially data that possesses some inherent structure. In the demonstration programs, the operator stack provides a simple example. In the BASIC version, the stack of composite items of information must be represented using separate arrays which are maintained in "parallel." The value of the top of stack is kept in yet another variable. In the Pascal implementation, the operator stack is considered to be a single entity. The structure of this entity is declared in the *type* section of the program:

```
TYPE
  .
  .
  .
 STACK = RECORD
     TOS: INTEGER;
     OPS: ARRAY[0..40] OF RECORD
          OPR: OPERATOR;
          PREC: INTEGER;
        END;
    END;
```

The stack is incarnated in the *var* section of the program:

```
VAR
  .
  .
  .
  OPSTACK: STACK;
```

The OPSTACK is a *single variable* whose structure is indicated by its type, namely STACK. The various parts of the stack may only be accessed by mentioning the name of the operator stack, OPSTACK first. For example,

```
OPSTACK.TOS
OPSTACK.OPS[I].PREC
OPSTACK.OPS[OPSTACK.TOS].OPR
```

and so on. To the long-time BASIC user, this seems like wasteful nomenclature, but it serves at least two important functions:

1. It documents the use of the data in the program for the future reader of the program. This documentation is directly a part of the code itself and is "forced" on the programmer.
2. It forces the programmer to write in more detail, thus preventing, in many cases, inadvertent modification of variables, which could lead to subtle bugs. This is much more important in larger programs, especially in those in which *many* variables may have identical structure. In such cases, the use of parallel arrays requires the invention of different *names* for the pieces of each individual variable. This proliferation of names can easily tax the memory of the best programer.

Listing 1

```
10 DIM LINE$(250)
11 DIM STACK(25)
12 DIM PRECEDENCE(25)
20 CLREOL=-968:KBD=-16384:CLR=
   -16368:HOME=-936
25 INIT=1900:PREPROCESS=1000
26 POSTPROCESS=1100
27 INTRO=3000
28 SCAN=2500:PARSE=2000
29 ERRLINE=22:WAIT=1200
30 OUTPUT=1300:OLINE=6
31 PUSH=1400:PULL=1450
32 STKLINE=10:NESTLINE=12:NOWPLINE=
   13
33 CONVERT=1500:LASTPLINE=14:TOKENL
   INE=15:TOSLINE=16
34 SHOWNEST=1600:SHOWPREC=1700

400 REM SET UP FOR A RUN
401 REM ================
405 CALL HOME
410 GOSUB INTRO: CALL HOME
415 GOSUB INIT: REM  SET UP SCREEN
500 REM MAINLINE DRIVER
501 REM ===============
505 VTAB 1: TAB 1: POKE 50,63
506 PRINT "INPUT EXPRESSION TO BE PA
    RSED"
507 CALL CLREOL
508 POKE 50,255
509 PRINT "===>  ";
510 INPUT LINE$:L= LEN(LINE$)
512 IF L#0 THEN 515: TEXT : CALL
    HOME: END
515 GOSUB PREPROCESS
520 FOR CI=1 TO L
525 CH$=LINE$(CI,CI)
530 POKE 50,63: VTAB 2: TAB CI+
    6: PRINT CH$;: POKE 50,255
535 IF CH$#" " THEN GOSUB PARSE
540 IF TOKEN#255 THEN 550
542 REM BAD TOKEN FOUND - ABORT
543 REM ======================
545 VTAB ERRLINE: TAB 5: PRINT
    "ILLEGAL INPUT"
546 GOSUB WAIT: GOTO 505
550 REM TOKEN WAS OK
555 REM CHECK NESTING OK
556 REM ================
560 IF NEST>=0 THEN 575
565 VTAB ERRLINE: TAB 1: PRINT
    "TOO MANY RIGHT PARENTHESES"

566 GOSUB WAIT: GOTO 505
575 GOSUB WAIT
577 VTAB 2: TAB CI+6: PRINT CH$
    ;
580 NEXT CI
590 GOSUB POSTPROCESS
599 GOTO 505
1000 REM PREPROCESS THE INPUT
1001 REM
1002 REM INCLUDES INITIALIZATIONS
1003 REM REPEATED BEFORE EACH PARSE
1004 REM =========================
1005 NOWP=-1:LASTP=-1:NEST=0
1006 TOS=0: REM  STACK POINTER
1010 OI=1: REM  OUTPUT INDEX
1015 VTAB OLINE: TAB 5: CALL CLREOL:
      CALL CLREOL
1020 GOSUB SHOWNEST: GOSUB SHOWPREC
1099 RETURN
1100 REM POSTPROCESS THE INPUT
1101 REM ====================
1105 NOWP=-1: GOSUB SHOWPREC
1110 IF NEST=0 THEN 1120
1115 VTAB ERRLINE: TAB 1: PRINT
     "NOT ENOUGH RIGHT PARENTHESES"
1120 IF TOS=0 THEN 1199
1125 GOSUB PULL
1190 GOSUB WAIT
1199 RETURN
1200 REM WAIT ROUTINE
1201 REM ============
1205 POKE CLR,0
1210 POKE 50,63: VTAB 24: TAB 5
1212 PRINT "PRESS ANY KEY TO CONTINUE
     ";
1213 POKE 50,255
1215 IF PEEK (KBD)<128 THEN 1215

1220 POKE CLR,0
1225 VTAB ERRLINE: TAB 1: CALL CLREOL

1226 VTAB 24: CALL CLREOL
1249 RETURN
1300 REM DISPLAY OUTPUT TOKEN AT
1301 REM APPROPRIATE POSITION ON
1302 REM THE SCREEN.
1303 REM ======================
1305 VTAB OLINE: TAB OI+6: PRINT
     CH$;
1310 OI=OI+1
1349 RETURN
1400 REM PUSH OPERATOR TOKEN ON THE
1401 REM STACK.  DISPLAY THIS ON
1402 REM THE SCREEN.
1403 REM =========================
1405 TOS=TOS+1
1410 STACK(TOS)= ASC(CH$)
1415 VTAB STKLINE: TAB TOS+4: PRINT
     CH$;
1420 PRECEDENCE(TOS)=NOWP
1425 VTAB TOSLINE: TAB 25: CALL
     CLREOL: PRINT TOS
1449 RETURN
1450 REM POP OPERATOR TOKEN FROM THE
1451 REM STACK TO THE OUTPUT. THE
1452 REM SCREEN IS UPDATED TO SHOW
1453 REM THIS TRANSFORMATION.
1454 REM ==========================
1455 IF NOWP>=PRECEDENCE(TOS) THEN
     RETURN
1460 OPR=STACK(TOS)
1465 TOS=TOS-1: IF TOS<0 THEN TOKEN=
     255
1470 VTAB STKLINE: TAB TOS+5: PRINT
     " ";
1475 GOSUB CONVERT:CH$=CHR$: GOSUB
     OUTPUT
1477 VTAB TOSLINE: TAB 25: CALL
     CLREOL: PRINT TOS
1480 VTAB LASTPLINE: TAB 25: CALL
     CLREOL: PRINT PRECEDENCE(TOS)

1485 GOTO 1455
1499 RETURN
1500 REM CONVERT NUM TO CHARACTER
1501 REM INTEGER BASIC CHR$ FUNCTION
1502 REM IN USER CONTRIBUTED SOFT-
1503 REM WARE.
1504 REM ==========================
1505 CHR=OPR
1510 CHS=CHR+128*(CHR<128)
1515 LC1= PEEK (224):LC2= PEEK (
     225)-(LC1>243): POKE 79+LC1-
     256*(LC2>127)+(LC2-255*(LC2>
     127))*256,CHS:CHR$="<": RETURN

1600 REM DISPLAY NESTING LEVEL
1601 REM ====================
1605 VTAB NESTLINE: TAB 25: CALL
     CLREOL: PRINT NEST
1649 RETURN
1700 REM DISPLAY CURRENT PRECEDENCE
1701 REM AND TOP OF STACK PRECEDENCE
```

(Continued)

```
1702 REM ===========================
1705 VTAB NOWPLINE: TAB 25: CALL
     CLREOL: PRINT NOWP
1710 VTAB LASTPLINE: TAB 25: CALL
     CLREOL: PRINT PRECEDENCE( TOS )

1749 RETURN
1900 REM ONE TIME INITIALIZATIONS
1901 REM THIS INCLUDES PRINTING
1902 REM THE SCREEN LAYOUT.
1903 REM ========================
1910 PRECEDENCE( 0 )=-2: REM  NEEDED IN
      ORDER TO STOP POSTPROCESSING
1950 VTAB 4: PRINT "****************
     ***********************";
1952 POKE 50,63: PRINT "OUTPUT":
      POKE 50,255
1954 PRINT "===>"
1956 VTAB 8: PRINT "****************
     ***********************";
1958 POKE 50,63: PRINT "STACK": POKE
     50,255
1960 PRINT "===>"
1962 VTAB 12: POKE 50,63: PRINT
     "NESTING LEVEL========>": CALL
     CLREOL
1963 PRINT "CURRENT PRECEDENCE===>"
     : CALL CLREOL
1965 PRINT "LAST PRECEDENCE======>"
     : CALL CLREOL
1966 PRINT "TOKEN================>"
     : CALL CLREOL
1967 PRINT "STACK DEPTH==========>"
     : CALL CLREOL
1969 POKE 50,255
1970 PRINT : PRINT : PRINT "  PRECEDE
     NCE IS CALCULATED BY:"
1972 PRINT : TAB 2: PRINT "PRECEDENCE
     =( NESTING LEVEL*10 )+TOKEN"
1999 RETURN
2000 REM EXECUTE PARSE MACHINE
2001 REM ACTIONS - CONVERT TO
2002 REM REVERSE POLISH NOTATION
2003 REM ======================
2005 GOSUB SCAN: REM  CONVERT CHAR TO
      TOKEN
2007 T$=CH$: REM  SAVE IN CASE OF PUL
     L
2008 VTAB TOKENLINE: TAB 25: CALL
     CLREOL: PRINT TOKEN
2010 REM THE "PARSE MACHINE" TAKES
2011 REM ACTIONS BASED ON THE VALUE
2012 REM OF THE CURRENT TOKEN.
2013 REM =========================
2020 IF TOKEN#-1 THEN 2030
2025 NEST=NEST+1: GOSUB SHOWNEST
2027 RETURN
2030 IF TOKEN#-2 THEN 2040
2035 NEST=NEST-1: GOSUB SHOWNEST
2037 RETURN
2040 IF TOKEN#0 THEN 2050
2045 GOSUB OUTPUT: RETURN
2050 IF TOKEN=255 THEN RETURN
2055 NOWP=NEST*10+TOKEN: GOSUB SHOWPR
     EC
2060 GOSUB PULL
2062 CH$=T$: REM  RESTORE AFTER POSSI
     BLE PULL
2065 GOSUB PUSH
2070 LASTP=NOWP
2099 RETURN
2500 REM DETERMINE NEXT TOKEN
2501 REM CONVERT CH$ TO INTERNAL
2502 REM FORM.  VALUES ARE:
2503 REM
2504 REM  OPERAND- 0
2505 REM  NOT     - 1 (')
2506 REM  AND/OR - 2 (&,!)
2507 REM  RELOP  - 3 (#,=,<,>)
2508 REM  ADDOP  - 4 (+,-)
2509 REM  MULOP  - 5 (*,/)
2510 REM  EXPOP  - 6 (↑)
2511 REM  LPAREN - -1 '('
2512 REM  RPAREN - -2 ')'
2513 REM
2514 REM =======================
2520 IF ( ASC(CH$)< ASC("A")) OR
     ( ASC(CH$)> ASC("Z")) THEN
     2525
2522 TOKEN=0: RETURN
2525 IF ( ASC(CH$)< ASC("0")) OR
     ( ASC(CH$)> ASC("9")) THEN
     2530
2527 TOKEN=0: RETURN
2530 IF CH$#"(" THEN 2540
2535 TOKEN=-1: RETURN
2540 IF CH$#")" THEN 2550
2545 TOKEN=-2: RETURN
2550 IF CH$#"'" THEN 2560
2555 TOKEN=1: RETURN
2560 IF (CH$#"&" AND CH$#"!") THEN
     2570
2565 TOKEN=2: RETURN
2570 IF (CH$#"#" AND CH$#"=" AND
     CH$#"<" AND CH$#">") THEN 2580

2575 TOKEN=3: RETURN
2580 IF (CH$#"+" AND CH$#"-") THEN
     2590
2585 TOKEN=4: RETURN
2590 IF (CH$#"*" AND CH$#"/") THEN
     2600
2595 TOKEN=5: RETURN
2600 IF CH$#"↑" THEN 2610
2605 TOKEN=6: RETURN
2610 TOKEN=255: RETURN : REM  ERROR T
     OKEN
3000 REM INTRODUCTION TO PROGRAM
3001 REM =====================
3005 VTAB 1: TAB 1
3009 POKE 50,63
3010 PRINT "  DEMONSTRATION OF EXPRES
     SION PARSING."
3011 POKE 50,255: PRINT
3012 PRINT "THIS PROGRAM CONVERTS INF
     IX NOTATION"
3014 PRINT "EXPRESSIONS TO REVERSE PO
     LISH NOTATION:"
3015 PRINT "ALSO KNOWN AS 'POSTFIX' N
     OTATION."
3018 PRINT
3020 PRINT "  THE INPUT EXPRESSION IS
      SCANNED FROM"
3022 PRINT "LEFT TO RIGHT.  OPERANDS,
      IN THIS DEMO"
3024 PRINT "REPRESENTED BY SINGLE LET
     TERS OR DIGITS,";
3026 PRINT "ARE OUTPUT WHEN ENCOUNTER
     ED.  OPERATORS"
3028 PRINT "ON THE OTHER HAND ARE STA
     CKED WHEN FIRST";

3030 PRINT "SCANNED.  THE TOP OF THE
     STACK IS SENT"
3032 PRINT "TO THE OUTPUT WHENEVER TH
     E PRECEDENCE"
3034 PRINT "OF THE INCOMING OPERATOR
     IS LESS THAN"
3036 PRINT "THAT OF THE TOP OF THE ST
     ACK."
3038 PRINT
3040 PRINT "  USE THE FOLLOWING SPECI
     AL CHARACTERS"
3042 PRINT "IN PLACE OF THE LOGICAL O
     PERATORS:"
3044 PRINT : TAB 5: PRINT "'AND' - &"

3046 TAB 5: PRINT "'OR'  - !"
3048 TAB 5: PRINT "'NOT' - '"
3990 GOSUB WAIT
3999 RETURN
```

Listing 2

```
PROGRAM POLISH;

  USES APPLESTUFF;

CONST

  OUTLINE       =        5;
  HOME          =       12;
  CLREOL        =       29;
  STACKLINE     =        9;
  NESTLINE      =       12;
  NOWPLINE      =       13;
  LASTPLINE     =       14;
  TOKENLINE     =       15;
  TOSLINE       =       16;
  DEBUGLINE     =       22;
  ERRORLINE     =       21;

TYPE

  TOKENVALUE =  (NOTOKEN,OPERAND,NOTOP,ANDOP,OROP,LSSOP,
                GTROP,EQLOP,NEQOP, PLUSOP,MINUSOP,MULTOP,
                DIVOP,EXPOP,LPAREN,RPAREN);

  OPERATOR    =  NOTOP..EXPOP;
  BYTE        =  0..255;

  STACK = RECORD
            TOS: INTEGER;
            OPS: ARRAY[0..40] OF RECORD
                                   OPR: OPERATOR;
                                   PREC:INTEGER;
                                 END;
          END;

VAR

  TOKEN:        TOKENVALUE;
  EXPRESSION:   STRING[40];
  RPN:          STRING[40];
  SCANPTR:      INTEGER;
  NEXTCHAR:     CHAR;
  OPSTACK:      STACK;
  PRECEDENCE:   ARRAY[OPERATOR] OF INTEGER;
  OPRCHAR:      ARRAY[OPERATOR] OF CHAR;
  NOWP:         INTEGER;
  LASTP:        INTEGER;
  OI:           INTEGER;
  NEST:         INTEGER;
  DONE:         BOOLEAN;


  PROCEDURE POPSTACK(P: INTEGER); FORWARD;
  PROCEDURE INVERSE;     EXTERNAL;
  PROCEDURE NORMAL;      EXTERNAL;
  PROCEDURE FLASH;       EXTERNAL;

  (***********************************)
  (*      W    A    I    T          *)
  (***********************************)

  PROCEDURE WAIT;
  VAR CH:CHAR;
  BEGIN
    IF KEYPRESS
    THEN
      READ(CH);
    (*ENDIF*)

    REPEAT
    UNTIL KEYPRESS;
    READ(CH);

  END (*WAIT*);

  (***********************************)
  (*    S    C    R    E    E    N    *)
  (***********************************)

  PROCEDURE SCREEN(CONTROL: BYTE);
  BEGIN
    WRITE(CHR(CONTROL));
  END;

  (***********************************)
  (*      E    N    T    E    R       *)
  (***********************************)

  PROCEDURE ENTER(N:STRING);
  BEGIN
    GOTOXY(0,DEBUGLINE);
    WRITE('ENTERING ');
    WRITE(N);
    WAIT;
  END;

  (***********************************)
  (*         E    X    I    T        *)
  (***********************************)

  PROCEDURE EXIT(N:STRING);
  BEGIN
    GOTOXY(0,DEBUGLINE);
    WRITE('LEAVING ');
    WRITE(N);
    WAIT;
  END;
  (***********************************)
  (*   S    T    A    R    L    I    N    E  *)
  (***********************************)

  PROCEDURE STARLINE;
  VAR I:INTEGER;

  BEGIN
    FOR I:=1 TO 40 DO WRITE('*');
    WRITELN;
  END (*STARLINE*);

  (***********************************)
  (*    S    H    O    W    N    E    S    T
  (***********************************)

  PROCEDURE SHOWNEST;
  BEGIN
    GOTOXY(25,NESTLINE);
    SCREEN(CLREOL);
    WRITE(NEST);

  END(*SHOWNEST*);

  (***********************************)
  (*     S    H    O    W    P    R    E    C     *)
  (***********************************)

  PROCEDURE SHOWPRECEDENCE;
  BEGIN
    GOTOXY(25,NOWPLINE);
    SCREEN(CLREOL);
    WRITE(NOWP);
    GOTOXY(25,LASTPLINE);
    SCREEN(CLREOL);
    WRITE(OPSTACK.OPS[OPSTACK.TOS].PREC);
  END (*SHOWPRECEDENCE*);


  (***********************************)
  (*     P    R    E    C    V    A    L    S      *)
  (***********************************)

  PROCEDURE PRECVALS;

    (*  INITIALIZE PRECEDENCE ARRAY *)

  BEGIN

    PRECEDENCE[NOTOP]   := 1;
    PRECEDENCE[ANDOP]   := 2;
    PRECEDENCE[OROP]    := 2;
    PRECEDENCE[LSSOP]   := 3;
    PRECEDENCE[GTROP]   := 3;
    PRECEDENCE[EQLOP]   := 3;
    PRECEDENCE[NEQOP]   := 3;
    PRECEDENCE[PLUSOP]  := 4;
    PRECEDENCE[MINUSOP]:= 4;
    PRECEDENCE[MULTOP]  := 5;
    PRECEDENCE[DIVOP]   := 5;
    PRECEDENCE[EXPOP]   := 6;

  END (*PRECVALS*);

  (***********************************)
  (*      O    P    R    V    A    L    S       *)
  (***********************************)

  PROCEDURE OPRVALS;

    (* INITIALIZE STRINGS TO PRINT *)
    (* OPERATORS WITH.             *)

  BEGIN

    OPRCHAR[NOTOP]    := '''';
    OPRCHAR[ANDOP]    := '&';
    OPRCHAR[OROP]     := '!';
    OPRCHAR[LSSOP]    := '<';
    OPRCHAR[GTROP]    := '>';
    OPRCHAR[EQLOP]    := '=';
    OPRCHAR[NEQOP]    := '#';
    OPRCHAR[PLUSOP]   := '+';
    OPRCHAR[MINUSOP]  := '-';
```

(Continued)

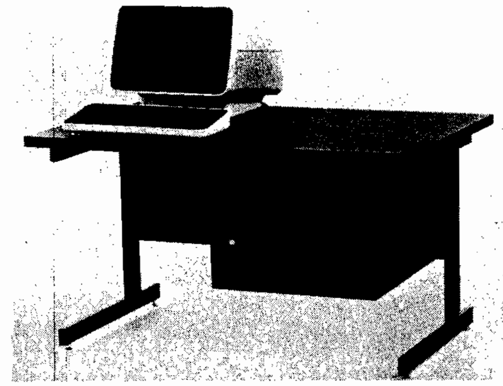
```
    OPRCHAR[MULTOP]  := '*';
    OPRCHAR[DIVOP]   := '/';
    OPRCHAR[EXPOP]   := '↑';

  END (*OPRVALS*);

  (************************************)
  (*   S  H  O  W  T  O  K  E  N     *)
  (************************************)

  PROCEDURE SHOWTOKEN( T: TOKENVALUE);
  BEGIN
    GOTOXY( 25,TOKENLINE );
    SCREEN( CLREOL );

    CASE T OF

      NOTOKEN: BEGIN END;
      OPERAND: WRITE( 'OPERAND' );
      NOTOP:   WRITE( 'NOTOP' );
      ANDOP:   WRITE( 'ANDOP' );
      OROP:    WRITE( 'OROP' );
      LSSOP:   WRITE( 'LSSOP' );
      GTROP:   WRITE( 'GTROP' );
      EQLOP:   WRITE( 'EQLOP' );
      NEQOP:   WRITE( 'NEQOP' );
      PLUSOP:  WRITE( 'PLUSOP' );
      MINUSOP: WRITE( 'MINUSOP' );
      MULTOP:  WRITE( 'MULTOP' );
      DIVOP:   WRITE( 'DIVOP' );
      EXPOP:   WRITE( 'EXPOP' );
      RPAREN:  WRITE( 'RPAREN' );
      LPAREN:  WRITE( 'LPAREN' );

    END;

  END (*SHOWTOKEN*);

  (************************************)
  (*     R   P   N   O   U   T       *)
  (************************************)

  PROCEDURE RPNOUT( C: CHAR );
  BEGIN
    (* ENTER('RPNOUT'); *)
    GOTOXY( OI,OUTLINE );
    WRITE( C );
    OI := OI + 1;
    (* EXIT('RPNOUT'); *)
  END (*RPNOUT*);

  (************************************)
  (*I  N  T  R  O  D  U  C  T  I  O  N*)
  (************************************)

  PROCEDURE INTRODUCTION;
  BEGIN

    SCREEN( HOME );
    INVERSE;
    WRITELN;
    WRITELN( '   DEMONSTRATION OF EXPRESSION PARSING.' );
    NORMAL;
    WRITELN;
    WRITELN( 'THIS PROGRAM CONVERTS INFIX NOTATION' );
    WRITELN( 'EXPRESSIONS TO REVERSE POLISH NOTATION:' );
    WRITELN( 'ALSO KNOWN AS ''POSTFIX'' NOTATION' );
    WRITELN;
    WRITELN( '   THE INPUT EXPRESSION IS SCANNED FROM' );
    WRITELN( 'LEFT TO RIGHT. OPERANDS, IN THIS DEMO' );
    WRITELN( 'REPRESENTED BY SINGLE LETTERS OR DIGITS' );
    WRITELN( 'ARE OUTPUT WHEN ENCOUNTERED. OPERATORS' );
    WRITELN( 'ON THE OTHER HAND ARE STACKED WHEN FIRST' );
    WRITELN( 'SCANNED.  THE TOP OF THE STACK IS SENT' );
    WRITELN( 'TO THE OUTPUT WHENEVER THE PRECEDENCE' );
    WRITELN( 'OF THE INCOMING OPERATOR IS LESS THAN' );
    WRITELN( 'THAT OF THE TOP OF THE STACK.' );
    WRITELN;
    WRITELN( '   USE THE FOLLOWING SPECIAL CHARACTERS' );
    WRITELN( 'IN PLACE OF THE LOGICAL OPERATORS:' );
    WRITELN;
    WRITELN( '      ''AND''  - &' );
    WRITELN( '      ''OR''   - !' );
    WRITELN( '      ''NOT''  - ''' );
    WAIT;
    SCREEN( HOME );

  END (*PROCEDURE INTRODUCTION*);

  (************************************)
  (*   I  N  I  T  I  A  L  I  Z  E  *)
  (************************************)

  PROCEDURE INITIALIZE;
  BEGIN
    GOTOXY( 0,4 );
    STARLINE;
    INVERSE;
    WRITE( 'OUTPUT' );
    NORMAL;
    WRITELN( '===>' );
    GOTOXY( 0,8 );
    STARLINE;
    INVERSE;
    WRITE( 'STACK' );
    NORMAL;
    WRITELN( '===>' );
    GOTOXY( 0,NESTLINE );
    INVERSE;

    WRITE( 'NESTING LEVEL========>' );
    SCREEN( CLREOL );
    GOTOXY( 0,NOWPLINE );
    WRITE( 'CURRENT PRECEDENCE===>' );
    SCREEN( CLREOL );
    GOTOXY( 0,LASTPLINE );
    WRITE( 'LAST PRECEDENCE======>' );
    SCREEN( CLREOL );
    GOTOXY( 0,TOKENLINE );
    WRITE( 'TOKEN================>' );
    SCREEN( CLREOL );
    GOTOXY( 0,TOSLINE );
    WRITE( 'STACK DEPTH==========>' );
    SCREEN( CLREOL );
    NORMAL;

  END (*PROCEDURE INITIALIZE*);

  (************************************)
  (*   P  R  E  P  R  O  C  E  S  S  *)
  (************************************)

  PROCEDURE PREPROCESS;
  BEGIN
    NOWP := -1;
    LASTP := -1;
    NEST  := 0;
    OPSTACK.TOS := 0;     (*TOP OF STACK*)
    OPSTACK.OPS[OPSTACK.TOS].PREC := -1;
    OI     := 11;         (*OUTPUT INDEX*)
    GOTOXY( OI,OUTLINE );
    SCREEN( CLREOL );
    WRITELN;
    SCREEN( CLREOL );
    SHOWNEST;

    SHOWPRECEDENCE;
  END (*PREPROCESS*);

  (************************************)
  (*  P  O  S  T  P  R  O  C  E  S  S*)
  (************************************)

  PROCEDURE POSTPROCESS;
  BEGIN
    NOWP := -1;
    SHOWPRECEDENCE;
    IF NEST > 0
    THEN
    BEGIN
      GOTOXY( 1,ERRORLINE );
      SCREEN( CLREOL );
      FLASH;
      WRITE( 'TOO FEW RIGHT PARENTHESES' );
      NORMAL;
    END;

    IF OPSTACK.TOS > 0
    THEN
      POPSTACK( NOWP );
    (*ENDIF*)

    WAIT;

  END (*POSTPROCESS*);

  (************************************)
  (*       S     E     T     U     P *)
  (************************************)

  PROCEDURE SETUP;
  BEGIN
    PRECVALS;
    OPRVALS;
    INTRODUCTION;
  END (*SETUP*);
```

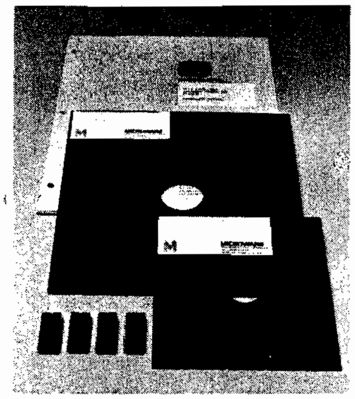
(Continued)

```
(*************************************)
(*      S      C      A      N      *)
(*************************************)

FUNCTION SCAN : TOKENVALUE;
VAR
  RETTOK: TOKENVALUE;
BEGIN
  RETTOK := NOTOKEN;
  WHILE RETTOK = NOTOKEN DO
  BEGIN
    NEXTCHAR := EXPRESSION[SCANPTR];
    SCANPTR  := SCANPTR + 1;

    CASE NEXTCHAR OF

    'A','B','C','D','E','F','G','H','I','J','K','L','M',
    'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
    '0','1','2','3','4','5','6','7','8','9':

      RETTOK := OPERAND;

    '''':       RETTOK := NOTOP;
    '&':        RETTOK := ANDOP;
    '!':        RETTOK := OROP;
    '<':        RETTOK := LSSOP;
    '>':        RETTOK := GTROP;
    '=':        RETTOK := EQLOP;
    '#':        RETTOK := NEQOP;
    '+':        RETTOK := PLUSOP;
    '-':        RETTOK := MINUSOP;
    '*':        RETTOK := MULTOP;
    '/':        RETTOK := DIVOP;
    '^':        RETTOK := EXPOP;
    '(':        RETTOK := LPAREN;
    ')':        RETTOK := RPAREN;

    END (*CASE*);

    IF RETTOK = NOTOKEN
    THEN
    BEGIN
      GOTOXY(0,23);
      WRITE('ILLEGAL CHARACTER IN EXPRESSION');
    END;

  END (*WHILE RETTOK = NOTOKEN*);
  SCAN := RETTOK;
  SHOWTOKEN(RETTOK);

END (* FUNCTION SCAN *);

(*************************************)
(* P    O    P    S    T    A    C    K   *)
(*************************************)

PROCEDURE POPSTACK;
VAR
  PC: CHAR;
BEGIN

  WHILE P < OPSTACK.OPS[OPSTACK.TOS].PREC DO
  BEGIN
    PC := OPRCHAR[OPSTACK.OPS[OPSTACK.TOS].OPR];
    RPNOUT(PC);
    GOTOXY(9+OPSTACK.TOS,STACKLINE);
    WRITE(' ');
    OPSTACK.TOS := OPSTACK.TOS - 1;
    GOTOXY(25,TOSLINE);
    WRITE(OPSTACK.TOS);
  END;

END (*POPSTACK*);

(*************************************)
(*    P  U  S  H  S  T  A  C  K    *)
(*************************************)

PROCEDURE PUSHSTACK(O: OPERATOR; P: INTEGER);
BEGIN
  WITH OPSTACK DO
  BEGIN
    TOS := TOS + 1;
    OPS[TOS].OPR := O;
    OPS[TOS].PREC:= P;
  END (*WITH*);
  GOTOXY(9+OPSTACK.TOS,STACKLINE);
  WRITE(OPRCHAR[O]);
  GOTOXY(25,TOSLINE);
  WRITE(OPSTACK.TOS);

END (*PUSHSTACK*);
```

```
(*************************************)
(*      P     A     R     S     E      *)
(*************************************)

PROCEDURE PARSE;

BEGIN
  SCANPTR := 1;
  WHILE SCANPTR <= LENGTH(EXPRESSION) DO
  BEGIN

   GOTOXY(3+SCANPTR,2);
   INVERSE;
   WRITE(EXPRESSION[SCANPTR]);
   NORMAL;
   TOKEN := SCAN;

   IF TOKEN = OPERAND
   THEN
     RPNOUT(NEXTCHAR)
   ELSE
     IF TOKEN = LPAREN
     THEN
     BEGIN
       NEST := NEST + 1;
       GOTOXY(25,NESTLINE);
       SCREEN(CLREOL);
       WRITE(NEST);
     END
     ELSE
       IF TOKEN = RPAREN
       THEN
       BEGIN
         NEST := NEST - 1;
         GOTOXY(25,NESTLINE);
         SCREEN(CLREOL);
         WRITE(NEST);
       END
       ELSE
       BEGIN

         NOWP := NEST*10 + PRECEDENCE[TOKEN];
         SHOWPRECEDENCE;
         POPSTACK(NOWP);
         PUSHSTACK(TOKEN,NOWP);

       END (*IF*);
     (*ENDIF*)
   (*ENDIF*)
   WAIT;
   GOTOXY(2+SCANPTR,2);
   NORMAL;
   WRITE(EXPRESSION[SCANPTR-1]);
  END (*WHILE*);

END (*PROCEDURE PARSE*);

BEGIN

  SETUP;
  DONE := FALSE;
  REPEAT

    INITIALIZE;
    GOTOXY(0,1);
    INVERSE;
    WRITELN('INPUT EXPRESSION TO BE PARSED');
    NORMAL;
    SCREEN(CLREOL);
    WRITE('===>');
    READLN(EXPRESSION);

    IF LENGTH(EXPRESSION) = 0
    THEN
      DONE := TRUE;
    (*ENDIF*)

    PREPROCESS;
    PARSE;
    POSTPROCESS;

  UNTIL DONE;
  SCREEN(HOME);

END.
```

A TEAM OF 6809 SUPERSTARS:
Smoke Signal's Chieftain™ Computer, and Software by Microware





HERE'S THE *TOTAL* 6809-BASED SYSTEM FOR THOSE WHO DEMAND UNSURPASSED POWER, FLEXIBILITY AND RELIABILITY

After years of worldwide use in diverse and challenging applications, the outstanding performers in 6809 computer operations are SMOKE SIGNAL and MICROWARE. These leading companies are recognized as the undisputed choices **when there is no room for compromises**.

WHY SMOKE SIGNAL AND MICROWARE LEAD THE 6809 FIELD

Smoke Signal began pioneering research and development on 6800/6809-based computer systems back in 1977. Microware worked three years to perfect OS-9 and BASIC09.

Both companies have evolved outstanding 6809-based products from early engineering research, **and both pay almost fanatical attention to detail**. For example . . .

SMOKE SIGNAL'S 6809-based Chieftain™ computer series has **proven** its superiority in hundreds of demanding tasks. From gold-plated connectors to highest-quality materials throughout, each Chieftain™ is built to deliver absolute dependability from day one, and **stay** that way through years of service.

• ENDURANCE-CERTIFIED . . . an exclusive Smoke Signal quality-control measure that positively verifies a component is free of defects, and meets or exceeds all specifications.

Every Chieftain™ is meticulously ENDURANCE-CERTIFIED at 2.2 MHz. That's SMOKE SIGNAL's endorsement of product perfection.

MICROWARE's state-of-the-art OS-9 UNIX*-like operating system and the BASIC09 language have been developed in close coordination with computer manufacturers to maximize optimum system performance. The finest possible support and documentation further ensure satisfaction. Microware software performance is best summed up in this remark by a 25-year computer veteran:

"BASIC09 IS THE FINEST HIGH-LEVEL LANGUAGE I'VE EVER SEEN IN THE INDUSTRY!"

Thousands of engineers and programmers use MICROWARE software products as their standard time-saving tool . . . to execute process-control applications . . . and for other vital functions. COBOL and PASCAL are also available under the OS-9 operating system.

HOW THIS REMARKABLE TEAM OF COMPUTER SUPERSTARS CAN SERVE *YOU*

SMOKE SIGNAL's Chieftain™ computer provides an array of configurations ranging from 5¼-inch drives for single-user applications to multi-user, multi-tasking capabilities. Winchester hard-disk drive systems are also available.

In other words, **breathtaking power** with as little as 48k memory; Microware's OS-9 Level Two can access up to one full megabyte that your Chieftain™ can address!

One more sampling of the awesome processing potential at your fingertips with the Smoke Signal Chieftain™ computer:

MICROWARE'S Stylograph screen-oriented word processing package instantly makes Chieftain™ an easy-to-use document preparation system with comprehensive editing commands.

THERE'S MUCH, MUCH MORE! Call or write SMOKE SIGNAL for details on Chieftain™ computers and MICROWARE software.

SMOKE SIGNAL Dealer opportunities are still available . . . please request information.

*UNIX is a trademark of Bell Telephone Laboratories.

SMOKE SIGNAL BROADCASTING | MICROWARE

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

☐ Send information about Chieftain™ computers and Microware software.

☐ Provide information about Smoke Signal's Dealer program.

Name ______________________

Address ______________________

City ____________ State ______ Zip ________

Telephone ( ) ______________________

MICRO New Publications

Mike Rowe
New Publications
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

General 6809

6809 Assembly Language Programming by Lance A. Leventhal. OSBORNE/McGraw-Hill (630 Bancroft Way, Berkeley, California 94710), 1981, 568 pages, diagrams, charts, listings, 6½ × 9¼ inches, paperbound.
ISBN: 0-931988-35-7 $16.99

This is a comprehensive book on 6809 assembly language programming. It is a text both for those who have never before programmed in assembly language and also for experienced programmers, as well as a valuable reference to the 6809 instruction set and programming techniques.

CONTENTS: Section I—Fundamental Concepts: *Introduction Assembly Language Programming*—A Computer Program; High-Level Languages. *Assemblers*—Features of Assemblers; Types of Assemblers; Errors; Loaders. *6809 Machine Structure and Assembly Language*—6809 Registers and Flags; 6809 Addressing Modes; Modes Which Do Not Specify Memory Locations; Memory Addressing Modes; Indexed Memory Addressing Modes; Program Relative Addressing for Branches; 6809 Instruction Set; 6800/6809 Compatibility; 6801/6809 Compatibility; 6502/6809 Compatibility; Motorola 6809 Assembler Conventions. Section II—Introductory Problems: *Beginning Programs*—Program Examples; Problems. *Simple Program Loops*—Program Examples; Problems. *Character-Coded Data*—Handling Data in ASCII; Program Examples; Problems. *Code Conversion*—Program Examples; Problems. *Arithmetic Problems*—Program Examples; Problems. *Tables and Lists*—Program Examples; Problems. Section III—Advanced Topics: *Subroutines*—Program Examples; Position-Independent Code; Nested Subroutines; Problems. *Parameter Passing Techniques*—The PSH and PUL Instructions; General Parameter Passing Techniques; Types of Parameters. *Input/Output Considerations*—I/O Device Categories; Time Intervals; Logical and Physical Devices; Standard Interfaces; 6809 Input/Output Chips. *Using the 6820 Peripheral Interface Adapter (PIA)*—Initializing a PIA; Using the PIA to Transfer Data; Program Examples; More Complex I/O Devices; Problems. *Using the 6850 Asynchronous Communications Interface Adapter (ACIA)*—Program Examples. *Interrupts*—Characteristics of Interrupt Systems; 6809 Interrupt System; 6820 PIA Interrupts; 6850 ACIA Interrupts; 6809 Polling Interrupt Systems; 6809 Vectored Interrupt Systems; Communications Between Main Program and Service Routines; Enabling and Disabling Interrupts; Changing Values in the Stack; Interrupt Overhead; Program Examples; More General Service Routines; Problems. Section IV—Software Development: *Problem Definition*—Inputs; Outputs; Processing Section; Error Handling; Human Factors/Operator Interaction; Examples; Review. *Program Design*—Basic Principles; Flowcharting; Modular Programming; Structured Programming; Top-Down Design; Designing Data Structures; Review of Problem Definition and Program Design. *Documentation*—Self-Documenting Programs; Comments; Flowcharts as Documentation; Structured Programs as Documentation; Memory Maps; Parameter and Definition Lists; Library Routines; Total Documentation. *Debugging*—Simple Debugging Tools; Advanced Debugging Tools; Debugging With Checklists; Looking for Errors; Examples. *Testing*—Selecting Test Data; Examples; Rules for Testing; Conclusions. *Maintenance and Redesign*— Saving Memory; Saving Execution Time; Major Reorganization. Section V—6809 Instruction Set: *The Instruction Set. Appendices*—*A.* Summary of the 6809 Instruction Set; *B.* Summary of 6809 Indexed and Indirect Addressing Modes; *C.* 6809 Instruction Codes, Memory Requirements, and Execution Times; *D.* 6809 Instruction Object Codes in Numerical Order; *E.* 6809 Post Bytes in Numerical Order. *Index.*

Apple

Beneath Apple DOS by Don Worth and Pieter Lechner. Quality Software (6660 Reseda Blvd., Suite 105, Reseda, California 91335), 1981, 174 pages, diagrams, charts, drawings, 5 3/8 × 8 3/8 inches, plastic comb binding with cardstock cover. $19.95

This book is intended to serve as a companion to Apple's DOS Manual, providing additional information for the advanced programmer or the novice Apple user who wants to know more about the structure of diskettes.

CONTENTS: *Introduction; The Evolution of DOS*—DOS 3; DOS 3.1; DOS 3.2; DOS 3.2.1; DOS 3.3. *Diskette Formatting*—Tracks and Sectors; Track Formatting; Data Field Encoding; Sector Interleaving. *Diskette Organization*—Diskette Space Allocation; The VTOC; The Catalog; The Track/Sector List; Text Files; Binary Files; Applesoft and Integer Files; Other File Types; Emergency Repairs. *The Structure of DOS*—Dos Memory Use; The DOS Vectors in Page 3; What Happens During Booting. *Using DOS from Assembly Language*—Direct Use of the Disk Drive; Calling READ/WRITE Track/Sector (RWTS); RWTS IOB by Call Type; Calling the DOS File Manager; File Manager Parameter List by Call Type; The File Manager Work Area; Common Algorithms. *Customizing DOS*—Slave vs. Master Patching; Avoiding Reload of Language Card; Inserting a Program Between DOS and Its Buffers; BRUN or EXEC a HELLO File; Removing the Pause During a Long Catalog. *DOS Program Logic*—Controller Card ROM — Boot 0; First RAM Bootstrap Loader — Boot 1; DOS 3.3 Main Routines; DOS File Manager; READ/WRITE Track/Sector; DOS Zero Page Use. *Appendix A. Example Programs*—Track Dump Program; Disk Update Program; Reformat a Single Track Program; Find Track/Sector Lists Program; Binary to Text File Convert Program. *Appendix B. Disk Protection Schemes. Appendix C. Glossary. Index.*

Apple II User's Guide by Lon Poole, with Martin McNiff and Steven Cook. OSBORNE/McGraw-Hill (630 Bancroft Way, Berkeley, California 94710), 1981, xii, 386 pages, photos, diagrams, tables, listings, 6 × 9¼ inches, paperbound.
ISBN: 0-931988-46-2 $15.00

This guide to the Apple II computer describes both the Apple II and the common peripheral devices including disk drives and printers. It assumes access to an Apple II system already hooked up.

CONTENTS: *Introduction. Presenting the Apple II*—(Keyboard and TV, Inside the Apple II, Memory, Cassette Recorder, Disk Drive, Programs, External Device Controllers, Game Controls, Printer, Graphics Tablet). *How to Operate the Apple II*—Turning the Power On (What You See on the TV, The Prompt Character); The Keyboard; The Cassette Recorder; Using the Disk II (The Disk Operating System, Preparing Blank Diskettes); Loading and Running a Program (Use the Right Version of BASIC, Loading a Program from Cassette, Loading a Program from Disk, Starting a Program Running, Setting TV Color); Miscellaneous Components; Coping with Errors (Error Messages, Correcting Typing Mistakes, Accidental Reset). *Programming in BASIC*—(Starting Up BASIC); Immediate and Programmed Modes (Printing Characters, Printing Calculations, Error Messages, Extra Blank Statements, Statements, Lines and Programs, Programmed Mode, Saving Programs on Cassette); Switching BASICs; Advanced Editing Techniques (Deleting Program Lines, Adding Program Lines, Changing Program Lines, Reexecuting in Immediate Mode);

(Continued on page 91)

MICRO Classified

SYM-1 Expansions

Bare W7AAY 4K RAM board - $8.00 plus SASE. Assembled W7AAY ROM board - $16.00. RAE symbolic disassembler source on cassette - $15.00. Instruction packet to add floppy disks to SYM -$15.00. RAE/KMMM software interface on 5¼" diskette - $15.00. SASE for more information.

John M. Blalock
Blalock & Associates
P.O. Box 39356
Phoenix, AZ 85069

OSI Superboard — Cabinet and Accessories

Pre-cut kit with hardware to build a handsome pine cabinet to house your superboard. Room inside for the power supply and all your extras, $20. RS-232 interface kit, $10. Send for our catalog of software, hardware, kits and accessories.

Dee Products
150 Birchwood Road
Carpentersville, IL 60110

Space Invaders for OSI

Bug free, smooth action, addictive, entirely in machine language. Requires only 2K and loads in 1½ minutes. Includes color, sound, 10 skill levels, and optional joystick control. Difficulty increases as game progresses. For C2-4P, C4P tape systems. $12.00.

Mike Kincaid
6653 Lunde Rd.
Everson, WA 98247

Used Micro Listing Service

Save time, money, mistakes, frustration. List as Buyer/Seller — Apple, PET, OSI, CP/M systems, floppies, printers — all equipment $300 and up. Pay only for results. Get thorough advice and listings over the phone. Call now: 800-327-9191 x 61 or 703-471-0044.

Used Computer Exchange
2329 Hunters Woods Plaza
Reston, VA 22091

New General Ledger by SBCS

This system for the Apple II is based on our standard G/L and is even more flexible and efficient. It includes increased error checking and user prompting. Reports are more detailed and include budgeting. Documentation is rewritten to provide more detail and clarity.

Small Business Computer Systems
4140 Greenwood
Lincoln, NE 68504
402/467-1878

Apple Undeleter

Apple Undeleter restores deleted, unrewritten files of all types. Written in heavily commented Applesoft for easy understanding, copying and modification, Undeleter works with any memory size Apple and is available for DOS 3.2 and 3.3. Price is $12.00 ppd.

D. Cox
787 Gantry Way
Mountain View, CA 94040

Timestack — A Programmable Controller

Expand your KIM-1 into a general-purpose machine. 80-page manual documents Clock/Port/RAM/PROM Expansion Board and controller software. Subroutine library includes user interaction routines, I/O, and clock controls. Complete manual — $15.00. SASE for more information and newsletter.

Hunter Services
P.O. Box 359
Elm Grove, WI 53122

Accounts Receivable by SBCS

This conversion of Osborne's Accounts Receivable software for the Apple II contains the same capabilities, plus many enhancements that increase your Apple's flexibility, speed, and performance. Use alone or integrate with General Ledger. Retail price $249.

Small Business Computer Systems
4140 Greenwood
Lincoln, NE 68504
402/467-1878

AIM 65 Utilities

Improve productivity with quality AIM software. UTIL1 package adds 18 commands to the AIM monitor. Copy and move capability for the editor, memory search and move, virtual I/O and much more. Manual $5 or send SASE for information on this and other software.

Nehalem Bay Software
25730 Beach Dr.
Rockaway, OR 97136

AIM-65 Newsletter * * Target

Target provides hardware and software information useful for AIM-65 and 6502 users. The 1979 and 1980 back issues are available for $12.00 while a continuing subscription costs $6.00. Just write to:

Target
Donald Clem
Route 2
Spenserville, OH 45887

PET/CBM Owners

Real world software at low cost. 2114 RAM adapter and 4K Memory Expansion for "old" 8K PETs. Write for free catalog!

Optimized Data Systems
Dept. M, Box 595
Placentia, CA 92670

C1P Extended Monitor

2K EPROM has 14 cursor control/editing functions, improved keyboard decoding. Machine language save, load, display, modify, move, breakpoint processing and much more. For 24, 32, 64 char/line. $39.95 plus $1.00 shipping. $1.00 for complete information.

Bustek
P.O. Box A
St. Charles, MO 63301

Spanish Hangman

2,000 SPANISH words and sentences taught in a fun way on the Apple. Send for your school's free 30-day evaluation diskette, from:

George Earl
1302 South General McMullen
San Antonio, TX 78237

Business Software by ADS

For the Apple II and Atari/800. Why pay more for a bunch of unrelated programs? Business Plus will handle invoices, statements, credit memos and more, much more! Just $299 complete or $25 for demo disk (credited towards purchase). VISA, Mastercharge accepted.

Advanced Data Systems
7468 Maple Avenue
St. Louis, MO 63143
314/781-9388

Each classified ad costs only $10.00 per insertion, pre-paid with typewritten copy. Please limit these entries to less than 40 words. (Oversized ads will be rejected and returned.) Title line, name and address are not considered in count. Ads received before the 20th of the second month preceding the month of publication will be published in next issue, i.e. May 20th for the July issue. For further information, call (617) 256-5515.

Common Array Names in Applesoft II

Here is a new command for Applesoft II. Its function is to change the names of floating point and integer arrays during program execution.

Steve Cochard
P.O. Box 236
Boyertown, Pennsylvania 19512

One aspect of the BASIC language which differs from other high-level languages, such as FORTRAN, is its lack of ability to handle subroutine calls with parameter lists. This feature of FORTRAN allows the programmer to specify what variables are to be passed to a subroutine. The FORTRAN subroutine name and subroutine call contain lists of the variable names to be used in the subroutine. What this does is to allow the programmer to call standard or "canned" subroutines from the main program without rewriting the subroutine to incorporate the variable names used in the main or calling program.

Any Apple disk user who keeps a subroutine library on disk must have come across this problem with Applesoft. The current solution is to either rewrite the subroutine to incorporate the variable names as used in the main program, or tailor the main program to conform with the standards established by the subroutines in use.

Another, somewhat smaller problem, is interchanging the elements of one array with those of another. This is found in game-type applications frequently. The current solution is to write a FOR-NEXT loop of sufficient depth, to swap each element. Needless to say, as the size or number of dimensions increases, so too does the execution time.

Listing 1: **Trivial program to show name changing and speed of an & command relative to BASIC. Note that the machine language program must be loaded at $300 for proper operation of program listings 1 and 2.**

```
  1 POKE 1013,76:POKE 1014,0: POKE 1015,3
 10 DIM A(15),B(15),C(1000),D(15),E(1000)
 20 FOR I= 1 TO 1000
 30 C(I)=INT(RND(1)*500)
 40 NEXT
 50 HOME: PRINT "INITIALIZED, HIT ANY KEY TO TRANSFER
    ELEMENTS OF ARRAY C TO ARRAY E";:GET A$
100 FOR I= 1 TO 1000
110 TEMP= C(I)
120 C(I)= E(I)
130 E(I)= TEMP
140 NEXT
150 PRINT "ELEMENTS 500 TO 510 OF ARRAY 'E'"
160 FOR I= 500 TO 510
170 PRINT E(I),: NEXT
180 PRINT "TRANSFER COMPLETE. HIT ANY KEY TO TRANSFER
    BACK USING COMMON ARRAY NAME COMMAND";:GET A$
200 &(C,T) :REM  CHANGE 'C' TO 'T'
210 &(E,C) :REM  ARRAY 'E' NOW HAS THE NAME 'C'
220 &(T,E) :REM  ARRAY 'C' HAS THE NAME OF 'E'
230 PRINT "TRANSFER COMPLETE. ELEMENTS RESTORED IN ARRAY
    'C'"
240 FOR I= 500 TO 510
250 PRINT C(I),:NEXT
260 PRINT "DONE"
```

Listing 2: **Another trivial program to show the use of the name change feature in use with subroutines.**

```
  1 POKE 1013,76:POKE 1014,0:POKE 1015,3
 10 DIM A(15),B(15,15),C(10),D(25)
 15 PRINT "THE ARRAY 'C'"
 20 FOR I= 1 TO 10
 30 C(I)= INT(RND(1)*100)
 40 PRINT C(I),
 50 NEXT
 60 &(C,J)
 70 GOSUB 200
 80 &(J,C)
 90 PRINT "THE ARRAY 'C' IS RESTORED"
100 FOR I= 1 TO 10: PRINT C(I),: NEXT: END
200 PRINT "THE ARRAY 'J'"
210 FOR I= 1 TO 10
220 PRINT J(I),:NEXT: RETURN
```

What do these two, seemingly unrelated, problems have in common? Each has the identical, simple solution: change the names of the arrays during program execution.

With the first problem, the solution is to simply change the names of the arrays stored in memory to those used in the subroutine before calling the subroutine. After subroutine execution, the names are changed again to the original. The second problem is solved not by interchanging array elements, but simply by interchanging array names.

The assembly language program presented here solves these problems by changing the names of integer or floating point arrays as stored in the Apple during program execution. The program uses the ampersand (&) as the interface between BASIC and itself. This feature of Applesoft greatly simplifies using utilities such as this. A very brief explanation of the & command may be found in the Applesoft II manual, and is included here for the sake of continuity.

> This symbol, when executed as an instruction, causes an uncon ditional jump to location $3F5.

Since this is the case, all that needs to be done is to place a JMP instruction in this location to the start of the machine language routine to be used. For this utility, which is assembled at location $300, the user would, from the monitor, enter the following to set the & "hook":

```
*3F5:4C 00 03
```

This, of course, may also be done from the BASIC program by the appropriate use of POKEs. In this example the following program line would need to be executed prior to utilizing the & command:

```
100 POKE 1013,76 : POKE 1014,0
: POKE 1015,3
```

Or in general form:

```
LINE# POKE 1013,76;POKE 1014,
      (ADDRESS MOD 256) : POKE
      1015, (ADDRESS/256) : REM
      ALL NUMBERS = INTEGERS
```

Once this is done the hook remains set until changed by either the program or user, or the computer is powered down.

Listing 3

```
                *---------------------------------
                *                                *
                *         COMMON ARRAY           *
                *           NAMES IN             *
                *         APPLESOFT II           *
                *              BY                *
                *          S. COCHARD            *
                *           (C) 1980             *
                *                                *
                * (S-C ASSMB II <4.0> FORMAT)    *
                *                                *
                *---------------------------------
                *
                *  NOTE: ONLY GLOBAL LABELS HAVE BEEN USED
                *        FOR COMPATABILITY WITH OTHER ASSEMBLERS
                *
                *
                              .OR $300
006B-           PTR1   .EQ $6B       START OF ARRAY SPACE
006D-           PTR2   .EQ $6D       END OF ARRAY STORAGE
0071-           TEMP   .EQ $71       TEMP STORAGE
0073-           MASK   .EQ $73
00B1-           CHRGT  .EQ $B1       APPLESOFT CHRGET ROUTINE
0210-           NAME   .EQ $210      TEMP STORAGE
0216-           NAM2   .EQ $216      TEMP STORAGE
0220-           ZPSV   .EQ $220      TEMP STORAGE
DEC9-           SNTX   .EQ $DEC9     SYNTAX ERROR
                *
                * START OF PROGRAM
0300- 48        START  PHA           SAVE FIRST CHARACTER
0301- A2 0A            LDX #10       SAVE SOME ZERO PAGE
0303- B5 6B     STA1   LDA PTR1,X
0305- 9D 20 02         STA ZPSV,X
0308- CA               DEX
0309- 10 F8            BPL STA1
030B- A9 00            LDA #00       CLEAR MASK
030D- 85 73            STA MASK
030F- A2 0C            LDX #$0C
0311- 9D 10 02  LOOP   STA NAME,X    CLEAR NAME
0314- CA               DEX
0315- 10 FA            BPL LOOP
0317- 68               PLA           GET FIRST CHAR BACK
0318- C9 28            CMP #'(       SEE IF IT'S A '(
031A- F0 02            BEQ CON7      YES! CONTINUE


031C- D0 1A            BNE SYER      NO! SYNTAX ERROR
031E- 20 B1 00  CON7   JSR CHRGT     CONTINUE WITH CHAR'S
0321- 8D 10 02         STA NAME      AND SAVE IT.
0324- E8               INX
0325- E8        LOO1   INX           GET SOME MORE TEXT
0326- E0 06            CPX #06       LEN OF NAME GREATER
0328- D0 02            BNE CON3      THAN 6 CHARACTERS?
032A- F0 0C            BEQ SYER      YES! THEN ERROR!
032C- 20 B1 00  CON3   JSR CHRGT     CONTUNUE WITH CHAR'S
032F- C9 2C            CMP #',       END OF ARRAY NAME?
0331- F0 08            BEQ CONT      YES! NEXT NAME
0333- 9D 10 02         STA NAME,X    NO! STORE IT.
0336- D0 ED            BNE LOO1      JUMP BACK FOR MORE.
0338- 4C C9 DE  SYER   JMP SNTX      JUMP TO APPLESOFT SYNTAX ERR
033B- CA        CONT   DEX           IS ARRAY AN INT ARRAY?
033C- BD 10 02         LDA NAME,X
033F- C9 25            CMP #'%
0341- D0 09            BNE CON1      NO, A FP ARRAY
0343- A9 80            LDA #$80      YES, SET MASK FOR NEG
0345- 85 73            STA MASK      ASCII.
0347- A9 00            LDA #00       NEXT, CLEAR % CHAR IN NAME
0349- 9D 10 02         STA NAME,X
034C- A2 00     CON1   LDX #00       GET SECOND NAME.
```

Listing 3 *(Continued)*

```
034E- 20 B1 00  LOO2    JSR CHRGT
0351- C9 25             CMP #'%       IS IT AN INT.ARRAY?
0353- D0 02             BNE CON8      NO, A FP ARRAY
0355- A9 00             LDA #00       YES, SET CHAR=0
0357- C9 29     CON8    CMP #')       END OF NAME?
0359- F0 0A             BEQ CON2      YES! CONTINUE
035B- 9D 16 02          STA NAM2,X    NO! STORE NAME # 2
035E- E8                INX
035F- E0 06             CPX #06       LEN GREATER THAN 6?
0361- D0 EB             BNE LOO2      NO, CONTINUE!
0363- F0 D3             BEQ SYER      YES! ERROR!
0365- A2 0C     CON2    LDX #$0C      MASK NAMES.
0367- A5 73     CON4    LDA MASK
0369- 5D 10 02          EOR NAME,X
036C- 9D 10 02          STA NAME,X
036F- CA                DEX
0370- 10 F5             BPL CON4
                *
                * LOCATE WHERE ARRAY IS STORED
                *
0372- A0 00     LOO3    LDY #00       LOOK AT FIRST NAME IN MEM.
0374- B1 6B             LDA (PTR1),Y
0376- CD 10 02          CMP NAME      IS IT = TO NAME
0379- D0 0A             BNE CON5      NO, LOOK SOME MORE
037B- C8                INY           NEXT CHAR IN NAME.
037C- B1 6B             LDA (PTR1),Y
037E- CD 11 02          CMP NAME+1    IS IT = NAME+


0381- D0 02             BNE CON5      NO, LOOK SOME MORE.
0383- F0 2A             BEQ FIND      FOUND ARRAY! NOW CHANGE IT
0385- A0 02     CON5    LDY #02       GET OFFSET TO NEXT ARRAY.
0387- B1 6B             LDA (PTR1),Y
0389- 85 71             STA TEMP      SAVE HI BYTE.
038B- C8                INY
038C- B1 6B             LDA (PTR1),Y
038E- 85 72             STA TEMP+1    SAVE LO BYTE
0390- 18                CLC           SET UP TO ADD
0391- A5 6B             LDA PTR1
0393- 65 71             ADC TEMP
0395- 85 6B             STA PTR1
0397- A5 6C             LDA PTR1+1
0399- 65 72             ADC TEMP+1
039B- 85 6C             STA PTR1+1
039D- A5 6E             LDA PTR2+1    WAS THAT THE LAST ARRAY
039F- C5 6C             CMP PTR1+1    IN MEMORY?
03A1- F0 04             BEQ CON6      MAYBE!
03A3- 10 CD             BPL LOO3      NO! NOT THIS TIME!
03A5- 30 13             BMI RTRN      WAY PAST IT. TIME TO END!
03A7- A5 6D     CON6    LDA PTR2      HOW 'BOUT LO BYTE
03A9- C5 6B             CMP PTR1
03AB- F0 0D             BEQ RTRN      YES, THIS IS THE END
03AD- D0 C3             BNE LOO3       NOPE, CONTINUE.
                * SWITCH NAMES IN MEMORY
                *
03AF- AD 17 02  FIND    LDA NAM2+1    FOUND IT. NOW
03B2- 91 6B             STA (PTR1),Y  SWITCH NAMES.
03B4- 88                DEY
03B5- AD 16 02          LDA NAM2
03B8- 91 6B             STA (PTR1),Y
03BA- A2 0A     RTRN    LDX #10       RESTORE ZERO PAGE
03BC- BD 20 02  RTR1    LDA ZPSV,X
03BF- 95 6B             STA PTR1,X
03C1- CA                DEX
03C2- 10 F8             BPL RTR1
03C4- 20 B1 00          JSR CHRGT     GET LAST CHARACTER
03C7- 60                RTS           AND RETURN TO BASIC
```

To use the COMMON ARRAY NAME program the program must first be loaded into memory. Since the program is relocatable, it will operate correctly without changes when residing anywhere in memory. A convenient place is starting at hex $300 (768 decimal). Next set the & hooks to the starting address of the program and it is ready to run.

The command to change an array name is of the following form:

&(AA,BB)
&(CAT%,DOG%)

or in general form:

&(name1(%),name2(%))

The % is optional and depends on the array type (int/fp). The command may be used in immediate execution mode or deferred execution mode (within a program). Program listing 1 and listing 2 show examples of the command in use.

Certain limitations are imposed when using this program. Floating point array names are restricted to a maximum of five characters, integer arrays have a maximum of four. This does not limit the versatility of the program, however, since only the first two characters of any variable name are significant in Applesoft. If a longer array name is in use, just shorten it to four or five characters for use in the & command. Everything will work out OK.

Array types may not be intermixed. That is, a floating point array will not be changed to integer and vice-versa.

Two array names must be present in the & command. If not, the program will assume that the first character after the comma is the second name. If used in this way, it is possible to have an array internally renamed to ") ".

If the first (old) array name in the command does not exist in the variable table, no changes will take place. This condition is not signaled to the user. Therefore, care should be taken to have the array DIMensioned prior to using the name change feature.

The Program

The program, quite simple in operation, consists of three parts. The first section reads the old and new array names from the Applesoft & statement. It then stores these names and checks for the array type, either integer or FP.

The two are differentiated, of course, by the presence or absence of the % sign in the array name. Applesoft, however, knows nothing of % signs. It differentiates the two by how the name is stored in memory. Floating point array names are stored as positive ASCII, integers as negative ASCII. In other words, the high order bit is clear or set, respectively. This is dealt with in the program by examining the last character in the first array's name. If it is a %, then a mask is set equal to $80, which in binary is a one followed by seven zeros. If the array is a floating point, then the mask is set equal to zero. With this done, all that is necessary is to "exclusive or" the names with the mask. This will set or clear the high order bit as required.

The second section of the program locates the array in memory. It first picks up the pointer to the start of array storage from locations $6B and $6C. Then the locations pointed to are examined and compared to the first name in the BASIC statement. If there is a match (if the array has been found), the program branches to the third section. If it is not a match, the offset to the next array is picked up from the variable table and added to the pointer. Now the pointer points to the name of the next array in memory. This process is repeated until either a match is found or the limit of array storage is reached. In this case, the program returns to BASIC but does not signal the user that a change has not taken place. Since this is so, the user should be sure the "old" array has been previously DIM'd in the BASIC program before attempting to change its name.

The third section does the actual work of changing the array name. All that is done, is that the "new" name is stored in place of the "old" one in the variable table.

The program has been designed to be completely portable, in that it will execute anywhere in memory. This has been accomplished by utilizing no absolute JMPs within the program by using forced branches. This results in a program with only relative branches (which are location-independent), and a program which may be loaded anywhere that free memory exists in the Apple.

The first two sections of this program are of great versatility, as the reader may have observed by this point. These routines may be incorporated in many other array-handling utilities to form the basis for programs to do such things as clear an array, equate two arrays, delete an array, etc.

MICRO

MICRO Hardware Catalog

Mike Rowe
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Name: **Pre-cut Cabinet Kit**
System: Ohio Scientific Superboard II
Description: This cabinet, manufactured by DEE Products, comes as a pre-cut kit built large enough to house the Superboard along with the 610 memory expansion board. The cabinet also has room for cooling fan, power supply and, (mounted on the rear panel) switches, connectors, and jacks. Built of pine, this handsome cabinet resembles the C1P cabinet and also incorporates a 10 degree tilt to the keyboard, easing use. When finished, this kit makes a quality protective home for your Superboard. Gluing and finishing required.
Price: $20.00 ppd.
Available: DEE Products
150 Birchwood
Lake Marion, IL 60110

Name: **Co-Ax Switching Matrices**
Description: Family of co-ax switching matrices with high speed, long life, high isolation. Cover DC to 1.26 Hz range, insertion losses as low as 0.2 db, available up to 10 × 10 in a single housing, bidirectional. Available with IEEE-488, RS-232C bus or telephone or remove manual interfaces. For switching video, data, RF.
Price: $2400 for typical 5 × 8 matrix
Available: Mar Lee Switch Co.
9330 N. Central Ave.
Upland, CA 91786

Name: **Bytewriter-1**
Memory: One line buffer capacity
Language: BASIC
Description: 7 × 7 dot matrix printer, friction feed, 80 c.p.s., 60 l.p.m., interfaces Apple, Atari and TRS-80, 80-columns per line and double-wide character set.
Price: $299.00
Available: Microtek, Inc.
9514 Chesapeake Dr.
San Diego, CA 92123

Name: **Model Q3 Printer Mechanism**
Memory: 45-Character Buffer
Description: The Model Q3 Printer is an exceptionally rugged, non-impact, thermal printer which is designed to provide the optimum in quiet operation. The Q3 features high resolution plotting capabilities (dot resolution of 0.17) giving the user flexibility to quickly present meaningful statistic graphs. The standard 80/132 selectability ensures users will have neatly formatted reports. Only 4½ pounds, the Q3 prints a full USASCII 5 × 7 dot matrix 96-character set.
Price: $825.00
Available: Computer Devices, Inc.
25 North Avenue
Burlington, MA 01803

Name: **Kleen-Line Isolator**
System: Model ISO-3
Hardware: Stand-alone A.C. power conditioning
Description: Eliminates interaction between processor, printer and peripherals. Also isolates equipment from power line noise and hash as well as high-voltage spikes and transients.
Price: $94.95
Available: Electronic Specialists
171 South Main St.
Natick, MA 01760

Name: **Pro-Paddle**
System: Apple II
Description: Pro-Paddle is the only heavy-duty paddle available for the Apple II. It features compact sturdy metal construction, long-life switches with large buttons and tactile feedback, high accuracy paddle movement, shielded coaxial cables, and a molded plug. They are constructed of the highest quality materials and workmanship available.
Price: $39.95 includes paddles and I/O cable
Author: Computerworks
Available: Rainbow Computing, Mail Order Dept.
19517 Business Center Drive
Northridge, CA 91324

Name: **Micromodem 100**
System: S-100 Bus Computers
Hardware: Low Speed Modem
Description: Direct connect data communications system for S-100 bus computers. Features 110 and 300 baud, full or half duplex and programmable auto dial and auto answer capabilities.
Price: $379.00 (suggested retail)
Available: Hayes Microcomputer Products, Inc.
5835A Peachtree Corners East
Norcross, GA 30092

The Extended Parser for the Apple II

This extended parser for the Apple II or Apple II Plus allows easy control of functions such as clear screen, delete to end of line, flash, and inverse.

Paul R. Wilson
19 Sunset Place
Bergenfield, New Jersey 07621

Back in the June 1980 MICRO (25:15), Edward H. Carlson wrote a sample extension for the parser of the Ohio Scientific computers. He stated that all Microsoft BASIC languages use this parser. I have checked both the Apple's and PET's and they jive with the parser of the Ohio Scientific, save in minor points.

The following is an excellent parser for the Apple II or Apple II Plus as it contains seven useful functions.

Table 1

BASIC	PARSER
CALL −936 (or HOME)	#S clears entire screen
CALL −958	#E clears screen from cursor to end
CALL −868	#L clears from cursor to end of line
POKE 50,127 (or FLASH)	#F puts output into flash mode
POKE 50,63 (or INVERSE)	#I puts output into inverse mode
POKE 50,255 (or NORMAL)	#N restores output to normal mode
TEXT	#T restores screen to text mode

Text in table 1 does not do a complete job. After use of Hi-Res, a later GR will not function properly. The Hi-Res screen will appear instead of Lo-Res. The T-command performs a C056 or POKE −16298,0 to restore GR's proper function, after the C051 or POKE −16303,0. It resets the scrolling screen to full size, but does not send the cursor to the bottom of the screen like TEXT. I only discovered this after I acquired my Disk Drive, which encourages quick succession of programs in one sitting. In many of them, I inserted POKE −16298,0 to guarantee that a use of Hi-Res in some previous program will not interfere with Lo-Res in the new one.

Although Mr. Carlson stated the syntax requirements of the parser in his June, 1980 article, some of you may not have read that, so I will repeat such. A "%" or "#" must precede the special one-keystroke commands. In immediate mode, they will be executed before the BASIC interpreter knows that they were even there. In deferred mode, the parser will not accept X #EXPR, but will execute it at once. You must enter it as X %EXPR. The parser will change % to # in sending the input line to memory.

Not only do these routines save typing, but they do not have to be interpreted. The BASIC interpreter takes time in finding and calling up the proper routines. A REAL compiler would look up these routines, write code for the variables for the routine to work on, and set up 20's and 4C's for the bare routines in BASIC.

To restore the parser to original form (and allow the area 300-3CF to be freed up for new code) one should CALL −151 into the monitor, and then enter

```
                    ;*
                    ;* EXTENDED PARSER FOR APPLE II
                    ;*
                    ;*      BY PAUL R. WILSON
                    ;*
                    ;
                    ;ERASES 1ST 6 BYTES OF PARSER AND REPLACES WITH
                    ;4C 15 03 AND THREE NOP'S
                    ;
                           ORG $300

                    ;
0300 A94C                  LDA #$4C
0302 85B1                  STA $B1
0304 A915                  LDA #$15
0306 85B2                  STA $B2
0308 A903                  LDA #$03
030A 85B3                  STA $B3
030C A9EA                  LDA #$EA
030E 85B4                  STA $B4
0310 85B5                  STA $B5
0312 85B6                  STA $B6
0314 60                    RTS               ;RETURN TO BASIC OR PROGRAM
0315 E6B8                  INC $B8           ;FIRST 6 BYTES
0317 D002                  BNE LBLA          ;OF NORMAL
0319 E6B9                  INC $B9           ;PARSER CODE
031B A5B8           LBLA   LDA $B8
031D 8D2803                STA $0328
0320 A5B9                  LDA $B9
0322 8D2903                STA $0329
0325 AD0502                LDA $0205
0328 C923                  CMP #$23          ;IS THE # SIGNAL GIVEN?
032A F00D                  BEQ LBLC          ;IF SO, REENTER EXTENDED PARSER
032C C925                  CMP #$25          ;IS THE % SIGNAL GIVEN?
032E D006                  BNE LBLB          ;IF NOT, BACK TO THE BASIC LINE
0330 A000                  LDY #$00          ;CHANGE % TO # IN STORING THE LINE IN MEMORY
0332 A923                  LDA #$23
0334 91B8                  STA ($B8),Y
```

(Continued)

B1:E6 B8 D0 02 E6 B9 N B1L by hand to patch, and disassemble the parser code and check it for proper restoration.

To save this routine simply type BSAVE EXTENDED PARSER, A$300, L$A0 and the disk system will do the rest. Lock the file for safety. A later long file or lack of space may attempt an over-write of an unlocked file.

A program written with extensive use of the extended parser commands will run only with the parser up and running. Otherwise it will crash with SYNTAX ERRORS.

If you carefully enter this as shown above, and save it to disk, you'll be able to use it in many Applesoft programs. I went over the code carefully both in writing it and in transcribing above, so I see no margin for errors. Happy parsing!

Paul R. Wilson is currently employed at Baruch College, NYC, as a lab technician in Natural Sciences. He has found a self-sustaining hobby in home computers and is especially interested in trying to revive LIFELINE on his Apple II.

MICRO

```
0336 4CB700   LBLB  JMP $00B7    ;BACK TO PARSING THAT LINE:
0339 20B100   LBLC  JSR $00B1    ;TEST FOR CHARACTER FOLLOWING # OR %
033C C953           CMP 'S       ;IS IT AN 'S'?
033E F01B           BEQ LBLD     ;IF SO, GO TO SCRCLR
0340 C945           CMP 'E       ;IS IT AN 'E'?
0342 F01D           BEQ LBLE     ;IF SO, GO TO ENDCLR
0344 C94C           CMP 'L       ;IS IT AN 'L'?
0346 F01F           BEQ LBLF     ;IF SO, GO TO LNCLR
0348 C946           CMP 'F       ;F?
034A F021           BEQ LBLG     ;TO FLASH
034C C949           CMP 'I       ;I?
034E F024           BEQ LBLH     ;TO INV
0350 C94E           CMP 'N       ;N?
0352 F027           BEQ LBLI     ;TO NORMAL
0354 C954           CMP 'T       ;T?
0356 F02A           BEQ LBLJ     ;TO TEXT
0358 4CB100         JMP $00B1    ;IF NONE OF ABOVE, BACK TO PARSER
035B 2058FC   LBLD  JSR $FC58    ;SCRCLR--SCREEN GOES DARK
035E 4CB100         JMP $00B1
0361 2042FC   LBLE  JSR $FC42    ;ENDCLR--CLEARS LINE
0364 4CB100         JMP $00B1
0367 209CFC   LBLF  JSR $FC9C    ;LNCLR--CLEARS LINE
036A 4CB100         JMP $00B1
036D A97F     LBLG  LDA #$7F     ;FLASH--OUTPUT INTO FLASH MODE
036F 8532           STA $32
0371 4CB100         JMP $00B1
0374 A93F     LBLH  LDA #$3F     ;INV--REVERSE FIELD
0376 8532           STA $32
0378 4CB100         JMP $00B1
037B A9FF     LBLI  LDA #$FF     ;NORM--RESET TO NORMAL OUTPUT
037D 8532           STA $32
037F 4CB100         JMP $00B1
0382 AD54C0   LBLJ  LDA $C054    ;RESTORES PAGE 1 OF SCREEN ($400-$7FF)
0385 AD51C0         LDA $C051    ;RESTORES TEXT MODE
0388 AD56C0         LDA $C056    ;RESTORES PROPER FUNCTION OF LORES GRAPHICS
038B A900           LDA #$00
038D 8520           STA $20      ;LEFT SIDE
038F 8522           STA $22      ; AND TOP OF SCREEN RETURN TO FULL
0391 A928           LDA #$28
0393 8521           STA $21      ;SCREEN RETURNS TO FULL WIDTH
0395 A918           LDA #$18
0397 8523           STA $23      ;BOTTOM OF SCREEN GOES TO BOTTOM
0399 4CB100         JMP $00B1
```

SEARCH

This program is appropriately entitled SEARCH. It is a utility routine designed to aid in writing and editing programs in Integer BASIC.

R.C. Merten
12307 Oak Street
Omaha, Nebraska 68144

This program's main function is to input a string of characters, variables, punctuation, etc. Then, search through the BASIC program in memory and print to the current output device any numbered lines in which a match has been found.

Several similar programs are available either commercially or in the literature. The problem is that most of them are used with Applesoft, or that special care and handling must be taken to separate the ASCII strings from tokenized material.

SEARCH can be used only on systems with Integer BASIC and the Sweet-16 interpreter in ROM. A language card loaded with Integer BASIC can also be used. It can be used with printers or any version DOS without modification. DOS does not have to be reconnected after running.

The program will make comparisons exactly as they would be printed during a listing of the program (including leading and trailing blanks). It will also find control characters (*i.e.*, control D) scattered throughout the program.

To use SEARCH, first load it in $300 to $3F4 and then type 300G from monitor level, or CALL 768 from BASIC. The screen will prompt you with ENTER STRING. Type in the characters to be searched for and hit RETURN. The program will print each numbered line in which a match is found. If you wish to stop the display from scrolling off the screen, push any key. Subsequently, pushing the space bar will display one line at a time. Pushing RETURN will abort the search program and return to BASIC.

SEARCH uses the Sweet-16 interpreter, and since many assemblers cannot handle these instructions, a hex dump has been provided. Using the Sweet-16 to handle 16-bit numbers reduces the equivalent amount of 6502 code used by 60 to 70 percent.

How the Program Works

When called, SEARCH uses the NXTCHR routine to enter your string into the standard input buffer starting at $200. If the only character you enter is a carriage return, the program immediately aborts and returns to BASIC. Normally, though, it starts building an array at $2000. The array contains the beginning addresses of all the BASIC program lines.

Next, it saves the output hooks and replaces them with a pointer to the subroutine called CATCH. The Integer BASIC LIST routine at $E04B is called and every listed character is sent to CATCH instead of the screen. CATCH checks each character as it is sent and tries to match it to the string that is still sitting in the input buffer. When a match is identified, the address of the last listed character in the BASIC pro-

```
********************************
*                              *
*     SEARCH INTEGER BASIC     *
*                              *
*        BY R.C. MERTEN        *
*          11/17/80            *
*                              *
*           REVISED            *
*          11/20/80            *
*                              *
********************************
ZERO        EQU   $00
R1          EQU   $01
R2          EQU   $02
R3          EQU   $03
R4          EQU   $04
R5          EQU   $05
R6          EQU   $06
R7          EQU   $07
R8          EQU   $08
LISTN       EQU   $E2
PPL         EQU   $CA
HIMEM       EQU   $4C
CSWL        EQU   $36
CSWH        EQU   $37
BUFF        EQU   $200
LNADD       EQU   $2000
FOUND       EQU   $2800
CATCH1      EQU   $27F0
LENGTH      EQU   $27F1
HOLD        EQU   $27F2
YSAV        EQU   $27F4
KBSTB       EQU   $C010
KBD         EQU   $C000
CROUT       EQU   $FD8E
COUT        EQU   $FDED
LIST        EQU   $E04B
LISTIT      EQU   $E06D
NXTCHR      EQU   $FD75
SW16        EQU   $F689
```

(Continued)

gram can now be found at $E2 and $E3. This address is put into the array called FOUND which starts building at $2800.

When LIST is finished, the output hooks are returned to their original values. Sweet-16 is again called to determine which line # the FOUND variable belongs in. The beginning address of that line # is placed in $E2 and $E3 and LISTIT ($E06D) is called to print that line to the screen. A short delay follows, along with a check to see if a key has been pushed, and the program continues. At the end, Integer BASIC is re-entered through the warm start routine at $E003.

For those who would like to expand on this program, the routines can easily be adapted to other purposes. For instance, it is sometimes quite handy for BASIC programmers to insert disallowed commands such as HIMEM, LOMEM or DELETE into a BASIC program. Finding the HEX address of the command within the program is difficult, especially if it is not near the start of the program. With these routines and a little ingenuity, finding the exact location in memory of any command can quite easily be found.

The SEARCH seems to be quite bulletproof with one exception. If an Integer program contains an assembly language routine this will sometimes cause it to hang up. The problem could have been corrected but it would make the SEARCH program greater than one page long.

If page 3 is already in use the SEARCH program can easily be relocated to any other portion of memory. There are, however, five locations that must be changed by hand if you are not using an assembler. These locations are one load and two jump instructions at $30A, $328 and $3AB. Also the pointers to the catch routine which are set up at $359 and $35D must be changed.

Hope you find what you're SEARCHing for.

For about the last 10 years Richard Merten has explored the electronics field both as a job and hobby. He is employed by the Union Pacific Railroad in the Communications Department. He bought his Apple about two years ago and has enjoyed designing both hardware and software for it. Some of his projects include his own version of a 16K expansion board and a totally programmable RS-232 communicative interface card, and a facsimile interface to allow both transmission and reception of Apple's Hi-Res screens.

```
                    RESTART   EQU   $E003
                    WAIT      EQU   $FCA8
                    *
0300:  20 8E FD     BEGIN     JSR   CROUT
0303:  A2 0C                  LDX   #$0C
0305:  A0 00                  LDY   #$00
0307:  8C F0 27               STY   CATCH1.   * ZERO INPUT COUNT
030A:  B9 E8 03     PRINT     LDA   TABLE,Y   * PRINT
030D:  09 80                  ORA   #$80
030F:  20 ED FD               JSR   COUT.
0312:  C8                     INY
0313:  CA                     DEX
0314:  D0 F4                  BNE   PRINT.    * NEXT LETTER
0316:  20 8E FD               JSR   CROUT.
0319:  20 75 FD               JSR   NXTCHR.
031C:  8E F1 27               STX   LENGTH.
031F:  A9 00                  LDA   #$00
0321:  8D 10 C0               STA   KBSTB.    * CLEAR STROBE
0324:  E0 00                  CPX   #$00
0326:  D0 03                  BNE   OVER.
0328:  4C B5 03               JMP   DONE.
032B:  20 89 F6     OVER      JSR   SW16.   ** LINE # ARRAY **
032E:  12 CA 00               SET   R2 PPL.
0331:  13 4C 00               SET   R3 HIMEM.
0334:  17 00 20               SET   R7 LNADD.
0337:  63                     LDD   @R3.
0338:  33                     ST    R3.       * GET HIMEM
0339:  62                     LDD   @R2.      * FIRST LINE ADD
033A:  32                     ST    R2.
033B:  31                     ST    R1.       * SAVE FOR LATER
033C:  D3           LOOP1     CPR   R3.       * AT END OF PROG?
033D:  03 07                  BC    OUT.
033F:  77                     STD   @R7.      * LINE ADD. ARRAY
0340:  42                     LD    @R2.      * GET INDEX
0341:  A1                     ADD   R1.       * MAKE NEW ADD.
0342:  32                     ST    R2.       * SAVE IT
0343:  31                     ST    R1.       * SAVE FOR LATER
0344:  01 F6                  BR    LOOP1.
0346:  23           OUT       LD    R3.       * HIMEM TO ARRAY
0347:  77                     STD   @R7.
0348:  17 00 28               SET   R7 FOUND * SETUP ARRAY
034B:  16 00 00               SET   R6 ZERO. * FOUND COUNTER
034E:  00                     RTN
034F:  A5 36                  LDA   CSWL.     * SAVE CSWL HOOK
0351:  8D F2 27               STA   HOLD.
0354:  A5 37                  LDA   CSWH.
0356:  8D F3 27               STA   HOLD+1
0359:  A9 BD                  LDA   #<CATCH. * POINT TO CATCH
035B:  85 36                  STA   CSWL.
035D:  A9 03                  LDA   #>CATCH.
035F:  85 37                  STA   CSWH.
0361:  20 4B E0               JSR   LIST.     * LIST TO CATCH
0364:  AD F2 27               LDA   HOLD.     * RESTORE HOOK
0367:  85 36                  STA   CSWL.
0369:  AD F3 27               LDA   HOLD+1
036C:  85 37                  STA   CSWH.
036E:  20 89 F6               JSR   SW16.   ** PRINT LINES **
0371:  26                     LD    R6.       * DONE IF ZERO
0372:  06 40                  BZ    DONE1.
0374:  17 00 20               SET   R7 LNADD.
0377:  12 00 00               SET   R2 ZERO. * FOR COMPARASON
037A:  13 00 28               SET   R3 FOUND.* START OF ARRAY
037D:  63           LOOP3     LDD   @R3       * GET FOUND ADD.
037E:  34                     ST    R4.       * HOLD
037F:  67           LOOP2     LDD   @R7.      * ADD. NEXT LN
0380:  D4                     CPR   R4.
0381:  02 FC                  BNC   LOOP2.
0383:  C7                     POPD  @R7       * BACKUP TWO
0384:  C7                     POPD  @R7.
0385:  D2                     CPR   R2.
0386:  06 29                  BZ    SAME.
0388:  32                     ST    R2.
0389:  18 E2 00               SET   R8 LIST#.
038C:  78                     STD   @R8.      * ADD OF LINE#
038D:  00                     RTN
038E:  20 6D E0               JSR   LISTIT.   * OUTPUT LINE
0391:  A9 00                  LDA   #$00
0393:  20 A8 FC               JSR   WAIT.
0396:  AD 00 C0               LDA   KBD.
0399:  10 13                  BPL   AROUND.
039B:  A9 00                  LDA   #$00
039D:  8D 10 C0               STA   KBSTB.
03A0:  AD 00 C0     LOOP4     LDA   KBD.
```

```
03A3:  C9 A0                 CMP   #$A0
03A5:  F0 07                 BEQ   AROUND.
03A7:  C9 8D                 CMP   #$8D.
03A9:  D0 F5                 BNE   LOOP4.
03AB:  4C B5 03              JMP   DONE.
03AE:  20 89 F6     AROUND   JSR   SW16.    ** BACK AGAIN **
03B1:  F6           SAME     DCR   R6.       * REDUCE COUNT
03B2:  07 C9                 BNZ   LOOP3.
03B4:  00           DONE1    RTN
03B5:  A2 00        DONE     LDX   #$00
03B7:  8E 10 C0              STX   KBSTB.
03BA:  4C 03 E0              JMP   RESTART.
03BD:  8C F4 27     CATCH    STY   YSAV.
03C0:  AC F0 27              LDY   CATCH1.
03C3:  D9 00 02              CMP   BUFF,Y
03C6:  D0 17                 BNE   CLEAR.
03C8:  C8                    INY
03C9:  CC F1 27              CPY   LENGTH.
03CC:  F0 07                 BEQ   SETIT.
03CE:  8C F0 27              STY   CATCH1.
03D1:  AC F4 27              LDY   YSAV.
03D4:  60                    RTS
03D5:  20 89 F6     SETIT    JSR   SW16.    ** LOAD ARRAY **
03D8:  12 E2 00              SET   R2 LIST#.
03DB:  62                    LDD   @R2.      * ADD FROM BASIC
03DC:  77                    STD   @R7.      * INTO FOUND ARRAY
03DD:  E6                    INR   R6.       * INCREASE COUNT
03DE:  00                    RTN
03DF:  A0 00        CLEAR    LDY   #$00
03E1:  8C F0 27              STY   CATCH1.   * ZERO STRING COUNT
03E4:  AC F4 27              LDY   YSAV.
03E7:  60                    RTS
03E8:  45 4E 54     TABLE    ASC   'ENTER STRING'
03F4:  00                    BRK
```

Applesoft Error Messages from Machine Language

The methods and data required to utilize Applesoft error messages in assembly language are presented. Use of these routines should be limited to assembly language routines that are interfaced with Applesoft programs.

Steve Cochard
P.O. Box 236
Boyertown, Pennsylvania 19512

Did you ever wonder how Applesoft generates its error messages? While writing an assembly language program that interfaced with Applesoft I found I needed more than just the simple "Syntax Error", which was the only one I knew how to utilize.

I started my search for the "errors" by looking at the machine code for the "Syntax Error" message which is located at $DEC9. It consists of only two commands:

```
LDX #$10
JMP $D412
```

This short routine, it seemed, was intended only to load the X register with the starting address of the word "SYNTAX" in a table of all error messages. This deduction proved true, and with a little more searching in the $D412 routine the table was found.

The error message table is located at $D260 and is 240 bytes long. By loading the X register with the appropriate index and then jumping to the $D412 routine, it is possible to utilize any error message from machine language or Applesoft.

Table 1 shows the values to be loaded into the X register to generate any of the available 17 messages. Listings 1 and 2 show very short machine and Applesoft programs to verify that this is true. Listing 3 shows a program that will list the entire table.

It should be noted that this procedure, if utilized in machine language, performs exactly as if the error had occurred in an Applesoft program. The error message is printed, the "bell" rings, the last executed line number is printed and the program stops. If an "ONERR GOTO" statement had been executed previously, the program will again operate as if the error had occurred in Applesoft, the object line of the "ONERR GOTO" will be jumped to and executed. Happy Errors!

Table 1

Value of X register	Error message
0	NEXT WITHOUT FOR
16	SYNTAX
22	RETURN WITHOUT GOSUB
42	OUT OF DATA
53	ILLEGAL QUANTITY
69	OVERFLOW
77	OUT OF MEMORY
90	UNDEF'D STATEMENT
107	BAD SUBSCRIPT
120	REDIM'D ARRAY
133	DIVISION BY ZERO
149	ILLEGAL DIRECT
163	TYPE MISMATCH
176	STRING TOO LONG
191	FORMULA TOO COMPLEX
210	CAN'T CONTINUE
224	UNDEF'D FUNCTION

***Listing 1:* Enter from the monitor to interface with program listing 2.**

```
300:LDX $0306
303:JMP $D412
```

***Listing 2:* Applesoft program to print error messages.**

```
10 INPUT "WHAT VALUE OF X ? ";X
20 POKE 784,X
30 CALL 768
```

***Listing 3:* This short program will list the entire table. Enter it from the monitor and then type in 300G.**

```
300:LDX #$00
302:LDA $D260,X
305:EOR #$80
307:BMI $0310
309:ORA #$80
30B:JSR $FDED
30E:LDA #$8D
310:JSR $FDED
313:INX
314:CPX #$FF
316:BNE $0302
318:RTS
```

Steve Cochard is one of the principals of Scientific Software, the author of the "Scientific Software Sweet 16 Assembler." He is a structural engineering supervisor with a large Engineering/Construction firm. Current activities with the Apple computer include development of Structural Analysis and Design systems, various machine language utilities, and a machine language floating point array/matrix manipulation package for use with Applesoft BASIC.

MICRO

Trick DOS

Apple DOS obviously is a live entity. It was created by a supreme being at Cupertino to mystify, amaze and tantalize us common folk. Let us literally turn the tables!

Sanford M. Mossberg
50 Talcott Road
Port Chester, N.Y. 10573

On booting a disk, the DOS command table (DCT) comes to reside at RAM locations $A884-$A908 (decimal 43140-43272). The last letter of each of the 28 DOS commands is represented by a high byte ASCII character which signals the end of the command. Other letters or numerals are written in low byte code. A zero marks the end of the DCT. Armed with these simple facts, we can trick DOS 3.2 or 3.3 into obeying our whims and desires.

Listing 1 provides code for TRICK DOS. Following initialization (lines 2000-2060) and optional instructions (lines 2500-2670), a menu is presented (lines 600-710), each item of which is analyzed:

1. *Display Current DOS Command Table:* The heart of the entire program is found in the subroutine at lines 100-180. The starting location (START) of the table never changes. Lines 120-130 search successive memory locations in the DCT until a zero byte is found. The end address of the table, not including the zero byte, is assigned to the variable FIN. Line 140 initializes the array DOS$(*,*), the contents of which are noted in line 102. Lines 150-180 PEEK DCT locations, fill the two-dimensional matrix and create a string (DOS$) which contains every character in the DCT. Subsequently, the array variables will be used to format screen display (lines 860-880 and 1060-1070), and the string variable will be manipulated to alter the command table by POKEing data into RAM. The displayed DCT may be listed to a printer (see figure 1).

Figure 1: **Current DOS Commands and Addresses**

DEC	HEX		DEC	HEX	
43140	A884	INIT	43206	A8C6	APPEND
43144	A888	LOAD	43212	A8CC	RENAME
43148	A88C	SAVE	43218	A8D2	CATALOG
43152	A890	RUN	43225	A8D9	MON
43155	A893	CHAIN	43228	A8DC	NOMON
43160	A898	DELETE	43233	A8E1	PR#
43166	A89E	LOCK	43236	A8E4	IN#
43170	A8A2	UNLOCK	43239	A8E7	MAXFILES
43176	A8A8	CLOSE	43247	A8EF	FP
43181	A8AD	READ	43249	A8F1	INT
43185	A8B1	EXEC	43252	A8F4	BSAVE
43189	A8B5	WRITE	43257	A8F9	BLOAD
43194	A8BA	POSITION	43262	A8FE	BRUN
43202	A8C2	OPEN	43266	A902	VERIFY

2. *Change DOS Command Table:* The program block starting at line 1000 first outputs current commands by utilizing the routine described earlier. The command to be changed (OC$) is requested in line 1080. Since keyboard input is in low byte code, the high bit of the final letter is turned on (line 1090). The validity of the command is checked in line 1100 and variable PT marks the position of the command in the array. An invalid command triggers an error message (line 1110) and returns the user to the prior input request. The replacement command (NC$) is solicited in line 1130 and high byte conversion occurs in line 1140. The subroutine at lines 400-500 rearranges the DCT. Commands preceding and following the changed command are contained in T1$ and T3$, respectively; the new command is placed in T2$. In line 460, DOS$ is recreated by concatenation of the above-noted strings. Lines 470-500 POKE the new command table into memory. An incidental, but important, feature of this entire section, and others, is the effective error trapping (lines

1080, 1110, 1120, 1130, 1170, 1180, 1210 and 1240) which prevents potential crashing of the program and assures professionally formatted screen display.

3. *Restore Normal DOS Command Table* and

4. *Try Sandy's Commands:* Data statements in lines 2100-2110 contain ASCII code for the normal DCT. Line 1330 reads the data into the variable NDOS$. A sample table which I have found useful is coded in lines 2120-2130. Line 1340 produces MYDOS$. Lines 1380-1390 replace the resident DCT with either of these strings, thus restructuring the entire command table rapidly.

5. *Exit Program:* At program termination all text and graphics modes should be normalized. Line 1510 accomplishes this by successively turning off Hi-Res, turning on text page one, clearing the keyboard strobe and setting a full text window. Although TRICK DOS does not require these steps, the habit is a good one to cultivate. After the program ends, the new command table will remain viable in RAM until rebooting occurs or power is discontinued. If you so desire, the new DCT can be preserved permanently by initializing a disk.

Knowing that DOS intercepts and reviews all commands before the Applesoft interpreter can process the command, several admonitions are appropriate. Each newly-created DOS command should have a character set that does not duplicate the first letters of any Applesoft BASIC command. To better understand this pitfall, imagine that we have changed "LOAD" to "L" and "RENAME" to "RE". Now, if we type "LIST" or "LEFT$", DOS understands this to mean LOAD (L = LOAD) the file "IST" or "EFT$", and the "FILE NOT FOUND" error message is returned. Typing "REM" would produce the same error message as DOS attempted to RENAME (RE = RENAME) the nonexistent file "M." So far this is annoying but not harmful.

Consider the dire results from changing "INIT" to "I." Any Applesoft command beginning with an "I" would promptly start initializing the disk. This would be catastrophic and must be avoided! For the reasons cited above, I advise you to peruse a list of Applesoft BASIC commands before modifying a DOS command. Changing "LOAD" to "LD", "RENAME" to "RNM" and "INIT" to "I*" would have avoided the chaos. Choice #4 from the menu will create a table of "safe" commands that I have found to be functional.

When you begin using a newly created DCT, mistakes will be inevitable and error messages will proliferate. The DCT commands "LOAD" and "SAVE" are special, in that they also exist as Applesoft commands to a cassette recorder. If either is used erroneously, the system will hang. Only by pressing "RESET" can you recover. If you do not have autostart ROM, altering these two commands may be more of a nuisance than an aid.

Experiment freely and enjoy your newfound power over DOS.

Sandy Mossberg is a physician who had no computer experience until he purchased an Apple II in February, 1980. His obsession is programming. He writes a monthly column for his computer club's publication *The APPLESHARE Newsletter*. The column is entitled, "Basic Tips and Technics" and deals with many aspects of Applesoft programming and DOS function.

Listing 1

```
10  REM   TRICK DOS

          BY SANDY MOSSBERG

20  TEXT : CALL  - 936: POKE  - 1
     6298,0: POKE  - 16300,0: POKE
      - 16368,0
30  GOSUB 2010: GOSUB 3010: GOSUB
     2510: GOTO 610
100  REM

         PEEK COMMAND TABLE
          AND CREATE ARRAY

102  REM ARRAY DOS$(R1-28,C1-2)
         C1=COMMAND
         C2=START ADDR

104  REM DOS$=DOS COMMAND TABLE

106  REM DOS=ADDR COMMAND TABLE

110 TM = START
120  IF  PEEK (TM) = 0 THEN FIN =
     TM - 1: GOTO 140: REM  FIND
     END OF TABLE
130 TM = TM + 1: GOTO 120
140 I = 1: FOR J = 1 TO 29: FOR K
      = 1 TO 2:DOS$(J,K) = "": NEXT
     K,J:DOS$(1,2) =  STR$ (START
     ):DOS$ = "": REM INITIALIZE
150  FOR DOS = START TO FIN
160  IF  PEEK (DOS) > 127 THEN DOS$(I,1
     ) = DOS$(I,1) +  CHR$ ( PEEK (DOS)
     ):DOS$ = DOS$ +  CHR$ ( PEEK (DOS)
     ):DOS$((I + 1),2) =  STR$ (DOS + 1
     ):I = I + 1: GOTO 180: REM  IF HI
      BYTE INCR I

170 DOS$(I,1) = DOS$(I,1) +  CHR$
     ( PEEK (DOS)):DOS$ = DOS$ +
      CHR$ ( PEEK (DOS))
180  NEXT DOS: RETURN

300  REM

         DEC --> HEX

310 HD% = DOS / 256:NBR = HD%: GOSUB
     340:HB$ = HEX$
320 LD% =  FN MOD(DOS):NBR = LD%:
      GOSUB 340:LB$ = HEX$
330 HEX$ = HB$ + LB$: RETURN
340 H% = NBR / 16 + 1:L% = NBR /
     16:L = L% * 16:L% = NBR - L +
     1
350 HEX$ =  MID$ (H$,H%,1) +  MID$
     (H$,L%,1): RETURN
400  REM

          REORGANIZE
         COMMAND TABLE

410  IF PT = 1 THEN T1$ = "": GOTO
     430
420 T1$ =  LEFT$ (DOS$, VAL (DOS$
     (PT,2)) - START)
430  FOR I = 1 TO  LEN (NC$):T2$ =
     T2$ +  MID$ (NC$,I,1): NEXT

440  IF PT = 28 THEN T3$ = "": GOTO
     460
450 T3$ =  RIGHT$ (DOS$,FIN + 1 -
      VAL (DOS$((PT + 1),2)))
460 DOS$ = T1$ + T2$ + T3$:T2$ =
     ""
470 DOS = START
480  FOR I = 1 TO  LEN (DOS$): POKE
     DOS, ASC ( MID$ (DOS$,I,1)):
     DOS = DOS + 1: NEXT
490 FIN = FIN +  LEN (NC$) -  LEN
     (OC$)
500  POKE FIN + 1,0: RETURN
600  REM

         MENU

610  HOME :TT$ = "============="
     : GOSUB 3110
620 TT$ = "TRICK DOS MENU": GOSUB
     3110
630 TT$ = "=============": GOSUB
     3110
640  VTAB 6: PRINT "1.DISPLAY CUR
     RENT DOS COMMAND TABLE.": PRINT

650  PRINT "2.CHANGE DOS COMMAND
     TABLE.": PRINT
660  PRINT "3.RESTORE NORMAL DOS
     COMMAND TABLE.": PRINT
670  PRINT "4.TRY SANDY'S COMMAND
     S.": PRINT
680  PRINT "5.EXIT PROGRAM.": PRINT
     : PRINT
690  VTAB 17: CALL  - 958: PRINT
     "     WHICH CHOICE? ";: GET I
     $: PRINT I$:CH =  VAL (I$)
700  IF CH < 1 OR CH > 5 OR I$ =
     "" THEN 690
710  ON CH GOTO 800,1000,1300,130
     0,1500
800  REM

         DISPLAY CURRENT TABLE

810  HOME :TT$ = "================
     ================": GOSUB 31
     10
820 TT$ = "CURRENT DOS COMMANDS &
      ADDRESSES": GOSUB 3110
830 TT$ = "=======================
     ==========": GOSUB 3110
840  IF  NOT FF THEN  VTAB 8: INVERSE
     :TT$ = " READING DOS COMMAND
      TABLE ": GOSUB 3110: NORMAL
```

```
850  GOSUB 110: VTAB 4: CALL  - 9
     58
860  PRINT : HTAB 2: INVERSE : PRINT
     "DEC";: HTAB 8: PRINT "HEX";
     : HTAB 22: PRINT "DEC";: HTAB
     28: PRINT "HEX": NORMAL : PRINT

870  FOR I = 1 TO 14
880  PRINT DOS$(I,2)" ";:DOS =  VAL
     (DOS$(I,2)): GOSUB 310: PRINT
     HEX$" "DOS$(I,1);: HTAB 21: PRINT

     DOS$((I + 14),2)" ";:DOS =  VAL
     (DOS$((I + 14),2)): GOSUB 31
     0: PRINT HEX$" "DOS$((I + 14
     ),1): NEXT
890  IF FF THEN  FOR I = 1 TO 5: PRINT

     : NEXT : RETURN
900  VTAB 22: PRINT "LIST TABLE T
     O PRINTER (Y/N) ? ";: GET I$

910  IF I$ = "Y" THEN FF = 1: HTAB
     1: CALL  - 998: CALL  - 958:
      PRINT B$: INVERSE : PRINT "
      TURN PRINTER ON AND PRESS A
     NY KEY ": PRINT : HTAB 10: PRINT
     " EXPECT A PAUSE ";: GET I$:
      PRINT : NORMAL : PRINT D$;D
     OS$(20,1);1: GOSUB 810:FF =
     0: PRINT D$;DOS$(20,1);0: GOTO
     610
920  IF I$ = "N" THEN 610
930  HTAB 1: GOTO 900
1000  REM
        CHANGE TABLE

1010  HOME :TT$ = "=============
     =": GOSUB 3110
1020 TT$ = "CHANGE COMMANDS": GOSUB
     3110
1030 TT$ = "===============": GOSUB
     3110
1040  VTAB 4: CALL  - 958: VTAB 8
     : INVERSE :TT$ = " READING D
     OS COMMAND TABLE ": GOSUB 31
     10: NORMAL
1050  GOSUB 110: VTAB 5: CALL  -
     958
1060  FOR I = 1 TO 7
1070  PRINT DOS$(I,1);: HTAB 10: PRINT

     DOS$((I + 7),1);: HTAB 20: PRINT
     DOS$((I + 14),1);: HTAB 30: PRINT

     DOS$((I + 21),1): NEXT
1080  VTAB 14: CALL  - 958: INPUT
     "TYPE COMMAND TO BE CHANGED:
      ";OC$: IF OC$ = "" THEN 118
     0
1090 OC$ =  MID$ (OC$,1, LEN (OC$
     ) - 1) +  CHR$ ( ASC ( RIGHT$
     (OC$,1)) + 128): REM TURN HI
      BIT ON IN LAST LETTER OF
     COMMAND
1100  FOR I = 1 TO 28: IF OC$ = D
     OS$(I,1) THEN PT = I: GOTO 1
     130: REM PT=POINTER TO
     POSITION OF COMMAND IN ARRAY

1110  IF I = 28 THEN  PRINT B$: VTAB
     16: INVERSE : PRINT " NOT A
     VALID CURRENT COMMAND ": NORMAL
     : FOR J = 1 TO 3000: NEXT : GOTO
     1080
1120  NEXT I
1130  VTAB 16: CALL  - 958: INPUT
     "TYPE NEW COMMAND: ";NC$: IF
     NC$ = "" THEN 1130
1140 NC$ =  MID$ (NC$,1, LEN (NC$
     ) - 1) +  CHR$ ( ASC ( RIGHT$
     (NC$,1)) + 128): REM TURN HI
      BIT ON IN LAST LETTER OF
     COMMAND
1150  PRINT B$: VTAB 18: HTAB 3: PRINT

     "CONFIRM (Y/N) ? ";: GET I$:
      PRINT I$
1160  IF I$ = "Y" THEN  VTAB 20: INVERS
E : PRINT " WRITING COMMAND TABLE
     ": GOSUB 410: VTAB 18: HTAB
     1: CALL  - 958: PRINT " CHAN
     GE COMPLETED ": NORMAL : GOTO
     1220
1170  IF I$ <  > "N" THEN  VTAB 1
     8: CALL  - 958: GOTO 1150
1180  VTAB 18: CALL  - 958: PRINT
     : PRINT "RETURN TO MENU OR T
     RY AGAIN (M/A) ? ";: GET I$:
      PRINT I$
1190  IF I$ = "A" THEN  GOTO 1080

1200  IF I$ = "M" THEN 610
1210  GOTO 1180
1220  VTAB 20: CALL  - 958: PRINT
     "ANOTHER CHANGE (Y/N) ? ";: GET
     I$: PRINT I$: IF I$ = "Y" THEN
     1040
1230  IF I$ = "N" THEN 610
1240  GOTO 1220
1300  REM

      RESTORE NORMAL TABLE OR
       INSTALL SANDY'S TABLE

1310  VTAB 20: INVERSE : PRINT "
     WRITING COMMAND TABLE ";
1320 NDOS$ = "":MYDOS$ = ""
1330  FOR I = 1 TO 132: READ D:ND
     OS$ = NDOS$ +  CHR$ (D): NEXT

1340  FOR I = 1 TO 67: READ D:MYD
     OS$ = MYDOS$ +  CHR$ (D): NEXT
     : RESTORE
1350 DOS = START
1360  IF CH = 3 THEN TM$ = NDOS$:
     TT$ = " NORMAL DOS COMMAND T
     ABLE REESTABLISHED ":FIN = S
     TART +  LEN (NDOS$) - 1
1370  IF CH = 4 THEN TM$ = MYDOS$
     :TT$ = " SANDY'S COMMAND TAB
     LE INSTALLED ":FIN = START +
      LEN (MYDOS$) - 1
1380  FOR I = 1 TO  LEN (TM$): POKE
     DOS, ASC ( MID$ (TM$,I,1)):D
     OS = DOS + 1: NEXT
1390  POKE FIN + 1,0
1400  HTAB 1: PRINT TT$: NORMAL :
      GOSUB 3210: HTAB 1: GOTO 69
     0
1500  REM

       END PROGRAM

1510  POKE  - 16298,0: POKE  - 16
     300,0: POKE  - 16368,0: TEXT
     : HOME
1520  VTAB 10: INVERSE :TT$ = " E
     ND OF TRICK DOS PROGRAM ": GOSUB
     3110: NORMAL
1530  VTAB 15: PRINT " INITIALIZI
     NG A DISK BEFORE REBOOTING":
      PRINT "WILL PRESERVE THE CU
     RRENT DOS COMMANDS"
1540  VTAB 22: END
2000  REM

       INITIALIZE

2010  DIM DOS$(30,2)
2020 D$ =  CHR$ (4):B$ =  CHR$ (7
     ):SS$ = "
      ": REM 21 SPACES
2030 H$ = "0123456789ABCDEF"
2040  DEF  FN MOD(X) = X -  INT (
     X / 256) * 256: REM SIMULATE
      MOD FUNCTION
2050 START = 43140: REM START OF
     TABLE
2060  RETURN
2100  DATA 73,78,73,212,76,79,65,
     196,83,65,86,197,82,85,206,6
     7,72,65,73,206,68,69,76,69,8
     4,197,76,79,67,203,85,78,76,
     79,67,203,67,76,79,83,197,82
     ,69,65,196,69,88,69,195,87,8
     2,73,84,197,80,79,83,73,84,7
     3,79,206,79,80,69,206,65,80,
     80,69,78,196
2110  DATA 82,69,78,65,77,197,67,
     65,84,65,76,79,199,77,79,206
     ,78,79,77,79,206,80,82,163,7
     3,78,163,77,65,88,70,73,76,6
     9,211,70,208,73,78,212,66,83
     ,65,86,197,66,76,79,65,196,6
     6,82,85,206,86,69,82,73,70,2
     17: REM NORMAL TABLE
2120  DATA 73,170,76,196,83,214,8
     2,85,206,67,72,206,68,204,76
     ,203,85,76,203,67,211,82,196
     ,69,88,195,87,210,80,83,206,
     79,208,65,208,82,69,206,67,6
     5,212,77,206,78,77,206,80,16
     3,73,163,77,65,216,70,208,73
     ,78,212,66,211,66,204,66,210
     ,86,69,210
2130  DATA 77,206,78,77,206,80,16
     3,73,163,77,65,216,70,208,73
     ,78,212,66,211,66,204,66,210
     ,86,69,210: REM
     SANDY'S TABLE
2500  REM

       INSTRUCTIONS

2510  HOME :TT$ = "===========":
      GOSUB 3110
2520 TT$ = "INSTRUCTIONS": GOSUB
     3110
2530 TT$ = "============": GOSUB
     3110
2540  VTAB 7: CALL  - 958: PRINT
     "DO YOU WANT INSTRUCTIONS (Y
     /N) ? ";: GET I$: PRINT I$: IF
     I$ = "N" THEN  RETURN
2550  IF I$ <  > "Y" THEN 2540
2560  POKE 34,4: VTAB 5: CALL  -
     958
2570  PRINT "1.THE DOS COMMAND TA
     BLE RESIDES AT RAM": PRINT "
      LOCATIONS $A884 TO $A908 (
     DEC 43140": PRINT "  TO 4327
     2).": PRINT
2580  PRINT "2.EACH COMMAND IS RE
     PRESENTED BY ASCII": PRINT "
      CHARACTER CODES. ONLY THE
     LAST LETTER": PRINT "  OF A
     COMMAND HAS THE HIGH BIT ON
     SO": PRINT "  THAT DOS CAN R
     ECOGNIZE THE END OF THE"
2590  PRINT "  COMMAND. NOTE THE
     EXAMPLES BELOW:": PRINT : PRINT
     "       LOAD = 4C 4F 41 C4": PRINT

     "       INIT = 49 4E 49 D4": PRINT

     "       RUN  = 52 55 CE": PRINT
     : PRINT
2600  PRINT "3.ZERO MARKS THE END
      OF THE TABLE."
```

```
2610  GOSUB 3210: HOME
2620  PRINT "4.THIS PROGRAM WILL
      ENABLE YOU TO ALTER": PRINT
      "  THE COMMAND TABLE. YOU MA
      Y DESIRE TO": PRINT "  CHANG
      E 'CATALOG' TO ";: INVERSE :
       PRINT "CAT";: NORMAL : PRINT
      " OR 'SAVE' TO ": PRINT "  "
      ;: INVERSE : PRINT "SV";: NORMAL

2630  PRINT ". BE SURE THAT YOUR
      NEW DOS COMMAND": PRINT "  D
      OES NOT DUPLICATE THE FIRST
      PART OF": PRINT "  AN APPLES
      OFT BASIC COMMAND, OTHERWISE
      ": PRINT "  UNUSUAL EVENTS M
      AY OCCUR. EXPERIMENT!"
2640  PRINT "  TIREDNESS OR SILLI
      NESS MAY RESULT IN": PRINT "
        WEIRD SYMBOLS!!!": PRINT
2650  PRINT "5.THESE MODIFICATION
      S WILL TRIGGER A": PRINT "
      SYNTAX ERROR IF A DIRECT OR
      DEFERRED": PRINT "  COMMAND
      UTILIZES 'NORMAL' TERMINOLOG
      Y."
2660  PRINT "6.";: INVERSE : PRINT
      "TRICK DOS";: NORMAL : PRINT
      " IS MENU-DRIVEN AND SELF-":
      PRINT "  PROMPTING. HAVE FU
      N!!!"
2670  POKE 34,0: GOSUB 3210: RETURN

3000  REM

         TITLE PAGE

3005  REM SF APPLE CORE FORMAT

3010  INVERSE : VTAB 4
3020  TT$ = SS$: GOSUB 3110: GOSUB
      3110
3030  TT$ = "       TRICK DOS
      ": GOSUB 3110
3040  TT$ = SS$: GOSUB 3110: GOSUB
      3110
3050  TT$ = "  BY SANDY MOSSBERG
      ": GOSUB 3110
3060  TT$ = SS$: GOSUB 3110: GOSUB
      3110: NORMAL
3070  VTAB 16:TT$ = "CUSTOMIZE YO
      UR SET OF DOS COMMANDS!": GOSUB
      3110
3080  GOSUB 3210: RETURN
3100  REM

         PRINT CENTER

3110  WIDTH = 20 - ( LEN (TT$) / 2
      ): IF WIDTH <  = 0 THEN  PRINT
      TT$: RETURN
3120  HTAB WIDTH: PRINT TT$: RETURN

3200  REM

         CONTINUE/END

3210  VTAB 23: HTAB 12: PRINT "[E
      SC] TO END"
3220  VTAB 24: PRINT  TAB( 8);"[S
      PACE] TO CONTINUE ";
3230  PRINT "[ ]";: HTAB 29: GET
      ZZ$: IF ZZ$ =  CHR$ (27) OR
      ZZ$ =  CHR$ (3) THEN  TEXT :
      HOME : GOTO 1510
3240  IF ZZ$ =  CHR$ (32) THEN  RETURN

3250  CALL  - 868: CALL  - 1008: GOTO
      3230: REM
```

MICRO

New Publications

(Continued from page 74)

Programming Languages; Elements of BASIC (Line Numbers Revisited, Blank Spaces, Variables, Arrays, Expressions); BASIC Statements (Remarks, Assignment Statements, Declaring Array and String Size, Branch Statements, Loops, Subroutine, Conditional Execution, Input and Output Statements, Halting and Resuming Program Execution); Functions (Numeric Functions, String Functions, System Functions, User-Defined Functions, Function Nesting). *Advanced BASIC Programming*—Direct Access and Control (Memory and Addressing); Using Peripheral Devices; Program Output and Data Entry (More About the PRINT Statement, PRINT Formatting Functions, Cursor Control and Special Video Effects, Text Windows, The CHR$ Function: Programming Characters in ASCII, Programming Data Entry, Forms Data Entry, Formatting Output, Programming Printers); Storing Data on Cassette; Program Optimization (Faster Programs, Compact Programs); Debugging; Immediate and Programmed Mode Restrictions. *The Disk II*—(About Disks, How Data is Stored on Disks, Locating Tracks and Sectors, Write Protecting); The Disk Operating System (Versions of DOS, Initializing Disks, Disk Files, Diskette Directory, Track/Sector List, Disk Crash); Booting the Disk II (How to Boot DOS); Beginning Disk Commands (CATALOG, LOAD, The Disk Version of the RUN Command, Specifying the Drive Number, Slot Specification, Volume Specification); More Disk II Commands (INIT, SAVE, DELETE, LOCK, UNLOCK, RENAME, VERIFY); Using DOS Commands in Programs; Using Disk Files (Using Sequential Files, How to Append to Sequential Files, The POSITION Command, Using Random-Access Files, A Practical Random-Access Example, The Byte Parameter); Other DOS Commands (EXEC, MAXFILES, Using DOS Debugging Aids); Machine Language (Binary Image) Disk Files (BSAVE, BLOAD, BRUN). *Graphics and Sound*—Low-Resolution Graphics (Setting Up the Graphics Page, Graphics Programming Statements); High-Resolution Graphics (Which Page Should You Use?, Setting Up the Graphics Display, Alternatives to HGR and HGR2, High-Resolution Colors, Plotting Points and Lines); Using High-Resolution Shapes) Defining Shapes, Assembling the Shape Table, Entering the Shape Table, Shape Drawing Commands); Apple II Sound (Operating the Speaker). *Machine Language Monitor*—(Accessing the Monitor, Leaving the Monitor); Functions of the Monitor (Examining the Microprocessor Registers, Altering Memory, Altering the Microprocessor Registers, Saving and Retrieving Memory with Apple II Peripherals, Moving and Comparing Blocks of Memory, The GO Command, Using the Printer, The Keyboard Command, Setting Display Modes, Eight-Bit Binary Arithmetic Using the Monitor, User-Definable Monitor Command); The Mini-Assembler (Accessing the Mini-Assembler, Monitor Commands in the Mini-Assembler, Leaving the Mini-Assembler, Instruction Formats, Using the Mini-Assembler, Disassembled Listings, Testing and Debugging Programs, Integrating Your Program with BASIC). *Compendium of BASIC Statements and Functions*—(Immediate and Programmed Modes, BASIC Versions, Nomenclature and Format Conventions); Statements (listed alphabetically); Functions (listed alphabetically). *Appendices: A.* Derived Numeric Functions; *B.* Editing Commands; *C.* Error Messages (Integer BASIC Error Messages, Applesoft Error Messages, DOS Error Messages); *D.* Intrinsic Subroutines; *E.* Useful PEEK and POKE Locations; *F.* BASIC Reserved Words (Integer BASIC, Applesoft, DOS); *G.* Memory Usage (General Memory Organization, The BASIC Language Interpreters, DOS Memory Requirements, Integer BASIC Memory Usage, Applesoft Memory Usage); *H.* Disk II Format (The Track/Sector List, The Directory); *I.* ASCII Character Codes and Applesoft Reserved Word Tokens; *J.* Hexadecimal-Decimal Integer Conversion Table; *K.* Bibliography; *L.* Screen Layout Forms. *Index.*

General Computer

Computer/Law Journal is a quarterly which began publication in 1978. It is published by the Center for Computer/Law (P.O. Box 54308 T.A., Los Angeles, California 90054). The journal covers such subjects as Patent Protection for Computer Software; Computer-Assisted Legal Research; Current Developments in Computer Law; Computer-Related Evidence Law; Electronic Funds Transfer Systems; and Computer Crime. Back issues are available. An annual subscription is $60.00 per volume in the U.S. and Canada, elsewhere $64.00.
ISSN: 0164-8756.

Bio-Medical

Medical Computer Journal: The Journal for Computers in Clinical Practice is a quarterly publication of the Doctor's Computer Club (42 East High Street, East Hampton, Connecticut 06424). It is supplemented by a quarterly newsletter called *Dr. Com Puter's Report.* The journal averages 24 pages per issue and the newsletter 4 pages. The journal covers such subjects as clinical practice, laboratory, ECG, X-ray, and system description. Both the journal and newsletter publish software programs. Subscription rates are $15.00 for members, $25.00 for organizations and anyone outside North America, and $10.00 for students and physicians in training.

Sorting with Applesoft

Applesoft BASIC makes special demands which often severely degrade the efficiency of a theoretically efficient sorting algorithm. This article presents Applesoft BASIC code for a sorting algorithm which avoids most of these special problems. Thus, this algorithm may be the best one to use in programs which require a large amount of sorting.

Norman P. Herzberg
32 Gulick Road
Princeton, New Jersey 08540

No, this is not another article on Shell's sort, or Heap sort. If you thought it was, then this article probably is just what you've been looking for.

Sorting alphanumeric data on the Apple using Applesoft BASIC can be very painful, because of "the dreaded garbage collection." As the Applesoft interpreter encounters string variables, it fills memory with the values of these strings, even though there may be only a few variables receiving these values. In a surprisingly short time memory is filled with old discarded string values (garbage). Once memory is full, Applesoft will 'tidy things up' throwing out all the garbage (outdated data) that has accumulated, so that only the current value remains for each variable in the program. In the worst cases this will take several minutes of computing time, even though the entire procedure is carried out in machine language. Forcing garbage collection, by calling the Applesoft function FRE(0) before memory gets full is of no help. The time it takes to perform the FRE function seems only to depend on the complexity and size of the string arrays in a program, not on the amount of garbage that has accumulated.

One requirement of an ideal sorting program for Applesoft is clear. I would like to sort without ever referring to any

```
1  REM       SORT DEMO
3  REM   NORMAN P. HERZBERG
4  REM
10  GOTO 1010
500  REM   SORT SUBROUTINE
510  FOR I = 0 TO NR - 1:S(I) = I + 1: NEXT :S(NR) = 0
520 START = 1: IF NR < 2 THEN 700
530 F = 1:TM = 0:I = S(0)
540  IF L > 1 THEN 650: REM  SORT ON VALUE
550  FOR DX = 0 TO 1: IF I <  = 0 THEN 580
555 C = I:T1 = 0:US = I:UP = I * I = S(I): IF I <  = 0 THEN 575
560  FOR JJ = 0 TO 1: IF N$(I,S) < N$(C,S) THEN S(T1) = I:T1 = I: GOTO 57
    0
565 S(UP) = I:UP = I
570 I = S(I):JJ = (I <  = 0): NEXT
575 S(UP) = I:S(T1) =  - US:I = S(0): GOTO 590
580  IF F THEN F = 0:START = S(0)
585 I =  - I:S(TM) = I:TM = I:I = S(I)
590 DX = (I = 0): NEXT
595  GOTO 700: REM NOW MOVE THE DATA
650  FOR DX = 0 TO 1: IF I <  = 0 THEN 680
655 C =  VAL (N$(I,S) + " "):T1 = 0:US = I:UP = I:I = S(I): IF I <  = 0 THEN
    675
660  FOR JJ = 0 TO 1: IF  VAL (N$(I,S) + " ") < C THEN S(T1) = I:T1 = I: GOTO
    670
665 S(UP) = I:UP = I
670 I = S(I):JJ = (I <  = 0): NEXT
675 S(UP) = I:S(T1) =  - US:I = S(0): GOTO 690
680  IF F THEN F = 0:START = S(0)
685 I =  - I:S(TM) = I:TM = I:I = S(I)
690 DX = (I = 0): NEXT
700 S(0) =  ABS (START)
710  PRINT " SORTING": REM   NOW REARRANGE THE DATA
720 I = S(0): FOR JJ = 1 TO NR:R(JJ) = I:I = S(I): NEXT
730  FOR I = 1 TO NR:S(R(I)) = I: NEXT
740  FOR J = 1 TO NR - 1: FOR I = 1 TO NH: & N$(J,I),N$(R(J),I): NEXT
750 TEMP = R(J):R(S(J)) = TEMP:R(J) = JS(TEMP) = S(J):S(J) = J
760  NEXT J
800  PRINT G$">>>>> SORTED"
810  PRINT "PRESS SPACE-BAR TO CONTINUE ";: GET Z$: RETURN
1000  REM    INITIALIZATION
1010 D$ =  CHR$ (4):G$ =  CHR$ (7): TEXT : HOME
1020  VTAB 10: HTAB 15
1030  PRINT "SORT DEMO "
1040  GOSUB 5010: REM    &-STRING SWAP INITLZ. !!   DESCRIBED IN CALL A.P
    .P.L.E.  JAN.  1980  PG. 37
1050 NR = 50:NH = 2: REM    50 LONG FILE WITH 2 FIELDS
1060  DIM N$(NR,NH),R(NR),S(NR),H$(2): REM    HEADER ARRAY H$ IS ONLY FOR
    THE DEMO
1070 H$(1) = "NAME":H$(2) = "ADDRESS"
1080  FOR I = 1 TO NR:C$ = "":N% =  RND (1) * 26 + 193:C$ = C$ +  CHR$ (N
    %):N% =  RND (1) * 26 + 193:C$ = C$ +  CHR$ (N%):N$(I,1) = C$: NEXT

1090  FOR I = 1 TO NR:N% =  RND (1) * 9 + 1:N$(I,2) =  STR$ (N%): NEXT
1500  REM    MAIN LOOP
1510  TEXT : HOME : PRINT
1520  PRINT "——— SORT DEMO ———"
1530  PRINT "1.  LIST DATA "
1540  PRINT "2.  SORT DATA "
1550  PRINT "3.  EXIT "
```

(Continued)

```
1560  PRINT : INPUT "WHICH # ? (1,2,3) ? ";Z$:Z =  VAL (Z$ + " ")
1570  IF Z < 1 OR Z > 3 THEN 1560
1580  IF Z = 3 THEN  PRINT "O.K.": END
1590  ON Z GOSUB 3010,2010
1600  GOTO 1510
2000 RENS OR T
2010 MF = 1: GOSUB 4510
2020  INPUT "ENTER # OF FIELD FOR SORT ";S$:S$ = S$ + " ":S =  VAL (S$): IF
     S < 1 OR S > NH THEN 2020
2030  PRINT : PRINT "DO YOU WANT TO SORT:": PRINT
2040  PRINT "1 ALPHABETICALLY"
2050  PRINT "2 NUMERICALLY"
2060  PRINT " OR"
2070  PRINT "3 EXIT "
2080  PRINT "  (SORTING TAKES ABOUT "10 +  INT (.15 * NR *  LOG (NR))" SE
     C.)": PRINT
2090  INPUT "WHICH # ";L$:L$ = L$ + " ":L =  VAL (L$)
2100  IF L < 1 OR L > 3 THEN 2090
2110  IF L = 3 THEN  RETURN
2120  PRINT : PRINT "SORTING ";: GOSUB 510
2130  RETURN
3000  REM    REPORT
3010  HOME
3020  PRINT "REPORTING N$ IN FORM (NAME,ADDRESS) ": PRINT
3030 XX = 0
3040  FOR I = 1 TO NR:XX = XX + 1: IF XX = 5 THEN XX = 1: PRINT
3050  PRINT "(";
3060  FOR H = 1 TO NH - 1: PRINT N$(I,H);",";: NEXT
3070  PRINT N$(I,H);")  ";
3080  NEXT I
3090  PRINT
3100  VTAB 23: PRINT "PRESS SPACE-BAR TO CONTINUE ";: GET Z$
3110  RETURN
4500  REM    SUB MENU
4510  HOME : PRINT "SELECT FROM:": PRINT
4520  IF MF = 0 THEN  PRINT "0 "H$(0)
4530  FOR I = 1 TO NH: PRINT I" "H$(I): NEXT I: PRINT
4540  RETURN
5000  REM    &-STRING SWAP
5010  FOR I = 810 TO 855: READ PP: POKE I,PP: NEXT
5020  CALL 810
5030  RETURN
5040  REM    MACHINE LANGUAGE POKES
5050  DATA 169,76,141,245,3,169,58,141,246,2,169,3,141,247,3,96,32,227,22
     3,133,132,124,32,190,222,32,227,223,160,2,177,133,72,177,131,145,133
     ,104,145,131,136,16,143,96,0,
```

string arrays at all, and if that is impossible, I certainly want to avoid garbage collection. I was motivated to find such a sorting algorithm while trying to improve the File Cabinet data management program provided through Apple's Software Bank. For this program to be any real use, it should be possible to sort through a list of some 100-odd addresses in a reasonable amount of time.

One tool for accomplishing this appeared in the January 1980 issue of *Call A.P.P.L.E.* On page 37 appeared a String-Swap subroutine which generates no extra garbage strings at all! See lines 5000-5060 for the routine, and line 740 for its use. (The Ampersand calls the routine.) Using this routine and a crude exchange sort would seem to be the way to avoid most of the garbage collection problem. However, I have no grudge against garbage collection itself, only the large amount of time it takes. A poor exchange sort algorithm wastes more time than it saves.

My next idea was to adopt Shell's sort and the String-Swap subroutine. The key requirement is to continue to avoid the garbage collection problem. This can be accomplished by sorting an alphanumeric array as a linked list, rearranging the links rather than the items themselves. If one then walks from link to link, one travels through the list in order. Of course most people want to sort *their* data, not data in a form someone else decides they should have collected. And where are the links in File Cabinet? The answer is, although there may be no links connecting the data we have, these links can be easily created.

Suppose the array to be sorted is called N$(I,J) where I = 1,...,NR, J = 1,...,NH. All you need do is create an array R of dimension NR, and set R(I) = I. Now R(I) points to the I-th item on the list. Instead of exchanging the elements N$(I,J) one need only change the values of the pointers R(I). At the end of the sorting process, one can then use the String-Swap routine to move the data into place without any string storage overhead. I actually did this, but found a new source of dissatisfaction. Shell's sort, and Quick sort too for that matter, are not 'stable' sorts. This means that if I sort an address list by last name, and then by state, the names within each state will no longer be in alphabetical order.

Recently I came across an article describing a variant of Quick sort that is stable. It is this algorithm which I will discuss below. The data to be processed must be augmented by a set of links S, rather than with pointers R. To implement this sorting algorithm we start by creating an array S, where S(I) = I + 1 for I = 0,...,NR − 1, and S(NR) = 0. The element S(I) points to the item that comes *after* item I. The initial list item is pointed to by S(0), and so initially is 1. The value 0 in S(NR) indicates that there is nothing following item NR. The list can now be sorted by changing the values in the S array. After the list has been sorted, if the smallest item was the K-th on the original list, then S(0) = K, and S(K) will point to the next smallest item, and so on. The relationship between the S links and the R pointers is given by the algorithm in line 720 in the program below. As you will note, in line 730 we replace the values in array S, which have served their purpose, with the values of the inverse of the function R. These backward pointers will be used in the actual process of rearranging the array N$, without ever using any other string array. (See line 750.)

The code itself is quite opaque, and I can do no more than refer the interested reader to the original paper: B. Cheek, "A Fast and Stable List Sorting Algorithm," *The Australian Computer Journal*, vol. 12, no. 2, May 1980.

There are two misprints in that paper, one trivial, and one not so trivial. In the line corresponding to my lines 575,675 the paper omits the minus sign in front of US (which is called uperstrt in the paper). Cheek also omits taking the absolute value of START: line 700.

Cheek gives timing estimates which show that this algorithm is as good as Quick sort. The 'disadvantage' of requiring the creation of linking fields is, for us, a great advantage, and the fact that it is a stable sort makes me believe it is the proper one to use in any Applesoft application where more than a couple of dozen items need be sorted.

The sample program that illustrates this algorithm has been set up so that it may easily be modified for inclusion as a part of File Cabinet. You may want to change the names of some of the arrays if you use it as a module in another program. The sorting is done in the subroutine at lines 500-810. Lines 500,710, and 800 may be omitted, and line 810 replaced with 810 RETURN. The actual sorting algorithm appears twice, in lines 550-590, where alphabetic data is sorted, and in lines 650-690 where numeric data is sorted. (If your data has embedded blanks you will need both sorts. Try comparing " - 123" " + 123" "123" and " 23 " and see what Applesoft thinks.)

The two sections of code are identical except for the use of the VAL function in the second sort routine, and there are other interesting differences. In line 655 we save the value of VAL(N$(I,S)) as C, and then in the loop starting at line 660, C is compared with new values VAL(N$(I,S)) until I becomes ≤ 0. In line 555, however, we do *not* save the string array N$(I,S), only the current value of I. In the loop starting at line 560 comparisons are made between N$(C,S) and N$(I,S). Thus the index calculation (locating N$(C,S)) is made in each iteration of the loop. This 'bad' programming practice avoids introducing a string variable C$, and so avoids producing garbage.

The rest of the program is included just for demonstration purposes. It creates a random list of 50 two-letter names and one-digit addresses. Sorting this list, first by address, and then by name, will demonstrate the speed and stability of the sorting algorithm. The timing estimate is just that, an estimate of the running time. I added 10 seconds for psychological reasons. Note that, as with Quick Sort, it is possible for the sort to take much longer than average. In particular, if the data is already sorted, the running time will be much worse than average. If you fear that this will happen, sort first on some other key to 'randomize' the data before sorting on the key of interest. This will bring the sorting time down to only twice the expected value.

Norman Herzberg is a professional mathematician who has been interested in computing and computers since his undergraduate days at Columbia College. At that time he was introduced to an I.B.M. "computer" that was programmed via a plug board. About 18 months ago he gave up his TI 59 calculator for an Apple, to see what it could do. He invites any and all readers with similar interests to contact him through the SOURCE CL1279.

MICRO

Expanding the Superboard

Build your own expansion board for the OSI Superboard including VIAs, PIAs, a sound chip, and a number of other possibilities.

Jack McDonald
Mews Cottage, Pond Lane
Clanfield, Portsmouth
PO8 ORG, England

Many articles and programs have appeared in computer magazines on AIM, SYM, and KIM systems with their VIAs and PIAs, leaving Superboard out in the cold! To correct this unbalanced situation I built an expansion unit for Superboard, consisting of PIAs and VIAs, with room for the addition of a 'Sound Chip' and further expansion if required.

In an attempt to standardize, I chose the decoded addresses closest to the SYM, because the Superboard doesn't seem to use E000-EFFF. In table 1 you can see that for VIA 1, the SYM's Axxx is equivalent to our Exxx. For example, A00B on the SYM is E00B. This makes it fairly easy to transfer, as the PIA/VIA registers are accessed by the two least significant hex digits (00-FF). On the prototype only a few of the chips were installed — most AIM/SYM applications use two VIAs at most. However, this circuit provides for decoding two PIAs, three 6522 VIAs, a 6532 VIA, a sound chip, and a spare. The 6520's could be used to select other devices. How about an alternative character ROM, or even characters in RAM? I'll leave that to you.

Connection to S/B is via a 40 pin to 40 pin jumper lead. A separate 5V feed to the VIA board is preferred but pin 11 of J1 could be used. Make sure that the Data Bus buffers are fitted to your S/B (U6,U7).

Figure 1

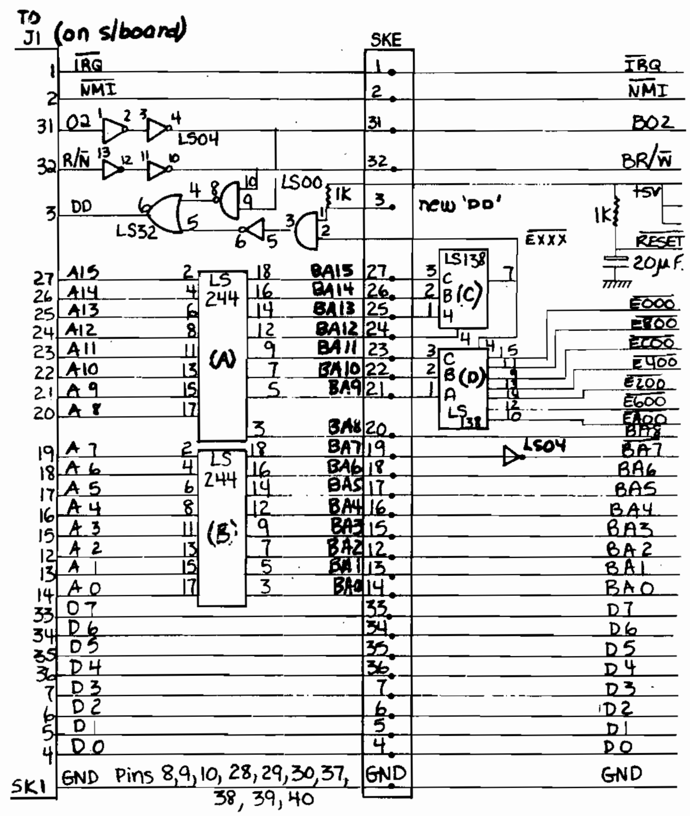


***Figure 2:* Simple D/A - A/D Voltmeter**

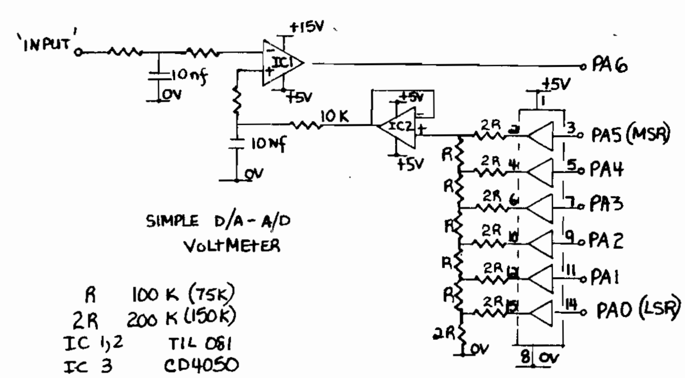


Table 1

6502

SYM/AIM	S/Bd	
Ab00	Eb00	ORB (PB0-PB7)
Ab01	Eb01	ORA (PA0-PA7)
Ab02	Eb02	DDR B
Ab03	Eb03	DDR A
Ab04	Eb04	T1L-L/T1C-L
Ab05	Eb05	T1C-H
Ab06	Eb06	T1L-L
Ab07	Eb07	T1L-H
Ab08	Eb08	T2L-1/T2C-L
Ab09	Eb09	T2C-H
Ab0A	Eb0A	SR
Ab0B	Eb0B	ACR
Ab0C	Eb0C	PCR(CA1,CA2, CB1,CB2)
Ab0D	Eb0D	IFR
Ab0E	Eb0E	AER
Ab0F	Eb0F	ORA

b is 0 for VIA 1
b is 8 for VIA 2
b is C for VIA 3 (SYM only)

6532

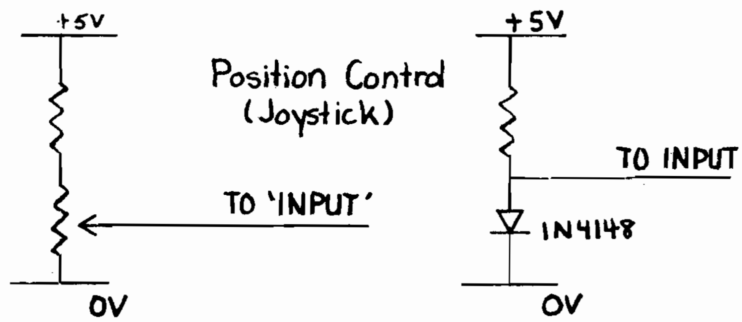
SYM/AIM	S/Bd	
A400	E400	ORA
A401	E401	DDRA
A402	E402	ORB
A403	E403	DDRB
A404	E404	W-edge detect, R-timer
A405	E405	W-edge detect, R-int flags
A406	E406	W-edge detect, R-timer
A407	E407	W-edge detect, R-int flags
A41C	E41C	TIMER-1T
A41D	E41D	TIMER-8T
A41E	E41E	TIMER-64T
A41F	E41F	TIMER-1024T

Two 74LS244's were used to buffer the 16 address lines, 4/6ths of a 74LS04 buffer the phase two and R/W. Alternatively three 74LS367's could be used. The 74LS32 plus 1/6 74LS04 and 1/2 74LS00 are needed to provide the necessary 'DD' signal to S/B and allow expansion via SKE to a 610 board or whatever. The new 'DD' input was required since open collector OR gates don't exist. Initially three O/C inverters were used.

The 74LS138(A) enables 74LS138(D) for addresses E000-EFFF and (D) decodes in 256-byte segments.

If power-on reset is used, all resets should be connected in parallel. Individual resets with switches can be used with an associated extra wiring "Jungle" or use the outputs of a PIA as a software reset.

Figure 3: A/D Interfaces



Listing 1

```
;* GENERAL SUCCESSIVE APPROXIMATION
;*  TO DRIVE DVM
;*
;* BY JACK MCDONALD
;*
;
;PIA = $EXXX
;
;XX, YY, AND AA ARE DETERMINED BY USER
;
INIT   LDA #$3F
       STA PIA           ;SET 6 OUTPUTS & 2 INPUTS
       LDA #$04
       STA PIA+1         ;ACCESS IORA
;
START  LDA #$00
       STA YY            ;CLEAR LOCATION YY
       LDA #$40
       STA XX            ;SET MSB IN LOC XX (BIT 5 FOR OUR D/A)
       LDX #$07          ;LOAD COUNTER (6 + 1, SINCE WE DEX FIRST)
LOOP   DEX
       BEQ FIN           ;DONE?
       LDA XX
       STA PIA           ;SET MSB ON D/A
;
;DE-GLITCH TIME DELAY
;
DEGL   LDA #$00
       STA AA
DLOOP  DEC AA            ;1ST DEC AA CONTAINS $FF
       BNE DLOOP         ;DELAY FOR $FF X 2 MICROSEC
;
       LDA PIA           ;READ PIA INPUT
       AND #$80          ;ONLY BIT 6 (A/D OUTPUT)
       BNE SAVE
       BEQ NEXT
SAVE   ADC YY            ;STORE RESULT AFTER ADDITION
       STA YY            ;YY HAS TOTAL SO FAR
NEXT   ROR XX            ;NEXT 'MSB'
       JMP LOOP
FIN    LDA YY            ;TAKE FINAL TOTAL
       JSR CRT           ;PRINT IT ON CRT
       JMP START         ;START AGAIN
```

Figure 4: A/D Interfaces

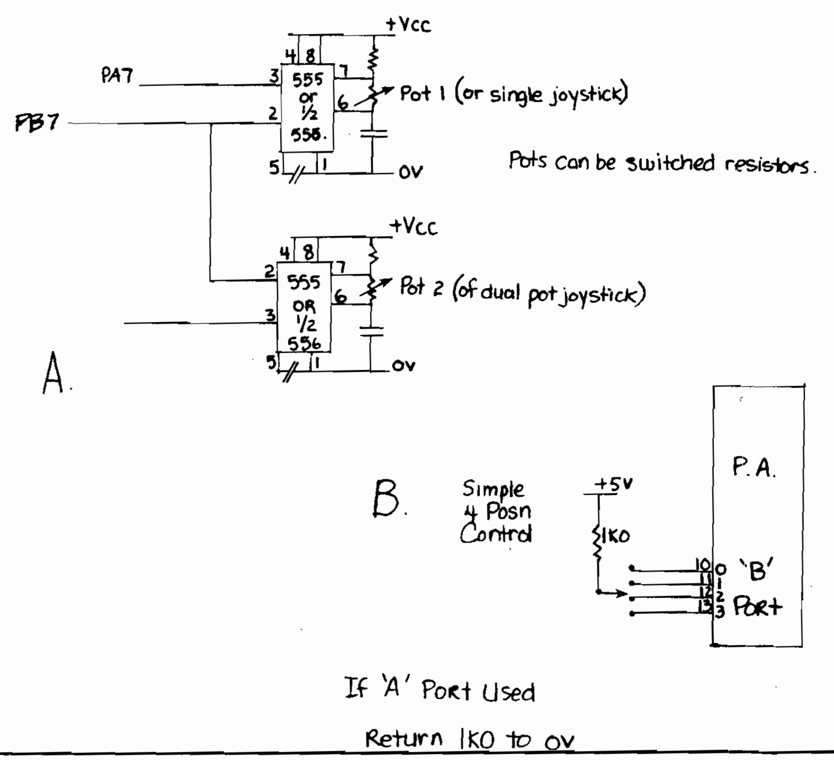


Figure 5

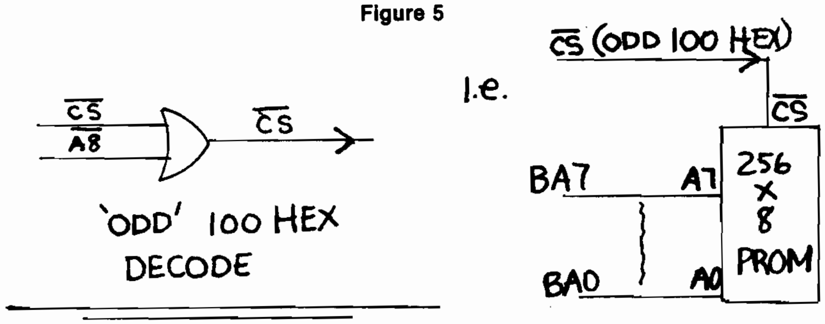
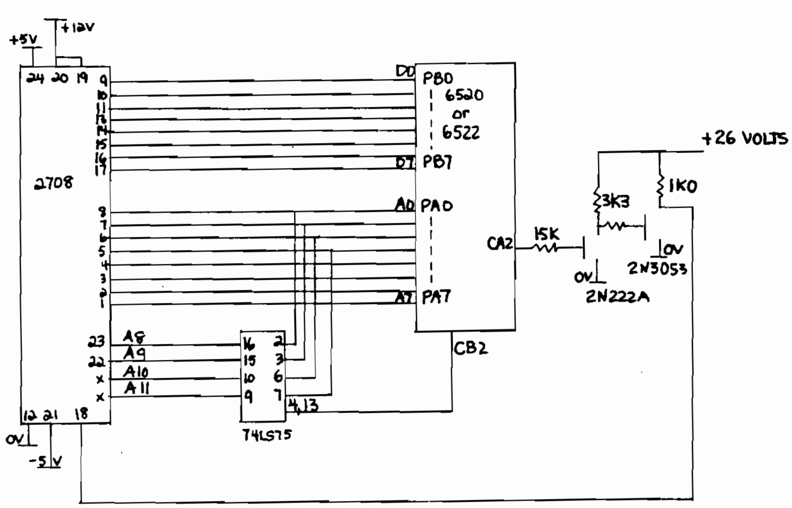


Figure 6: EPROM/PROM Programmer



The expansion board was constructed on 'VERO' DIP Board and the 40 pin sockets were straddled across the two supply rails (see figure 1). In the USA 'VECTOR' is a near equivalent. To make output connections, 16-pin Dil sockets (use only the 8 pins connected to the PA/PB outputs) 'VECTOR' type VCT-4493-1 may be suitable. Wire-wrap/wire pen or Rats Nest can be used with wire-wrap allowing tidier modifications.

Figure 2 shows a simple A/D-D/A converter, which performs the function of a 6-bit digital voltmeter. "De-glitching" has not been included — a software delay is used instead.

Figure 3 gives two very primitive input interfaces for the DVM. Listing 1 is a successive approximation program to drive the DVM. Improvements to the circuits and program are possible at the expense of simplicity, but the circuit is adequate for simple control applications and learning about D/A's in general.

The resistor values should be kept between 150K and 330K for the 2R, to minimize the effect of the 4050 "on" resistance (about 1K). R is two paralleled 2R's.

Figure 4(a) is a simple method of implementing joystick controls. The variable resistor in the timing circuit of the 555/556 alters the duration of the output pulse. This pulse is detected by the PA7 pin of a 6532 VIA in its interrupt mode. The 555 is triggered by the low transition of PB0 on the same device. The software on interrupt reads the timer; then, by using a 'dead zone' and a no-action (or stop), can be defined [i.e., 0-130(up), 131-140(stop), 141-255 (down)]. Thus the stop position is not too critical to locate manually.

Figure 4(b) shows an ultra-simple switch position detector. By reading the four bits, one of four possibilities is detected, i.e. LDA PIA, and #$0F — value left in A is the switch number.

Figure 5 indicates the additional decoding for 100 hex 'boundaries' — 256-byte PROMS, etc.

Figure 6 is a commonly used "EPROM programmer" of the on-board variety. The address is first latched into PA0-PA7 and the data byte to be programmed is latched into PB0-PB7. Finally the programming pulse is applied via CA2 for the recommended time. As the 8 bits (PA) will only address 256 bytes, a 74LS75 is used as an address extender. If PA0-PA4 are initially zero then clocking the '75 via CB2 clears the high-address bits (A8-A11). After 256 bytes, latch a one on PA0, clock CB2, and A8 on the 2708 is 'on.' Then do the next 256. Listing 2 gives the necessary steps.

Figure 7 indicates how to hang on a "sound chip." See manufacturer's data sheets for programming information.

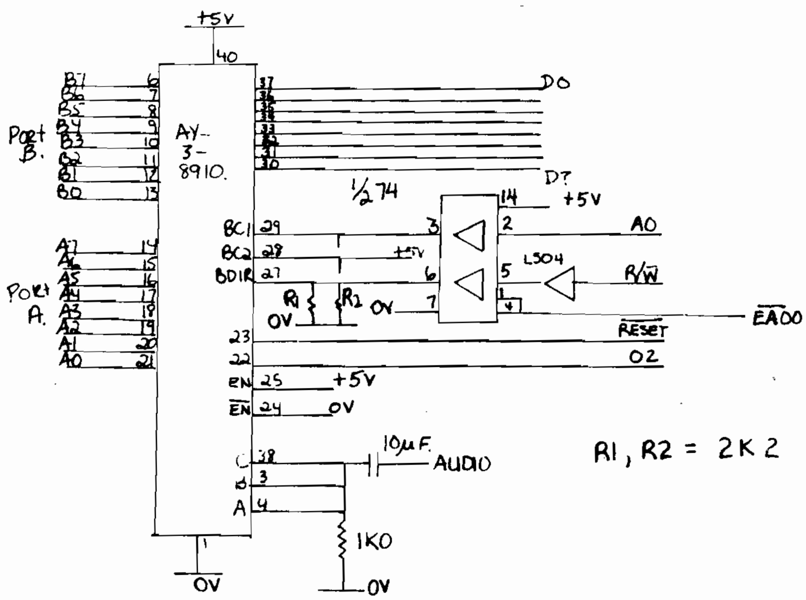
The final circuit of figure 8 is for a Paper Tape reader. The unit used was an old (free) "Computer Mechanisms Corp" ratchet relay type, with long contact fingers sensing holes in the tape. These contacts are connected to PA0-PA7 of a PIA. The relay is driven by a small CMOS FET via the CA2 output. The listing given reads 256 bytes but can be altered to increase this. The reader can also read 5-bit tapes. It is only necessary to mask off the high 3 bits in the main program — LDA, PIA, and #$1F — this should appease the TTY'ers. In 8-bit form it is ideal for disassembling tapes produced from ROM/PROM, etc., since keyboard and LED displays are painfully slow!

For more information refer to MICRO (7:17), (11:31), (13:41), (17:27), (17:55), and Sybex's *6502 Applications*.

Listing 2

```
                ;* EPROMMER
                ;*
                ;* BY MCDONALD
                ;*
                LOCN   EPZ $00
                BUFF   EQU $500
                PIA    EQU $EC00
                ;
                       ORG $300

                ;
0300 A900       START  LDA #$00
0302 8D01EC            STA PIA+1       ;DDRA
0305 A9FF              LDA #$FF
0307 8D00EC            STA PIA         ;ALL A OUPUTS
030A A904              LDA #$04
030C 8D00EC            STA PIA
030F A900              LDA #$00
0311 8D02EC            STA PIA+2       ;DDRB
0314 A9FF              LDA #$FF
0316 8D03EC            STA PIA+3       ;ALL B OUTPUTS
0319 A904              LDA #$04
031B 8D02EC            STA PIA+2
031E A900              LDA #$00
0320 8D00EC            STA PIA         ;PA'S TO ZERO
0323 8D02EC            STA PIA+2       ;PB'S TO ZERO
0326            ;
0326            ;25 MICROSEC DELAY SUBROUTINE
0326            ;
0326 A9FC       LODLY  LDA #$FC
0328 8500              STA LOCN
032A C600       LOOP   DEC LOCN
032C D0FC              BNE LOOP
032E 60                RTS
032F            ;
032F            ;1 MILLISEC DELAY SUBROUTINE
032F            ;
032F A904       HIDLY  LDA #$04
0331 8501              STA LOCN+1
0333 202603            JSR LODLY
0336 C601              DEC LOCN+1
0338 F003              BEQ FINI
033A 202603            JSR LODLY
033D 60         FINI   RTS
033E            ;
033E A900              LDA #$00
0340 8502              STA LOCN+2
0342 A964              LDA #$64
0344 8503              STA LOCN+3
0346 A000       PROG   LDY #$00
0348 B90005     MOV    LDA BUFF,Y
034B 8D02EC            STA PIA+2       ;DATA
034E 202603            JSR LODLY
0351 A93C              LDA #$3C
0353 8D01EC            STA PIA+1       ;CA2 ON
0356 202F03            JSR HIDLY       ;FOR 1 MILLISEC
0359 A934              LDA #$34
035B 8D01EC            STA PIA+1       ;CA2 OFF
035E 202603            JSR LODLY
0361 C8                INY             ;INC RAM/ROM ADDRESS
0362 98                TYA
0363 8D00EC            STA PIA         ;LOW ADDRESS BITS
0366 A8                TAY             ;TAY TO TEST Z FLAG
0367 D0DF              BNE MOV         ;256 NOT DONE?
0369            ;
0369 E602       ADINC  INC LOCN+2
036B C904              CMP #$04
036D F014              BEQ HUND
036F A502              LDA LOCN+2
0371 8D00EC            STA PIA
0374 A93C              LDA #$3C
0376 8D03EC            STA PIA+3
0379 A934              LDA #$34        ;INC ADDRESS EXTENSION
037B 8D03EC            STA PIA+3
037E A900              LDA #$00
0380 8D00EC            STA PIA         ;RESET LOW ADDRESS BITS
0383 C603       HUND   DEC LOCN+3
0385 D0BF              BNE PROG        ;100 TIMES YET?
0387            ;JMP EXIT TO MONITOR?
```

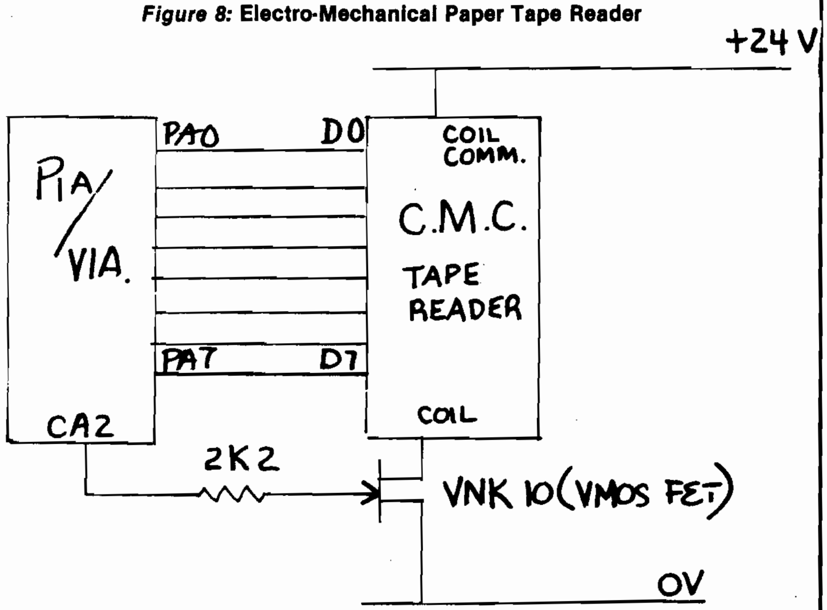
Figure 7: Sound Generator Interface



BDIR	BC1	Address Hex	Address Dec	Function
0	0	EA00	59904	(READ) INACTIVE
0	1	EA01	59905	READ FROM DSE
1	0	EA00	59904	WRITE DATA TO PSG
1	1	EA01	59905	(WRITE) LATCH ADDRESS

Example: POKE 59905,7 POKE 59904,130 places (DEC) 130 in register 7.

Figure 8: Electro-Mechanical Paper Tape Reader



Listing 3

```
                       ;* JOYSTICK ROUTINE
                       ;*
                       ;* BY JACK MCDONALD
                       ;*
                       ;6532 ADDRESSED AT $E400
                       ;
                              ORG $E480

                       ;
E480 A980                     LDA #$80
E482 8D03E4                   STA $E403        ;PB7 IS OUTPUT
E485 8D02E4                   STA $E402        ;PB7 SET HIGH
E488 AD05E4                   LDA $E405        ;CLR INT FLAG
E48B 8D06E4                   STA $E406        ;ENABLE PAY IRG (DATA =
                                                "DON'T CARE")
E48E A90F                     LDA #$0F         ;16 MICROSEC
E490 8D1EE4                   STA $E41E        ;TIMES 64
E493 A900                     LDA #$00
E495 8D02E4                   STA $E402        ;RESET 555 TIMER
E498 AD05E4            WAIT   LDA $E405
E49B 2940                     AND #$40         ;IRQ YET?
E49D F0F9                     BEQ WAIT         ;NO
E49F AD04E4                   LDA $E404        ;YES, READ TIME VALUE
E4A2 40                       RTI              ;RETURN TO MAIN PROG.
                                                WITH TIME VALUE IN ACC
```

Listing 4

```
                       ;* PAPER TAPE READER
                       ;*
                       ;* BY JACK MCDONALD
                       ;*
                       PIA    EQU $EA00
                       BUFF   EQU $03FF
                       ;
                              ORG $300

                       ;
0300 A000                     LDY #$00
0302 201803            READ   JSR STEP
0305 AD00EA                   LDA PIA
0308 C9FF                     CMP #$FF         ;START BYTE?
030A D0F6                     BNE READ         ;NO
030C 99FF03            LOOP   STA BUFF,Y       ;YES...START READING
030F 201803                   JSR STEP
0312 AD00EA                   LDA PIA
0315 C8                       INY
0316 D0F4                     BNE LOOP
0318                   ;EXIT HERE, AS 256 BYTES HAVE BEEN READ
0318 A90E              STEP   LDA #$0E
031A 8D0CEA                   STA PIA+$0C      ;TURN CA2 ON
031D A920                     LDA #$20
031F 8DFD03                   STA $03FD        ;DELAY HIGH BYTE
0322 A9FF              DELAY  LDA #$FF
0324 8DFC03                   STA $03FC        ;DELAY LOW BYTE
0327 CEFC03                   DEC $03FC
032A D0F6                     BNE DELAY
032C CEFD03                   DEC $03FD
032F D0F1                     BNE DELAY
0331 A90C                     LDA #$0C
0333 8D0CEA                   STA PIA+$0C      ;TURN CA2 OFF
0336 60                       RTS
```

MICRO Software Catalog

Mike Rowe
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Name: **SEGS**
System: OSI
Language: OS65D
Hardware: Disk
Description: Adds segmentation commands to BASIC. Allows segment calls (like GOSUB's) to subroutines stored on disk. By nesting calls, *large* programs may be written and will run in memory. Write for more information.
Price: $25.00
Available: Universal Systems
2020 W. County Rd. B
Minneapolis, MN 55113

Name: **Fast Facts**
System: Apple II & Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Disk 3.2 or 3.3, line printer desirable
Description: This selection of programs was created and designed by a Certified Financial Planner for quick analysis of the personal investment planning needs of his clients. It was professionally programmed for efficient and accurate operation. Fast Facts operates very easily with single key program selection and printing commands. In many cases the entire planning sequence is completed in less than 60 seconds. Specific program objects are divided into six systematic program fields. They are: 1) planning for retirement, 2) college financing for the kids, 3) diversifying your investments, 4) the result of inflation in devaluing your earnings, 5) costs of borrowing money and loan balance at any point in time, 6) investment calculations for compounding and future values. These programs were planned with care to allow you to change input data and in many cases identify erroneous entry values. Their primary value rests with their speed and ease of operation with no need to learn special control characters.
Copies: Version 1.1 just released
Price: $95.00 includes disk and instructions
Author: Monte C. Fremouw
Available: Richard Lorance CFP
c/o Richard Lorance and Associates, Ltd.
3336 N. 32nd Street, Suite 102
Phoenix, AZ 85018

Name: **SYM-FORTH 1.0**
System: SYM-1
Memory: 16K
Language: 8K machine language and FORTH
Hardware: Serial terminal and RAE ROMS
Description: SYM-FORTH 1.0 is a faithful implementation of the fig-FORTH model with the following additional features: unique input line editor; built-in 6502 FORTH assembler; dual cassette interface; FIG-style screen editor; upgrade to 79-STANDARD available through subscription to newsletter.
Copies: 50
Price: $135 US/$155 Canada - cassette version includes 74-page user guide, 100-page source listing, and object on cassette.
$150 US/$175 Canada - disk version for dual HDE mini disk system, as above but supplied on two mini floppies. System boots with 79-Standard installed.
Author: John W. Brown
Available: Saturn Software Limited
8246 116A St.
Delta, BC., V4C 5Y9, Canada

Name: **Pegasus**
System: UCSD Pascal operating systems
Memory: 48K and the Pascal Language Card
Language: UCSD Pascal
Hardware: Apple II, Language Card, CRT.
Description: This is a Data Base Management System. You can create, define, manipulate, print, list, write to disk, view and generally use data files. It is extremely user-oriented, especially for the novice user. It is menu driven.
Price: $195.00 MSRP includes program diskette, technical manual, and 'cookbook.'
Author: Shakti Systems Inc.
Available: Powersoft, Inc.
POB 157
Pitman, NJ 08071

Name: **6502 C Cross-compiler**
System: UNIX/V7, UNIX/V6 or Idris, RT-11, RSTS/E, RSX-11, VAX/VMS
Memory: 28K
Language: C
Hardware: PDP-11 series, LSI-11 series, VAX series
Description: This product is a C cross-compiler running on any of the above-mentioned hardware/software systems. It generates symbolic assembly language for the 6502 microprocessor. The full C language, as described by Kernighan and Ritchie's *The C Programming Language*, is supported except for three minor features. This product complements the existing line of C compilers and cross-compilers from Whitesmiths, Ltd, of New York.
Price: $1600 plus media charge ($30 for floppies, $50 for magtape) includes documentation and binary license for use on a single host CPU
Author: Staff
Available: Advanced Digital Products, Inc.
1701 Twenty-first Ave., S.
Nashville, TN 37212

Name: **Home Energy Survey**
System: OSI-4P and PET-2000
Memory: 24K (OSI), 8K/16K/32K (PET)
Language: BASIC
Hardware: Minifloppy (OSI)
Cassette (PET)
Description: This program calculates the savings a home owner will achieve by adding storm windows, changing thermostat settings, caulking, weather-stripping, adding ceiling insulation, and adding floor insulation. The program is valid for the 48 contiguous states and for the following heating and cooling fuels: oil, natural gas, electricity, wood, propane (LPG), and coal. The user inputs city, state, fuel cost, window area, floor area, thermostat settings, ceiling and floor R values.
Price: $15.95
Author: David E. Pitts
Available: David E. Pitts
16011 Stonehaven Dr.
Houston, TX 77059

Name: **C.O.R.P. (Combined Operations Re-entrant Programming Data-Base Management System)**
System: Apple II
Memory: 48K
Language: Applesoft BASIC
Hardware: 2 disk drives (DOS 3.3), Applesoft in ROM, video monitor, optional printer
Description: C.O.R.P. is a program generator that writes complete data-entry and print programs in Applesoft BASIC. These programs are written on a standard DOS 3.3 disk and may be modified by the user. The system includes a sort, update and copy facility along with the ability to modify important system functions. The generated programs utilize keyed random access for fast record retrieval. A complete diagnostic package is also included.
Price: $189.95 includes master and diagnositc disks/manual
Author: Alexander Maromaty
Available: Maromaty & Scotto Software Corp.
P.O. Box 610
Floral Park, NY 11001

Name: **GRAFPAK, TIGR**
System: Apple II
Memory: 32K minimum
Language: BASIC and machine
Hardware: Disk II and Integral Data IDS 560 or 460
Description: Provides 1 or 2 × horizontal and 1 to 3 × vertical reproduction of either Hi-Res page, and 1 to 3 × vertical reproduction of both pages side by side. 3 × horizontal and 4 or 5 × vertical reproduction on IDS 560, only. Normal/inverse inking and indentation in inches are user-specified. Compatible with most I/O cards. Extremely simple to use. Versions available for other printers!
Price: $39.95 for 460 version (plus 5.5% tax in Ohio)
$49.95 for 560 version
Author: Robert Rennard
Available: SmartWare
2281 Cobble Stone Court
Dayton, Ohio 45431

Name: **Job Control System**
System: Apple II
Memory: 48K
Language: Pascal
Hardware: Three disk drives and a 132-column printer capable of performing a form feed.
Description: Computer-assisted job control for small-to-medium-size companies in manufacturing, construction and service industries. This system provides management with reliable measures of productivity furnishing up-to-the-minute job status data for determining the real cost of producing a product or providing a service. Several valuable reports including job listing, job cost summaries, detailed individual job reports, and work-in-process reports give profit/loss values and variances so that job estimates and work standards can be fine-tuned.
Price: $750.00
Author: Shop Controls Inc.
Available: High Technology Software Products Inc.
P.O. Box 14665
8001 N. Classen Blvd.
Oklahoma City, OK 73113

Name: **FBASIC Compiler**
System: All Ohio Scientific 8" Disk Systems (OS65D Operating System)
Memory: 48K
Language: FBASIC
Hardware: OSI 8" disk systems
Description: Super-fast BASIC compiler. Compiles an integer-subset of OSI/Microsoft BASIC into native 6502 machine code. Features user-definable array locations, WHILE loops, GOTOs and GOSUBs to absolute addresses, direct access to 6502 registers, and much more. FBASIC is fully diskbased, and is capable of producing programs larger than available memory.
Price: $155.00 ppd. includes 8" disk with compiler, many example programs, and user manual.
Author: Richard Foulk
Available: Pegasus Software
P.O. Box 10014
Honolulu, HI 96816

Name: **0-3. Option Strategy Charts**
System: PET
Memory: 8K
Language: BASIC
Hardware: PET/CBM
Description: Charts are plotted for two assumed situations of option strategies of puts and calls and their combinations. The plot of strategy values for a series of underlying stock prices permit comparison of the assumptions.
Price: $15.00
Author: Claud E. Cleeton
Available: Claud E. Cleeton
122-109th Ave., S.E.
Bellevue, WA 98004

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

MICRO 6502 Bibliography: Part XXXV

1025. MICRO No. 32 (January, 1981)

Davis, Robert V., "Print Using," pg. 6.
Print Using for the OSI C1P.

Finkbeiner, Tim, "List Disable," pg. 6.
List disable for OSI ROM BASIC.

Young, George, "Keyboard Encoding," pg. 7-14.
Add a keypad or keyboard to your 6502 micro.

Childress, J.D., "A Better Apple Search/Change," pg. 17-19.
An improved version of the Search/Change program for the Apple.

Bassman, Mike, "Vectors and the Challenger 1P, pg. 21.
A tutorial on Vectors and how to use them on the OSI C1P.

Kolbe, Werner, "PET Symbolic Disassembler," pg. 23-26.
This disassembler generates labels and symbols for the critical addresses.

Flynn, Christopher J., "AIM 65 File Operations," pg. 29-32.
The third part of a series on AIM 65 file processing.

Tenny, Ralph, "Full Disassembly Listing on Small Systems," pg. 37-39.
A utility for the KIM or other small system.

Green, Len "Bridge Trainer," pg. 41-46.
A program for the SYM-1.

Wright, Loren, "PET Vet," pg. 51.
Notes on the update for VIC, finding BASIC variables, etc.

Neiburger, E.J., "Make a Clear Plastic Cover for Your Apple," pg. 53.
A constructional, how-to article related to the Apple.

Little, Gary B., "Searching String Arrays," pg. 57-59.
An Apple matching language program to rapidly search a large string array.

DeJong, Marvin L., "Interfacing the 6522 Versatile Interface Adapter," pg. 65-72.
How to implement the 6522 on your 6502 system.

Cain, Les, "Fun With OSI," pg. 75-76.
A checker game using C1P graphics.

Anon., "Ohio Scientific's Small Systems Journal," pg. 82-86.
Memory Tests, Bit Rotation Test, Pseudo-Random Test, etc.

Staff, "MICRO Software Catalog: XXVIII," pg. 87.
Fourteen software items for 6502 micros.

Dial, Wm. R., "6502 Bibliography: Part XXVIII," pg. 90-94.
Over 150 additional references to the extensive 6502 literature.

1026. Apple Bits 3, No. 1 (January, 1981)

Kovalik, Dan, "Taking the Mystery and Magic Out of Machine Language," pg. 3-4.
This month's tutorial on machine language includes a routine called Directory Compress, eliminating the holes left in the directory by deleted files.

1027. Compute! 3, No. 1, Issue 8 (January, 1981)

Butterfield, Jim, "Financial Fuzzies," pg. 22.
A numbers formatting routine for the PET.

Deemer, B.J., "Spend Time, Save Money!", pg. 22-23.
Hints on using the PET cassette.

Semancik, Susan, "Micros with the Handicapped," pg. 26-27.
Discussion of techniques for the handicapped (PET).

Albrecht, Bob and Firedrake, George, "The Mysterious and Unpredictable RND: Part 1."
A tutorial on the PET use of the RND function.

Pratto, R., "Cursor Classifications Revisited," pg. 38.
A system of classification for PET programs.

Butterfield, Jim, "Odds and Ends Re PET Cassette Tape."
A collection of PET cassette-related hints and notes.

Falkner, Keith, "Load PET Program Tapes into the Apple II," pg. 50-59.
A "PET Loader" for the Apple.

DeJong, Marvin L., "Programming and Interfacing the Apple, with Experiments," pg. 61-65.
A hardware and experimental article related to the Apple.

Crawford, Chris, "Player-Missile Graphics with the Atari Personal Computer System," pg. 66-71.
An Atari graphics tutorial.

Baker, Al, "The Fluid Brush," pg. 72-73.
A joystick-graphics program for the Atari.

Lindsay, Len, "Atari Disk Menu," pg. 74-77.
An Atari tutorial on disk menus.

Bruun, James L., "Using the Atari Console Switches," pg. 77.
Some hints on using those switches by the Atari keyboard.

Beseke, Roger, "The Atari Disk Operating System," pg. 78-79.
A quick and brief description of what you can do with Atari DOS.

White, Jerry, "Atari Sounds Tutorial," pg. 79-80.
Discover some of the sounds of your Atari.

Gordon, Thomas G., "A 6502 Disassembler," pg. 81-82.
A disassembler for the OSI micros.

Berger, T.R., "A Small Operating System: OS65D — The Kernel: Part 1," pg. 84-91.
A tutorial on the OS65D system for OSI micros.

Stanford, Charles L., "OSI C1P Fast Screen Clears Revisited," pg. 91.
Techniques for screen clearing on OSI micros.

Mansfield, Richard, "The Screen Squeeze Fix for CBM 8000," pg. 92-93.
How to adapt programs to the new CBM 80-column screen micros.

Herman, Harvey B., "Hooray for SYS," pg. 96-100.
A tutorial for the SYS command for PETs, with three listings.

Butterfield, Jim, "Scanning the Stack," pg. 102-106.
An instructional article on PET's machine language.

Isaacson, Dan, "Detecting Loading Problems and Correcting Alignment On Your PET," pg. 114-115.
Hints on improving the reliability of the PET cassette loading procedure.

Peterson, T.M., "Spooling for PET with 2040 Disk Drive," pg. 118.
Save to disk now, print later.

Levinson, V.M.D., "Variable Dump for New ROM PETs," pg. 118-120.
A routine to list all defined PET BASIC program variables and give current values.

Wuchter, Earl H., "The 32K Bug," pg. 120.
A special procedure for 32K PETs to avoid screen boundary problems.

Hudson, Arthur C., "An 'Ideal' Machine Language Save for the PET," pg. 121-122.
A procedure for the PET.

Huckell, Gary R., "PET/CBM IEEE Bus Error," pg. 124-125.
An error in the PET IEEE I/O routine and a fix.

Rehnke, Eric, "The Single Board 6502: High-Speed Data Transfer," pg. 126-130.
Software which dumps object code from either the AIM, SYM, or 6522-equipped Apple to a KIM board.

Hooper, Philip K., "Caveat Interrupter or Placating a Rebellious KIM without Sacrificing RAM," pg. 132.
An experiment with a runaway KIM.

Chamberlin, Hal, "Expanding KIM-Style 6502 Single Board Computers," pg. 138-142.
Part 1 of a series on expanding small micros.

1028. Call -Apple 4, No. 1 (January, 1981)

Goez, Eric E., "Real Variable Study," pg. 8-23.
About numbers, scientific notation, several listings, etc. for the Apple.

Reynolds, Lee, "Applesoft Sub-String Search Function," pg. 26-30.
A utility for Apple users, called Ampersand-Instr. Function.

Zant, Robert F., "Data Storage Techniques," pg. 35-38.
An article to assist the understanding of files.

Anon., "How to Enter Call -Apple Assembly Language Listings," pg. 39.
A short instructional article for the Apple assembly language.

Wiggington, Randy, "Fast Garbage Collection," pg. 40-45.
Speed up your Apple with this utility.

Ender, Philip B., "Pascal Zap," pg. 47-49.
A utility allowing access to any block on the disk, including the directory and deleted files.

Lingwood, David A., "Adding Lines to Running Applesoft," pg. 51-53.
This assembler program can be instructed to replace any REM statement with program code in a running BASIC Applesoft program.

Anon., "Write -Apple," pg. 55.
Some notes on the fix for the use of Applewriter with the Paymar Lower Case chip; also a fix for a bug on the DOS 3.3 master disk.

Horsfall, Richard C., "Bsaving and Bloading Arrays in Integer BASIC and Applesoft," pg. 58-61.
Two utility listings for the Apple.

1029. Peek(65) 2, No. 1 (January, 1981)

McGuire, Dick, "Tech Notes," pg. 2-5.
Fix for packer; user defined input; cassette corner; US error; right justification, etc.

Wallis, Terry L., "OS65U Port #5 to Port #8 Modification," pg. 10.
An assembly source listing to modify OS65U so that a Port #5 command sends output to Port #8.

Grittner, Kurt, "Print Enhancements of 65D V3.0," pg. 15-18.
A formatting program for numbers and dollars/cents on OSI systems.

1030. Apple Gram 3, No. 1 (January, 1981)

Matzinger, Bob, "Binary Manipulation," pg. 4-7.
How the computer handles numbers.

Meador, Lee, "MON I/NOMON I Flag," pg. 8.
A discussion of the MON function.

Carpenter, Chuck, "Apple Blossoms — For Newcomers," pg. 16-17.
A short introduction to assembly and machine languages.

Hatcher, Rich, "Hello Program Improvement," pg. 20-21.
A Hello program for the Apple disk.

1031. Appleseed 2, No. 5 (January, 1981)

Pump, Mark, "Apple II DOS Internals," pg. 4-6.
DOS memory/disk addresses cross reference for the Apple.

1032. Apple Assembly Line 1, Issue 4 (January, 1981)

Sander-Cederlof, Bob, "A Calculated GOSUB for Applesoft," pg. 8.
Restore this useful function to Applesoft.

Sander-Cederlof, Bob, "How to Move Memory in the Assembler," pg. 2-6.
A tutorial on moving data with the S-C Assembler, with two machine language listings.

Meador, Lee, "Putting COPY in S-C Assembler II," pg. 9.
How to install this function.

Laumer, Mike, "EDIT Command for S-C Assembler II," pg. 10-11.
Discussion and listing for a new feature for the Assembler.

1033. Creative Computing 7, No. 1 (January, 1981)

Fee, Peter, "No PET Peeves," pg. 24-25.
The VIC-20 is a new Commodore computer based on the 6502 and selling under $300.

Nasman, Leonard, "Atari Music Composer Cartridge," pg. 26.
A music system for the Atari.

Kruse, Richard M., "An Atari Library of Sound," pg. 74-78.
A series of listings for Atari sound routines from which you can select for adding that ringing telephone, etc. to your program.

Miller, David, "Apple-Sketch," pg. 110-118.
An instructional article on Hi-Res graphics, including a program for the Apple to make things easier.

Lubar, David, "Apple II Lo-Res Shape Tables," pg. 120-124.
Simplify moving Lo-Res figures around by using shape tables as is common with the Apple Hi-Res graphics.

Hitchcock, Paul, "Hi-Res Text for the Apple," pg. 126-129.
Embellish that Hi-Res Apple display with text. Label the axes of graphs, etc.

Bobhop, Bish, "Lit'l Red Bug," pg. 130-131.
A car-driving game for the Apple.

Tunbo, David, "The Digital Couch," pg. 132-133.
Turn your Apple into a psychiatrist.

Piele, Donald T., "How to Solve It — With the Computer," pg. 142-151.
Part Four on probability with a number of problems and Apple solutions.

Yob, Gregory, "Personal Electronic Transactions," pg. 156-163.
A printer list program, discussion of 6502 machine language (Monjana/1) and the big keyboard. Also a Hangmath for PET.

Carpenter, Chuck, "Apple-Cart," pg. 170-175.
Discussion of DOS 3.3. An Applesoft BUG in the GET/Val function.

Blank, George, "Outpost: Atari," pg. 176-179.
Discussion of bytes, nibbles and bits in digital counting. Notes on Atari graphics, etc.

1034. Abacus II 3, Issue 1 (January, 1981)

Davis, James P., "Printer On Says-A-Me (Expanded)," pg. 3-4.
Many new features added in this new listing of the Apple/Trencom 200/AII-g printer program.

Morris, Gary, "Apple II Disk Soft Sectoring," pg. 5-6.
Discussion of diskette nibblizing and self sync.

Davis, James P., "Update Your DOS 3.3," pg. 7-8.
An update for a DOS 3.3 bug and some further improvements, especially for those running language cards or turnkey systems.

Smith, Paul D., "Convert Feet, and 1/16ths to Decimal Feet and Back Again," pg. 9.
A subroutine for the Apple useful to architects, engineers, contractors, etc.

Morris, Gary, "Format of Directory Information for Apple Pascal," pg. 10-12.
An instructional article for Pascal users.

Robbins, Greg, "Bload Finder," pg. 13.
A program to print the starting address and length in hex of a binary program immediately after it is Bloaded.

Anon, "IAC Apnote: Tabbing with Apple Peripherals," pg. 14-16.
A utility for the Apple.

1035. Personal Computing 5, No. 1 (January, 1981)

Jong, Steven, "Word Processing Software Roundup," pg. 26-33.
A review of a number of word processors including several for 6502 systems.

Pritchett, Robert A., "A Pseudo-Numeric Key Pad for the Apple II," pg. 46-47.
An inexpensive substitute for a separate numeric keypad.

Swan, Tom, "Understanding BASIC Language Operations," pg. 68-72.
An introduction to Applesoft, including two utility routines, which remove REM statements from Integer or Applesoft listings.

1036. The Harvest 2, No. 6 (February, 1981)

Anon., "More Pascal," pg. 6-8.
Program Lookit is a primer that will display the Pascal character set on your Apple Hi-Res screen.

1037. Apple/Sass 3, No. 1 (February, 1981)

Burger, Mike, "Text POKE Locator," pg. 8-9.
An Apple program to find the POKE locations on the Text Screen.

McDowell, Bob, "Integer REM Formatter."
A short utility to assist in formatting REM statements on the Apple.

McDowell, Bob, "Secret," pg. 12.
A routine to provide copy protection on a tape program.

Lew, Art, "READ,, DATA and Selective RESTORE," pg. 21.
A short utility routine for the Apple.

Lew, Art, "Musical Notes," pg. 23.
A simple sound routine for the Apple.

1038. Peelings II 2, No. 1 (January/February, 1981)

Staff, "Apple Programs Software Reviews."
Over 20 programs for the Apple are reviewed in some detail. Included are utilities, sound routines, personal programs, data base management, games and miscellaneous discussions.

1039. Recreational Computing 9, No. 4, Issue 49 (January/February, 1981)

Walker, Robert J., "PET Budget Program," pg. 14.
This program for the PET totals expenses on a daily basis in six categories for the week.

Lopez, Antonio, Jr., "The Key to the Education Revolution," pg. 18-21.
A series of educational math programs, some for the Apple or adaptable to 6502 systems.

1040. Apple-Com-Post Issue 7 (ca. February, 1980)

Goetzke, Uwe, "Neue PASCAL-(Er)kenntnisse," pg. 16-17.
Notes for Pascal users, including a routine for PEEK and POKE, Integer input, etc.

Goetzke, Uwe, "Pascal-eine Einfuehrung," pg. 16.
An introduction to Apple Pascal.

Barbieri, Nino, "Program Kneipe," pg. 20.
A graphics program for the Apple.

1041. Apple-Com-Post Issue 8 (ca. April, 1980)

Schultz, Heinz Juergen, "Verriegelung der RESET — Taste beim Apple II," pg. 8.
Hardware mod to disable the Reset key on the Apple.

Schultz, Heinz Juergen, "Bauanleitung eines Microfon-Verstaerkers fur Apple II-Kassetteneingang," pg. 9.
A hardware addition for the Apple to amplify the cassette input.

1042. Apple-Com-Post Issue 10 (ca. August, 1980)

Reske, B., "HGR-Besonderheiten des ITT 2020," pg. 10.
Notes on Hi-Res graphics for the Apple.

Vermehr, Jochen, "Unterbrechung des Laufenden Programms mit einer Nachricht," pg. 1214.
An interrupt routine using an Apple clock.

Zimmermann, W., "Der Apple II Liest PET-Programme," pg. 15-17.
An Apple program to read PET tapes.

1043. Apple-Com-Post Issue 11 (ca. October, 1980)

Dederichs, W., "Umschalter fuer die Verschiedenen Darstellungsformen des Apple II," pg. 4-8.
A tutorial for the Apple.

Dietrich, M., "Einfuehrung in Assembler," pg. 9-12.
Introduction to assembly language.

Knuelle, A., "Quicksort," pg. 15-16.
Sorting in BASIC and in Pascal.

Reske, Bruno, "Pascal Echounterdrueckung u. — Druckfiles," pg. 17-18.
A Pascal printing program for the Apple.

1044. Apple-Com-Post Issue 12 (ca. December, 1980)

Schultz, H.J., "Umbauanleitung Apple II auf Grosz/Kleinschreibung," pg. 5-6.
An upper case/lower case modification for the Apple.

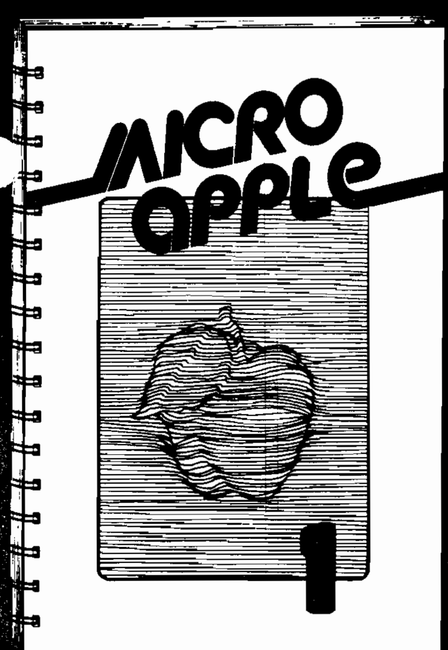
Dederichs, W., "Analyse des Befehls HPLOT ...TO...," pg. 7-8.
All about the HPLOT command in Apple Hi-Res graphics.

Dederichs, W., "Pascal-Text-Files Lesen," pg. 12-15.
An Applesoft program to read Pascal text files.

MICRO

Advertisers' Index

GET MORE OUT OF YOUR APPLE WITH MICRO on the APPLE



MICRO/Apple

Over 30 Apple Programs on Diskette — For Less Than $1.00 Apiece! No Need to Type In Hundreds of Lines of Code!

224-page book and diskette $24.95*

*Add $2.00 for surface shipping. Massachusetts residents add 5% for sales tax.

MICRO's new book for Apple II users lets you

- Speed up programming in Applesoft and Integer BASIC!
- Add Apple II Plus editing features — at no cost!
- Round and format numbers accurately in business applications!
- Get lowercase letters and punctuation into Applesoft strings — at no cost!
- Do a shape table easily and correctly!
- Play the hit game "Spelunker"!
- And much, much more!

With MICRO/Apple 1, the first volume in our new series, you receive

- 30 choice articles from MICRO (1977-80), complete with listings, all updated by the authors or MICRO staff,

plus

- 38 tested programs on diskette (13 sector, 3.2 DOS format, convertible to 3.3).

Ask for MICRO/Apple at your computer store or

Call Toll-free 800-227-1617, Ext. 564

In California, call 800-772-3545, Ext. 564

VISA and Mastercard Accepted

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, Massachusetts 01824

Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT
CONSUMER PRODUCTS

Microsoft Consumer Products, 400 108th Ave. N.E.,
Bellevue, WA 98004. (206) 454-1315