NO. 55 DECEMBER 1982

MICRO™

THE 6502/6809 JOURNAL

p.70: SPEECH
p.73: AIM REPAIR
p.83: WW BOARD
p.35 BUL BOARD

Commodore Feature

ATARI Graphics

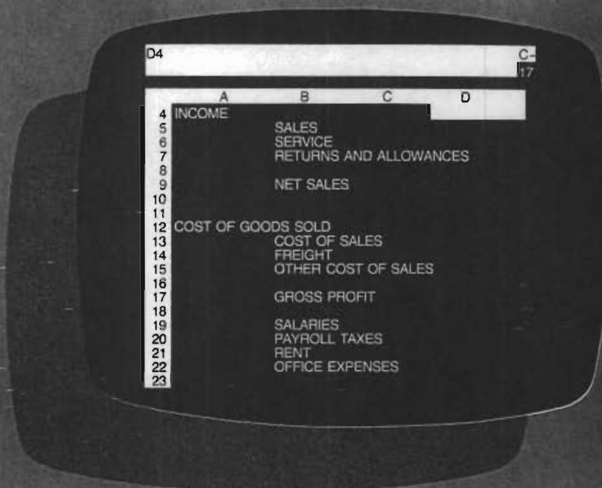
APPLESOFT GOTO/GOSUB Checker

68000 Logic Instructions

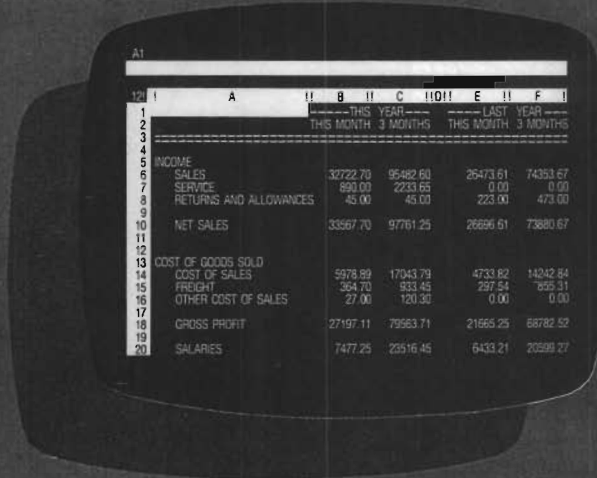
0 74470 11690 12

MAGICALC™

The Visual Difference



VisiCalc™



MAGICALC™

You Won't Need VisiCalc™ To See Your Savings

MAGICALC is the second generation of spreadsheet programs. A state-of-the-art system, it provides you with all the power you need by fully utilizing the most current software and hardware breakthroughs—without your having to break the bank to afford it.

MAGICALC includes:

- **70-column upper- and lower-case video display**
- **Full 80-column board display**
- **40-column standard display**
- **Individual column widths**
- **Invisible columns for confidential data**
- **Full compatibility with VisiCalc™**
- **Hard-disk compatibility**
- **Special offer for VisiCalc™ owners**
- **Magic editing keys**

MAGICALC™—An outstanding product at a pleasing price

$149.95

Breakthrough to today's technology by contacting:

artsci inc. ™ where art and science work for you

5547 satsuma avenue • north hollywood, california 91601 • 213/985-2922

MICRO™

December Highlights

Commodore Machines Featured

This month we cover the full range of Commodore's machines: the PET, VIC, SuperPET, and the exciting new Commodore 64. Each machine has its own distinct features, but also shares characteristics with the other Commodore family members. CBM users will want to read all the Commodore related articles in this issue.

The second part of the University of Rochester's series (p. 59) discusses the use of an inexpensive device, the analog transducer, which can be applied to many problems outside the college teaching laboratory. The analog transducer makes it possible for your digital computer to deal with quantities measured on a continuous scale — light, voltages, densities.

Contributing Editor Jim Strasma starts on a six-part series (p. 37) that will help you write better program packages. In particular, it will cover CBM's powerful, yet poorly understood, relative record system. The first part, however, deals with designing a modular program package, setting things up, and passing parameters. Jim uses portions of the public domain program "Bennett's Mail List 4040" to illustrate his points.

We also offer a number of utilities for Commodore machines. Hans Hoogstraat's "BASIC Squeeze for PET" (p. 42) is a cassette buffer-sized program that can be saved with a fully expanded and commented BASIC program. When the program is run, it makes a call to the squeeze routine, which compresses the program to take less space and run faster. Troup and Strasma's "SOUP" (p. 52) is a compare program for machine-language routines saved on disk. Thomas Henry's "BASIC Line Delete for PET and VIC" (p. 47) adds the capability of deleting more than one BASIC program line at a time.

In our "Short Subjects" section (p. 97) we have two items of interest to users of Commodore machines. Terry Peterson explains the ASCII character set on the SuperPET and reveals some hidden features. "VIC Jitter Fixer," by Contributing Editor Dave Malmberg, can be added to your paddle, joystick, and light-pen programs to give you more reliable readings from these devices.

Finally, we feature the new Commodore 64 computer in both "PET Vet" and on our data sheet. Loren Wright's column (p. 54) reviews the graphic capabilities of this exciting new computer, and the data sheep (p. 109) provides a memory map, interfacing information, and lists of graphics and sound registers.

Expand Your Computer's Capabilities with New Hardware

The BSR X-10 allows you to control remotely a wide variety of electrical devices in your home. There are two versions available; one sends its signals using power lines as antennas, and another uses ultrasonic signals. Each light or appliance is connected to its own receiver module. John Krout's "Home Control Interface for C1P" (p. 77) shows how to add ultrasonic circuitry to your computer at a cost much less than the BSR ultrasonic option. David Hayes's "Atari Meets the BSR X-10" (p. 82) shows how to convert the unit for control from Atari's controller ports.

If you've ever looked at a 6502 programming manual, you might have noticed all the unused op codes. Now you can use those codes to execute your own machine-language routines. Curt Nelson and his associates ("Utilizing 6502's Undefined Operations," p. 93) present a circuit that causes the 6502 to execute your code, instead of crashing, when it encounters an unused op code.

In "Programmable Character Generator for OSI" Colin Macauley demonstrates how to define your own characters (p. 88). OSI readers shuld turn to our OSI book announcement on page 25.

Joe Hootman's in-depth coverage of the 68000's instruction set continues (p. 85) with a discussion of the logic instructions. As usual, convenient reference tables are included.

Apple and Atari

Paul Swanson concludes his three-part series on Atari's character graphics (p. 22) with a demonstration of patching into Atari's vertical blank interrupt routine. His "From Here to Atari" column (p. 32) covers a variety of topics, including Atari's new software acquisition centers and some technical tidbits.

Peter Meyer presents an "Applesoft GOTO/GOSUB Checking Routine" (p. 26) that displays all incorrect GOTO and GOSUB references. "ILISZT for Integer BASIC," by Leonard Anderson, is a follow up to a similar program he presented for Applesoft (p. 13). It produces an attractive, formatted listing of your Integer BASIC program, complete with indentation, paging, and other fancy features. Tim Osborn's "Apple Slices" (p. 65) presents a general-purpose binary search routine that can be called using the & vector.

MICRO™

NO. 55 DECEMBER 1982

MICRO™

THE **6502/6809** JOURNAL

COMMODORE FEATURE

37 **It's All Relative — CBM Disk Techniques, Part 1** .... *James Strasma*
Get the most from CBM's powerful disk operating system

42 **Squeeze for PET Programs** .... *Hans Hoogstraat*
Squeeze out imbedded blanks, line separators, and comments

47 **BASIC Line Delete for PET/CBM and VIC** .... *Thomas Henry*
A machine-language program to delete blocks of BASIC lines

52 **SOUP: A CBM Machine-Language Compare Program** .... *Henry Troup and James Strasma*
A compare program for machine-language program files

59 **Microcomputers in a College Teaching Laboratory, Part 2** .... *Richard Heist, Thor Olsen, and Howard Saltsburg*
Analog transducers in a digital world

BASIC AIDS

13 **APPLE ILISZT for Integer BASIC Programs** ...... *Leonard Anderson*
Print your program in a clear, structured format and detect embedded binary code

19 **BASIC Macro Function for Cursor Control on the OSI** .... *Kerry Lourash*
Insert statements with just two keys

22 **ATARI Character Graphics from BASIC, Part 3** ...... *Paul Swanson*
Add to ATARI's vertical blank interrupt routines

26 **APPLESOFT GOTO/GOSUB Checking Routine** ... *Peter J.G. Meyer*
Verify all GOTO and GOSUB references in your program

HARDWARE

69 **Adding Voice to a Computer** .... *Michael E. Valdez*
A low-cost procedure for sampling and reproducing voice

74 **Enhanced Video for OSI C1P** .... *David Cantrell and Terry Terrence*
Add five chips — and several features

77 **Home Control Interface for C1P** .... *John Krout*
Add your own ultrasonic control

82 **ATARI Meets the BSR X-10** .... *David A. Hayes*
Use ATARI's controller ports

85 **68000 Logic Instructions** .... *Joe Hootman*
Our discussion of the 68000 instruction set continues

88 **Programmable Character Generator for OSI** .... *Colin Macauley*
Design your own character set

93 **Utilizing the 6502's Undefined Operation Codes** .... *Curtis Nelson, Richard Villarreal, and Rod Heisler*
Hardware to use these op codes for new pseudo-instructions

DEPARTMENTS

3 December Highlights
7 Editorial
9 Letterbox
30 CoCo Bits
32 From Here to ATARI
35 MICRO News
54 PET Vet
65 APPLE Slices
91 Updates/Microbes
97 Short Subjects
99 New Publications
100 Reviews in Brief
103 Software Catalog
107 Hardware Catalog
108 6809 Bibliography
109 Data Sheet
111 Advertiser's Index
112 Next Month in MICRO

About the Cover

```
100 Fröliche Weinachten
200 Joyeux Noël
300 Felices Pascuas
400 Buon Natale
500 Happy Holidays
```

This month MICRO is taking a holiday from presenting a graphic with a computer theme on our cover. Instead, we want to offer our warmest greetings — in five languages. The colorful lights in the picture belong to the city of Frankfurt, Germany and symbolize the festive glow of the holiday season. Froliche Weinachten!

Cover photo by Phil Daley

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

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to

MICRO INK
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

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

MICRO™ Editorial

Getting to Know You

"It's more useful than my Swiss army knife." Now that's what we like to hear about MICRO and that's what one of you said in response to our reader survey. But we did the survey for more than a pat on the back.

We did the survey to find out just as much as we can about who you are and what kind of information, both in editorial content and advertising, you need and want.

We discovered that you are an extremely well-educated, affluent, gainfully employed bunch of people with a great deal of technical computer knowledge at your command — and you want more.

33% of you have advanced degrees
70% have incomes over $25,000
60% are programmer/analysts, engineers, or technicians, and
90% of you have intermediate to advanced knowledge of software and 80% of hardware.

No wonder only 6% of our readers consider MICRO too technical. Your biggest beef? Not enough information on your own system — whatever that may be. Too much Apple, not enough Apple, not enough Atari, not enough OSI. Now we know that that is going to be something of a problem in a publication that covers more than one system, or more than one chip, but we think it's important to cross-fertilize, to generalize, to bring you knowledge and information that is transferable. Our goal is to make at least half of the magazine non-system specific, while dividing the other half in much the way our readers are divided — about half Apple and the other half heavily weighted toward OSI, Commodore, Atari, and 6809 systems. Interest in the 6809 and 68000 remains high, especially among users who are adding boards and processors to 6502 machines.

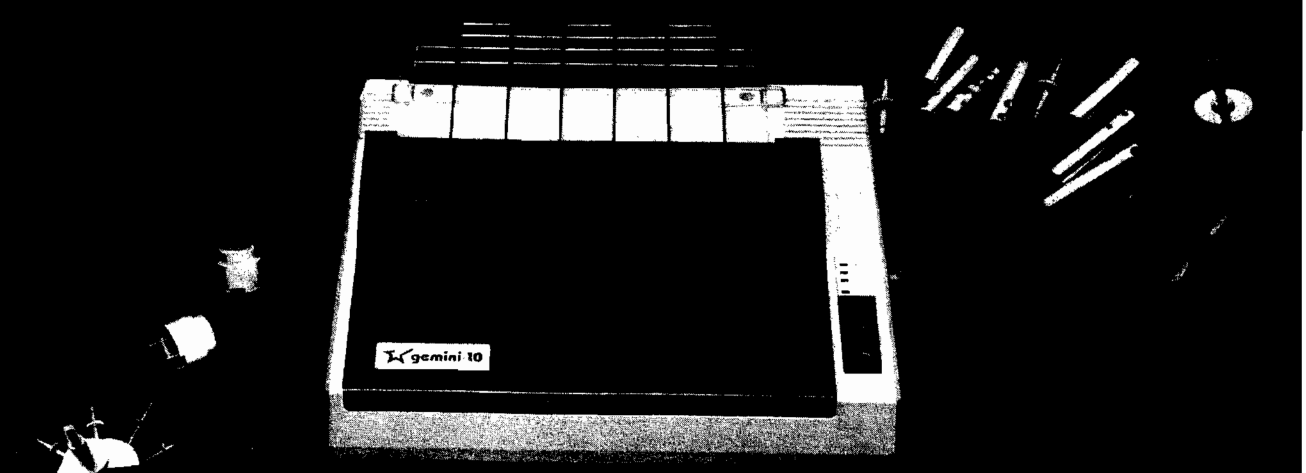
A great many of you (62%) use more than one kind of system and 46% have systems both at home and at work; nearly all of you plan to spend money adding more equipment during the coming year. We trust that the reviews, hardware and software catalogs, and advertisements are helping you make those purchases.

There is a great proliferation of system-specific publications and more and more information for the beginning computer user. We are trying not to clutter up the magazine with information you already have — you've learned a lot over the last few years and we want to help you build on that knowledge. You've matured, the market has matured, and MICRO is growing along with you. The system-specific magazines are a great place to get hints, corrections, fixes, and details about your own equipment — the kind of material it made sense for us to publish back in 1977 when no one else coverd the 6502. But now that manufacturers are doing a better job of providing documentation and there are lots of publications for beginners, we want to concentrate on more advanced issues that cut across machine and processor lines, that keep you abreast of new developments and stretch your knowledge into new areas.

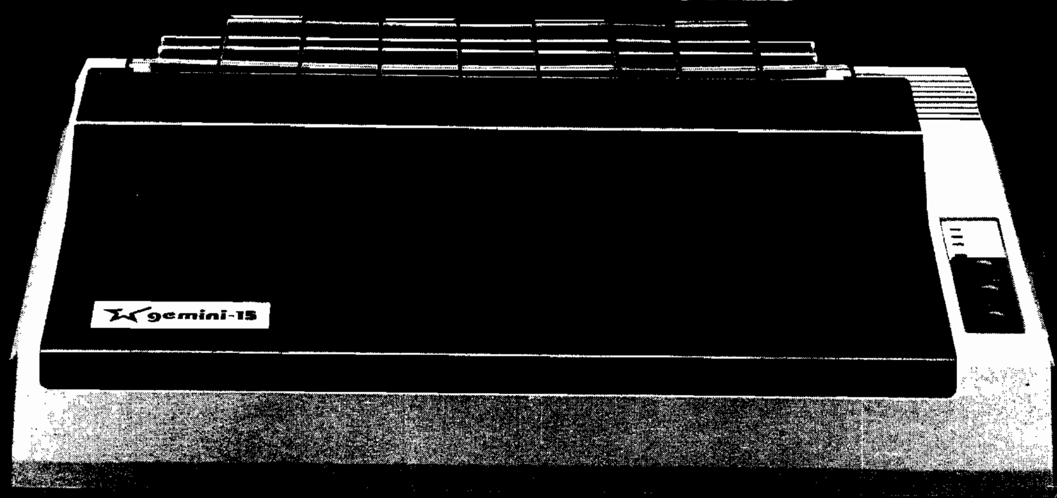
MICRO's editorial schedule for the next year reflects that concern. This is the last system-specific feature we'll be running. Upcoming issues will feature various kinds of peripherals, languages, operating systems, communications. With your strong engineering background you'll want to know what new processors are being developed and how they can be used even before they're available in complete systems. There are new programming languages being developed — we will look at what they are, which ones are worth pursuing for what purposes, etc. We will provide information in the form of data sheets and information sheets on a variety of products and issues. And most interesting of all we will explore new modes of computer use: e.g., networks, communications, automated offices, and industrial control systems.

We think that advanced computer expertise is best imparted in a journal that doesn't limit itself to one system or one chip or one operating system. After all, the whole industry is moving toward compatibility and we think that is a step in the right direction. In light of that fact, and as a result of all we've learned about you and your interests from the survey, as of next month (i.e., with the January 1983 issue), we will change MICRO's subtitle to ***"Advancing Computer Knowledge."*** We are in no way abandoning the 6502 or the 6809 or any of the specific systems we've been covering. We are, instead, making a statement about your technical expertise, your maturity and the industry's, and our desire to move toward ever increasing compatibility and wider proliferation of advanced information and knowledge. You — the sophisticated user — need your own publication; we hope it's MICRO.

Mary Grace Smith



GEMINI–
FOR PRINTER VALUE THAT'S OUT OF THIS WORLD



Over thirty years of down-to-earth experience as a precision parts manufacturer has enabled Star to produce the Gemini series of dot matrix printers—a stellar combination of printer quality, flexibility, and reliability. And for a list price of nearly 25% less than the best selling competitor.

The Gemini 10 has a 10″ carriage and the Gemini 15 a 15½″ carriage. Plus, the Gemini 15 has the added capability of a bottom paper feed. In both models, Gemini quality means a print speed of 100 cps, high-resolution bit image and block graphics, and extra fast forms feed.

Gemini's flexibility is embodied in its diverse specialized printing capabilities such as super/sub script, underlining, backspacing, double strike mode and emphasized print mode. Another extraordinary standard feature is a 2.3K buffer. An additional 4K is optional. That's twice the memory of leading, comparable printers. And Gemini is compatible with most software packages that support the leading printers.

Gemini reliability is more than just a promise. It's as concrete as a 180 day warranty (90 days for ribbon and print head), a mean time between failure rate of 5 million lines, a print head life of over 100 million characters, and a 100% duty cycle that allows the Gemini to print continuously. Plus, prompt, nationwide service is readily available.

So if you're looking for an incredibly high-quality, low-cost printer that's out of this world, look to the manufacturer with its feet on the ground—Star and the Gemini 10, Gemini 15 dot matrix printers.

star®
micronics • inc
MAKING A NAME FOR OURSELVES

1120 Empire Central Place, Suite 216, Dallas, TX 75247
For more information, please call Bob Hazzard, Vice President, at (214) 631-8560.

Back and FORTH

Dear Editor:

I was quite pleased with the two articles on FORTH in the June issue of MICRO. Regarding the benchmark comparisons of BASIC, FORTH, and RPL (page 63), I would have to say that Mr. Stryker is apparently somewhat biased in his viewpoint, since he is the father of RPL. What he appears to have done is take perfectly readable FORTH and translate it into hieroglyphics. Surely, the FORTH word DUP is more meaningful as a stack operator than "#", and who would ever guess what ";", ".", and "%" have to do with anything? Single-character words are very useful for lazy typists, but they do tend to produce "write-only" code for those who need to determine what a program is doing.

Every FORTH implementation I have ever seen has a machine-language primitive to handle block moves on a character basis. Why do we go through the gyrations of listing 1B when the word CMOVE would do just as well (actually better!)? Even without using CMOVE, the word BLKM would execute faster and with fewer FORTH words if it were written:

```
: BLKM  OVER + SWAP
        DO DUP C@ I C! 1 +
        LOOP DROP ;
```

This word expects a slightly different order of things to be on the stack than originally specified: FROM TO and COUNT (634 826 150 using his numbers). This is the same order that CMOVE would expect them also. I am sure that this arrangement would be of benefit for RPL as well.

Regarding the SHUFFLER benchmark; first of all, it appears there is a typographical error of omission in line 8 of listing 2B, since the word MOD referred to in the text is not there. Even so, however, the way the routine was implemented can do nothing but slow it down.

Finally, regarding the Falling-Tone benchmark, I certainly feel the author's comments on page 68 regarding how hard it was to come up with a FORTH implementation, show a decided lack of understanding of structured programming! Listing 3A shows the same lack of structure that can be no way blamed on BASIC itself. After analyzing what the program is supposed to do, the following structured code would have been much clearer:

```
1010  DC=20:FOR Z = 20 TO 255
1020  DC=DC-Z
1030  IF DC > = 0 THEN 1020
1040  POKE 59464,Z
1050  DC=DC+256
1060  NEXT
1070  POKE 59467,0:POKE
      59466,0:RETURN
```

The same code written in FORTH looks like this:

```
: TONE 0 59464 C! 16 59467 C!
170 59466 C! 20 256 OVER DO
    BEGIN I - DUP 0 <
    UNTIL
    I 59464 C! 256 +
          LOOP DROP 0 59466 ! ;
```

Notice that we use 0 59466 ! to reset both 59466 and 59467 to zero, since FORTH inherently works with 16-bit numbers and uses 8-bit numbers only occasionally. I would probably do the same thing at the beginning of TONE to set up 59466 and 59467 initially, assuming this is a PIA register address of some sort. At any rate, the structure is there and can also be used in the RPL version, I'm sure.

Edward B. Beach
5112 Williamsburg Blvd.
Arlington, VA 22207

Dear Editor:

In "BASIC, FORTH, and RPL" (MICRO 49:63), three different computer languages are compared in terms of speed and memory economy using three benchmark programs. However, within the text of the article there were some comments made about FORTH by the author, Timothy Stryker, which require rebuttal.

Mr. Stryker states that program modules in RPL do not execute directly but rather place their address on the stack where a second call operator (&) actually executes this address. As correctly noted, this is in contrast to FORTH where the defined word directly executes; it does not need a second execute operator. This allows *all* FORTH definitions to be treated as syntactically equal. Programmers may freely mix FORTH language words with their own new definitions — indeed, there is no difference in the internal dictionary structure between these two parts.

On the other hand, RPL forces us to use (&) for execution of all new words while pre-existing ones are immune to this rule and execute directly, creating an inconsistent syntax. That this is memory efficient is doubtful. The higher level definitions of any nontrivial application program can consist of a large proportion of user-defined operators, each one of which would require the addition of this execute operator in RPL. This probably consumes some memory in the compiled form and it certainly and unnecessarily clutters up the source code. With FORTH, the address of any definition can be placed on the stack with an additional operator when it is desired, although this function is seldom needed.

It is true that FORTH handles symbols differently depending on whether they are variables, constants, or executing subroutine names. This is part of the beauty of the language, not a weakness. Each type of symbol has a different function. Subroutine names execute, constants leave their value on the stack, and variables leave their address so we can suffix them with load or store operators. Nothing could be simpler or more efficient: uniformity of function by means of inconsistent internal operation. RPL reverses this, giving us consistent internal operation while forsaking clarity of function at the programmer's level. This forces us to be even more aware of what each definition does — something I would prefer to be left up to my compiler.

As Mr. Stryker correctly states, the FORTH string literal print word (.") and the numeric print words never leave their output string on the stack. This is seldom needed and would possibly slow down the system. Besides, the stack may not be large enough to safely handle this, since on the 6502 the FORTH stack is placed in page zero (shared with a few other FORTH locations and probably some used by the host computer for disk or terminal I/O). If we need to alter the string in numeric conversion and printing, FORTH has some primitives available for inserting additional characters in the string. With a minor effort we can add print using to an application program or make it a permanent part of the FORTH we use each day. Other than the string literal defining word (."), there are no other string operators defined in the FORTH standards, but these are not difficult to add to such an easily extensible language.

Some additional points: The modulo primitive in the fig-FORTH 6502 model takes 1.2 milliseconds to execute. No random-number generator is defined by the Group, so the poor speed of this word in Mr. Stryker's unnamed FORTH version was not optimized for speed by whomever wrote it.

Language experimentation and comparison is certainly needed to fuel the evolutionary process of computer technology. But it should best be done with the full understanding of each language involved.

Raymond Weisling
Jalan Citropuran No. 23
Solo, Jawa Tengah
Indonesia

Dear MICRO:

Thanks very much for the chance to respond to Mr. Beach and Mr. Weisling in regard to their letters concerning my recent article.

First of all, I take exception to the contention in both of these letters that I unjustly biased the benchmarks and the conclusions drawn therefrom in favor of RPL. In fact, precisely *because* I knew that this objection might be raised, I bent over backward to give the benefit of every doubt to FORTH. This may not be immediately apparent in the article because I did not make a point of saying so, but, for example, wherever my measured execution times varied slightly from one run to the next, I uniformly presented FORTH's fastest time, and RPL's slowest; for another, I specifically excluded from consideration any benchmarks involving manipulation of character strings, stack-resident arrays, finite-state automata, and other operations that RPL handles much more naturally than FORTH. Further evidence of this concern will become apparent below.

First I'll address Mr. Beach and his comments on the use of single-character operator-tokens. I do agree that RPL source must look like hieroglyphics to a person versed in FORTH — but perhaps you remember what FORTH (or any computer language) looked like before you became fluent in it. Experienced RPL users have as little difficulty reading RPL source as you do reading FORTH. The advantages of single-character operator-tokens are three: 1. as you acknowledge, they cut down on typing time; 2. they cut down on the physical size of the source, so that more source can be fit into memory at once when undertaking nontrivial applications; and 3. they speed up compilation by cutting down on the operator-token search time.

Thank you for pointing out a better method of doing block moves in both FORTH and RPL. In writing the benchmarks, I was primarily concerned about making sure that the FORTH and RPL versions were as close to identical in approach as possible, so I missed seeing that the block móve could be done more efficiently in the way you suggest. You may be interested to know, though, that the FORTH source you show for this routine yields an execution jiffy-count of 717, considerably in excess of the 591 given for FORTH in the article. The reason? Your use of the composite "1+" operator in the innermost loop. When the sequence "1 +" is substituted for this, the execution time falls to 584 jiffies. Spaces, as you note in your letter *are* important in FORTH — one might even say, alarmingly so. They make no difference in RPL. Unfortunately, the use of even the sped-up form of your block-move algorithm does not change the standings. FORTH requires 84 program bytes to do it in 584 jiffies, whereas the following RPL equivalent:

```
BLKM: ; + 1 - % FOR # PEEK FN
POKE 1 + NEXT . RETURN
```

requires only 52 bytes to do it in 508, a "merit ratio" of 1.85 to 1.

Now, there seems to be some confusion in your letter regarding various aspects of the SHUFFLER benchmark. To begin with, there are no typos anywhere in the article. The MOD routine is, as stated, internal to the RND routine I used. This RND routine, modeled after that available under MMSFORTH, expects an integer passed to it on the stack, and returns a random number in the range from 0 up to that integer minus 1 — hence, the MOD.

Moving on to your comments regarding the third benchmark: you are right. There was no need for me to introduce unstructured code in this case. The new FORTH TONE routine you exhibit takes only 3465 jiffies, and requires only 130 bytes of program space. The corresponding RPL routine is:

```
TONE: 0 59464 POKE 16 59467 POKE
170 59466 POKE 20 256 ; FOR
LOOP: FN - # 0 < IF
FN 59464 POKE 256 +
THEN LOOP GOTO END
NEXT . 0 59466 ! RETURN
```

which requires 83 bytes of storage and executes in 3338 jiffies. The resulting merit ratio of 1.62 to 1 represents a considerable improvement. You were right, incidentally, not to condense the leading POKEs of 59467 and 59466 into a single store — the order of the POKEs into those 6522 VIA registers makes a big difference.

On to Mr. Weisling's letter. Programmers who are bothered by the necessity of suffixing their subroutine references with an ampersand in RPL are free to eliminate the space separating the two and thereby regard the composite "SUBRNAME&" as just a one-keystroke-longer method of invoking the routine. You doubt that this is memory efficient. Please find out for certain by way of the following procedure: take any nontrivial FORTH application program to which you have access and count up the number of occurrences of (A) invocations of the thirty or forty real low-level FORTH "primitives" such as DUP, "=", IF, DO, "@", and things of that nature (including ";" but not including ":"); (B) references to literal numeric quantities, whether CONSTANTs or not, it does not matter, which fall in the range from 0 to 63; (C) references to literal numeric quantities greater than 63 but less than 32768, plus all references to VARIABLEs, CVARIABLEs, and whatnot; (D) all references to literal numeric quantities not covered under B or C; and (E) all routine-invocations (other than ":") not covered under A. Be sure, if you count a routine-invocation under E, that you also consider the body of that routine part of the program source. Now form the sum A + B + 2*C + 3*D + 3*E. This is a rough approximation of the number of object program bytes that would be required, were the program translated, absolutely mechanically from FORTH into RPL. Multiply this by about 0.8 to arrive at the memory size of the equivalent program, had it been designed in RPL to begin with.

Next, a discussion on symbol handling. The fact that RPL is more efficient has been demonstrated already. That it is simpler may be difficult to appreciate second-hand like this, but RPL "gives us consistent internal operation" *without* forsaking "clarity of function at the programmer's level." The question of how aware the programmer needs to be as to "what each definition does" has nothing to do with it.

The ability to manipulate character strings conveniently is fundamental to most user-oriented software development. Indeed, your remark about the size and location of the FORTH stack points up the fact that this is one area in which FORTH's extensibility does it little good. RPL locates both stacks in page one: the parameter stack is the hardware stack, and the return stack is an indexed sort of affair down below it. Stack-resident strings up to 60 characters long or so can be manipulated freely without fear of crashing the machine — and execution is brought to a controlled halt if the 64-word stack entry limit is exceeded.

And on your last point: under my version of FORTH, a public-domain version identifying itself simply as "fig-FORTH 1.0" (which, however, includes such exotic facilities as double-precision and floating-point math, IEEE-488 I/O, etc.), the following routine, as timed with an actual watch, takes 2 minutes and 40 seconds to execute:

```
: TEST 30000 0 DO 6543 52 MOD DROP
LOOP ;
```

When the MOD is replaced with another DROP, it takes 14 seconds. I leave you to draw your own conclusions.

Timothy Stryker
Samurai Software
P.O. Box 2902
Pompano Beach, FL 33062

MICRO

Your opinions, comments, and criticisms can be aired in MICRO too. Send mail to Letterbox, MICRO, P.O. Box 6502, Chelmsford, MA 01824.

CHRISTMAS SEASON SPECIALS!

Let ARK COMPUTING Make This Your Best Christmas Ever!

Super Fan II by R.H. Electronics **59.95/79.95**

Applicard, a high performance Z-80 card with 64K Ram, complete with CP/M
4 mhz **324.95**/445.00
6 mhz **395.00**/595.00

Microsoft Z-80 card with CP/M and Microsoft Basic
2 mhz **269.95**/395.00

Microtek Parallel Printer Interface complete with centronic compatible connector
64.95/79.95

Lazer Lower Case +Plus with Character Set +Plus **49.95**/84.90
Lower Case +Plus alone **39.95**/59.95

Lazer Graphics +Plus **99.95**/159.95
Graphics +Plus and Lower Case +Plus **134.95**/219.90

Computer Stop 16K Ram Board **69.95**/149.95

Computer Stop Omnivision 80 Column board
129.95/295.00

Videx Video-term with softswitch, inverse character set and 80 column Visicalc preboot
295.00/450.00

Wizard BPO 16K buffered printer interface (expandable to 32K) **134.95**/179.95

Wizard 80, 80 column board **195.00**/295.00

Lazer Pascal **29.95**/39.95

Anix 1.0 **34.95**/49.95

Lazer Forth **44.95**/59.95

D Tack 68000 board for the Apple II with 4K Ram **895.00**

Lazer Model/32 (16032 board for the Apple II)
CALL!

Lisa	**59.95**/79.95
Lisa Educational Pak	**79.95**/119.95
Alien Ambush	**19.95**/29.95
Bandits	**19.95**/29.95
Cannonball Blitz	**24.95**/34.95
County Fair	**19.95**/29.95
Cranston Manor	**24.95**/34.95
Cyclod	**19.95**/29.95
David's Midnight Magic	**24.95**/34.95
Dosource 3.3	**24.95**/39.95
Dueling Digits	**19.95**/29.95
Falcons	**21.95**/29.95
Firebird	**21.95**/29.95
Foosball	**19.95**/29.95
Horizon V	**25.95**/34.95
Genetic Drift	**19.95**/29.95
Kabul Spy	**24.95**/34.95
Jelly Fish	**19.95**/29.95
Lemmings	**19.95**/29.95
Labyrinth	**19.95**/29.95
Mouskattack	**24.95**/34.95
Outpost	**19.95**/29.95
Red Alert	**19.95**/29.95
Pig Pen	**24.95**/34.95
Russki Duck	**25.95**/34.95
Minator	**24.95**/34.95
Track Attack	**19.95**/29.95
Thief	**17.95**/29.95
Space Quarks	**19.95**/29.95
Snack Attack	**19.95**/29.95
Swash Buckler	**24.95**/34.95
Gin Rummy	**24.95**/34.95
The Dictionary	**69.95**/99.95
General Manager	**99.95**/149.95
4 Ft. Disk Cable	**19.95**/29.95
Visicalc	**179.95**/250.00
Using 6502 Assembly Language Book	**14.95**/19.95
Kids and The Apple Computer Book	**15.95**/19.95
Apple Panic	**19.95**/29.95
Kraft Joystick	**49.95**/69.95

ARK COMPUTING INC.

Your Salvation In The Sea Of Inflation.

714 735-2250
P.O. Box 2025
Corona, CA 91720

APPLE ILISZT for Integer BASIC Programs

by Leonard Anderson

ILISZT prints an Integer BASIC program in a clear, structured format with the ability to detect embedded or attached BINARY code.

ILISZT
requires:
Apple II with both
Integer and Applesoft
Disk drive
Printer

The purchase of several disks at the end of 1981 added a number of Integer BASIC programs to my Apple II library. No listings were available and I decided to print all of them.[1] Several had embedded binary code, a condition that caused much "nonsense" display on both screen and printer. "LISZT" was already up and running (MICRO 48:37), so it seemed logical to modify this Applesoft program to format Integer listings. The ILISZT result kept the original format and added the ability to find exact binary code addresses.

ILISZTER is the formatting and printing program, run by EXEC file ILISZT. ILISZTER is Applesoft rather than Integer. While an Integer program might seem better, many Apple II owners possess ROM or RAM cards for language duality and ILISZTER seems more compact in Applesoft due to string-handling capability. Another advantage is that ILISZTER can be re-run without disk operations or loss of Integer source code.

ILISZTER retains the original features such as separation of concatenated statements, indenting, and remark highlighting. Multiple-iterator NEXT statement handling for restoring FOR-NEXT loop indents is an improvement. The added binary code determination and restoration routine is useful for listing certain utilities.[2] Since Integer BASIC differs from Applesoft, a brief review of Integer structure will help provide an understanding of ILISZTER.

Integer BASIC Source Code

Figure 1 shows one line number of source code in Integer. The first byte contains the number of bytes per line with the next two bytes having the line number in binary. End-of-line is signified by the end byte having a value of one.

Each entered line is immediately checked for syntax. Line numbers are limited to 32767 but may be modified by utilities. Numeric constants are converted to binary on entry, an advantage for program execution time.

All function words are stored as one-byte "tokens" in the range of zero to 127 decimal. Punctuation, arithmetic, and logical operators are also tokens. Eight tokens are unused and three others are used only with keyboard entries. ASCII characters have the high bit set to use the decimal range of 128 and 255. Token and character values are opposite that of Applesoft.

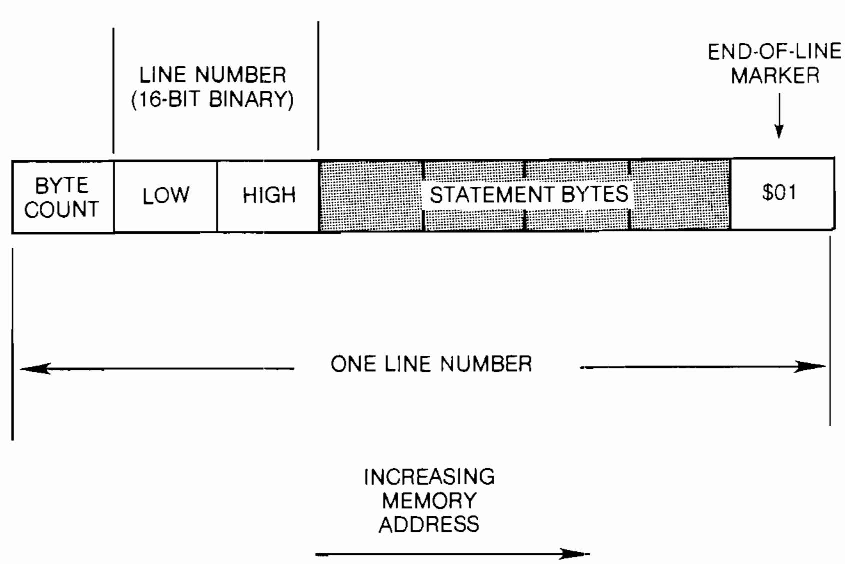
A major difference also exists in handling numeric constants within Integer. Certain functions permit a following numeric constant or variable name. Distinction of a numeric constant is done by making the first byte following an ASCII number ($B0 to $B9, not allowed as first letter of a variable) with the next two bytes containing the numeric constant in binary.

Integer BASIC is located just below the highest free memory address. Integer does not need the three-null end of program marker required by Applesoft. Other details may be found in earlier publications.[3,4,5]

An EXEC File for Glue

If an Integer program exists in memory, loading an Applesoft program will not destroy the Integer source code. Loading does change the Integer start-of-program pointer at $CB, $CA (203, 202). Integer end-of-program, or HIMEM at $4D, $4C (77, 76) remains unchanged.

Figure 1: **Source code structure on one line number in Integer**



HIMEM will restore to the end of free memory on re-loading an Integer program; the mechanism is unknown but confirmed through experiments.

EXEC file ILISZT is executed after loading the Integer program to be listed. The first two POKEs in ILISZT generator MAKE ILISZT will move the Integer HIMEM pointer into the LOMEM space at $4B, $4A (75, 74). LOMEM also restores on Integer reload. The last two POKEs move the start-of-program into the space normally used for Integer HIMEM.

Running ILISZTER will automatically switch over to Applesoft without disturbing the new Integer start and end addresses. MAKE ILISZT can be deleted when EXEC text file ILISZT is generated.

Starting ILISZTER

The first line resets Applesoft high memory to prevent string operations from overwriting the Integer source. Token words are initialized at line 91. Since quotes are tokens if not in a remark, the DATA declaration uses an "&" symbol with conversion *via* the IF and CHR$(34) statement.

A token evaluation array is generated in V at line 96. The V array is used in line parsing to test unused tokens and tokens that may have following numeric constants. Unused tokens (V=2) may be nulls or single spaces; spaces were written just in case the binary-insert routine crashed.

The choice of lower-case characters in token words is up to the user. Mixed-case token words give distinction from normal upper-case variables. Available utilities can edit upper-case source code by adding hexadecimal $20 to each desired lower-case letter.[6]

Initial display at line 98 is optional but it does indicate proper location and operation. The "DIFFERENT START ADDRESS" prompt allows listing to begin *after* an embedded binary; binary addresses will appear in normal printouts. ILISZTER can be RUN after any RESET or list completion without disturbing Integer source code.

Printer control in lines 107 to 110 should be set to your particular printer and interface. Subroutines at lines 17 and 18 can be changed to other runtime control. Source code control characters are converted to letters before output.

Lines that Parse in the Right

A source code line parse begins at

ILISZTER

```
0 PS =  PEEK (77) * 256 +  PEEK (76) - 1: HIMEM: PS: GOTO 82
1  REM "GET BYTE" SUBROUTINE   *
2 P = P + 1:B =  PEEK (P): RETURN
3  REM "BLANK LINE PRINT" SUBROUTINE   *
4 D = 0: GOSUB 6: PRINT S$: RETURN
5  REM "TEST PAGE SUBROUTINE   *
6 LC = LC + 1: IF LC =  < LP THEN  RETURN : REM NOT A NEW PAGE
7  GOSUB 17:LC = 6:PC = PC + 1: PRINT S$: PRINT BB$;LB$;"<continued>"
8  REM A FORM-FEED FOR TOP OF NEXT PAGE; ALLOWS VARIATION FOR DIFFERENT P
   RINTERS.
9  FOR K = 1 TO 4: PRINT S$: NEXT
10  REM PRINT THE HEADER
11 H$(4) = "Integer   Page " +  STR$ (PC): FOR K = 1 TO 4:E =  INT ((LL -
    LEN (H$(K))) / 2) + 1: PRINT M$; LEFT$ (BB$,E);H$(K): NEXT :K =  FRE
   (0): PRINT S$: IF  NOT D THEN  RETURN
12  REM PUT LINE NUMBER IN BRACKETS AS A STATEMENT IDENTIFICATION ON NEXT
    PRINT PAGE
13 N$ =  STR$ ( VAL (N$)):K =  LEN (N$): REM N$ IS NOW WITHOUT SPACES; BR
   ACKET N$ AND ATTACH TO STATEMENT CHARACTERS
14 C$ =  RIGHT$ (( LEFT$ (LB$,(6 - K)) +  CHR$ (91) + N$ +  CHR$ (93) + S
   $),8) +  RIGHT$ (C$,( LEN (C$) - 8)):K =  FRE (0): RETURN
15  REM *   *   MX-80 STANDARD/ITALICS SUBROUTINES   *   *   *
16  REM "GRAFTRAX" Only.  Single-character-set printers should DELETE the
   se calls throughout if not used for other print functions.
17  PRINT  CHR$ (27)"5";: RETURN : REM ESC-5 IS STANDARD SET
18  GOSUB 17: IF RF THEN  PRINT  CHR$ (27)"4";: REM ESC-4 IS ITALICS SET
19  RETURN
20  REM HEXADECIMAL CONVERT SUBROUTINE   *
21 A$ = "": REM ENTER WITH 'L' AS DECIMAL NUMBER, RETURN IN 'A$'
22  FOR K = 1 TO 4:D =  INT (L / 16):E =  INT ((L - (D * 16)) + 1):L = D:
   A$ =  MID$ (X$,E,1) + A$: NEXT : REM PREFIX THE "$" HEX NOTATION
23 A$ = "$" + A$:K =  FRE (0): RETURN
24  REM BEGIN A NEW LINE NUMBER WITH TEST OF NUMBER OF BYTES IN LINE FROM
    FIRST BYTE, THEN CONVERT BINARY LINE NUMBER TO DECIMAL
25  GOSUB 2: IF P =  > PE GOTO 123: REM POINTER EQUAL TO OR BEYOND END OF
    INTEGER PROGRAM
26 LA = P:BC = B: IF B > 127 GOTO 114: REM BYTE COUNT TOO LARGE, PROBABLE
    ATTACHED BINARY
27 TN = TN + 1: REM BUMP LINE NUMBERS, THEN MAKE LINE NUMBER STRING
28  GOSUB 2:L = B: GOSUB 2:L = B * 256 + L:B =  LEN ( STR$ (L)):N$ =  RIGHT$
   (( LEFT$ (LB$,(7 - B)) +  STR$ (L) + "  "),8)
29  REM BEGIN STATEMENT LINE PARSING WITH FIRST-BYTE DECISION
30 D = 0: GOSUB 2: IF B = 93 AND  NOT RF THEN  GOSUB 4: GOTO 34: REM SEPA
   RATE REM-GROUPS BY BLANK LINES
31  IF B = 93 AND RF GOTO 34
32  IF RF THEN RF = 0: GOSUB 4
33  REM RE-ENTRY POINT FOR NEXT BYTE IN STATEMENT DECISION
34  IF B < 128 GOTO 39: REM BYTE IS A TOKEN
35  IF B = 255 THEN B = 159: REM RUBOUT ($FF) BECOMES UNDERLINE BETWEEN B
   ARS
36 B = B - 128: IF B < 32 THEN B = B + 64:G$ = G$ +  CHR$ (124) +  CHR$ (
   B):B = 124: REM PUT CONTROL CHARACTERS BETWEEN BARS
37 G$ = G$ +  CHR$ (B): GOSUB 2: GOTO 34
38  REM    TOKENS
39  IF V(B) > 1 THEN G$ = "": GOTO 114: REM UNUSED TOKEN, PROBABLE BINARY
    PROGRAM ATTACHED SO GATHERING IS NULLED
40  IF B = 1 OR B = 3 THEN G$ = G$ + S$: GOTO 57: REM FORCE A NEW PRINT L
   INE ON E-O-L OR A COLON DELIMITER; SPACE ATTACHED TO PREVENT PRINT-L
   INE CRASH
41  IF B = 93 THEN TR = TR + 1:RF = 1:RS = 1: REM A "REM"
42  IF B = 37 AND  PEEK (P + 1) = 85 THEN G$ = G$ + T$(B):CF = 1: GOTO 57
   : REM FORCE A NEW LINE ON "THEN" FOLLOWED BY "FOR", SET CONDITIONAL
   FLAG
43  IF B = 85 THEN FF = 1: REM A "FOR"
44  IF B <  > 89 GOTO 51: REM SKIP AROUND A "NEXT"
45 FS = FS - 1:PT = P + 1: IF CF THEN FS = FS - 1: REM DECREMENT "FOR" SP
   ACER ON "IF" FLAG SET, BEGIN SCANNING AHEAD FOR 2 OR MORE ITERATORS
46 BT =  PEEK (PT): IF BT = 1 OR BT = 3 GOTO 49: REM NO OTHER ITERATOR
47  IF BT = 90 THEN FS = FS - 1: REM COMMA FOUND, DECREMENT "FOR" SPACER
48 PT = PT + 1: IF PT <  = (LA + BC) GOTO 46: REM CHECK AGAIN FOR ANOTHER
    COMMA WITHIN LINE
49  IF FS < 0 THEN FS = 0
50  REM GATHER TOKEN THEN TEST FOR A FOLLOWING 3-BYTE NUMBER GROUP
51 G$ = G$ + T$(B):L = B: GOSUB 2: IF V(L) = 0 GOTO 34: REM NO NUMBER SHO
   ULD FOLLOW
52  IF B < 176 OR B > 185 GOTO 34: REM THE $B0-$B9 FIRST-BYTE NOT THERE S
   O NO NUMBER FOLLOWS.  FALL-THROUGH IGNORES FIRST-BYTE AND DOES DECIM
   AL STRING CONVERSION
53  GOSUB 2:L = B: GOSUB 2:L = B * 256 + L:G$ = G$ +  STR$ (L): GOSUB 2: GOTO
    34
54  REM ADD EXTRA INDENT EACH SPLIT LINE, LIMITING ON "REM" STATEMENTS
55 TS = TS - 1:SF = 0:RS = RS + 1: IF RS > 2 THEN RS = 2
56  REM FIRST ENTRY TO PRINT-LINE BUILD, GET TOTAL INDENT SPACES PLUS SPL
    IT-POINT LOW LIMIT 'E'
57 TS = TS + 1:K = IM * (FS + RS):E = K + 13: IF K > 0 THEN G$ =  LEFT$ (
   BB$,K) + G$
```

(continued)

```
58  REM BUILD TOTAL PRINT-LINE STRING
59  IF  NOT D THEN C$ = N$ + G$
60  IF D THEN C$ = LB$ + G$
61  REM TEST FOR LONG LINE, SPLIT IF NECESSARY
62 K =  LEN (C$) - LL: IF K < 1 GOTO 74: REM NOT A SPLIT LINE
63 G$ =  RIGHT$ (C$,K):C$ =  LEFT$ (C$,LL):SF = 1
64  REM BEGIN SPLITTING WITH SEARCH FOR A SPACE
65 D = LL
66  IF  MID$ (C$,D,1) = S$ GOTO 72
67 D = D - 1: IF D > E GOTO 66
68 D = LL: REM SPLIT NEXT AT ARITHMETIC OPERATOR OR COMMA
69 K =  ASC ( MID$ (C$,D,1)): IF K = 42 OR K = 43 OR K = 44 OR K = 45 OR
    K = 47 OR K = 124 GOTO 72
70 D = D - 1: IF D > E GOTO 69: REM FALL-THROUGH IS NO SPLIT
71  GOTO 74: REM NEXT LINE IS SPLITTING INSTRUCTION
72 K = LL - D: IF K > 0 THEN G$ =  RIGHT$ (C$,K) + G$:C$ =  LEFT$ (C$,D)
73  REM TEST PAGE LINE-COUNT, INSERT SPACES AS REQUIRED, THEN PRINT
74  GOSUB 6:K =  LEN (C$): IF SF = 0 OR K < 2 OR RF THEN 77: REM FORGET M
    ARKING UNDERLINING ON "REM"S
75  IF  MID$ (C$,K,1) = S$ THEN C$ =  LEFT$ (C$,(K - 1)) +  CHR$ (95): REM
    PUT A TRAILING UNDERLINE AT LAST SPACE AS A MARKER FOR THE LEFT-HAND
     STRING
76  IF  LEN (G$) > 2 AND  LEFT$ (G$,1) = S$ THEN G$ =  CHR$ (95) +  RIGHT$
    (G$,( LEN (G$) - 1)): REM PUT A LEADING UNDERLINE AT FIRST SPACE OF
    RIGHT-HAND STRING AS A MARKER
77  GOSUB 17:K =  LEN (C$): PRINT M$; LEFT$ (C$,8);: GOSUB 18: PRINT  RIGHT$
    (C$,(K - 8)):K =  FRE (0): IF SF THEN D = 1: GOTO 55: REM PRINT REST
     OF A SPLIT LINE
78 RS = 0: IF FF THEN FS = FS + 1:FF = 0
79 D = 0:SF = 0:G$ = "": IF B = 1 GOTO 25: REM GET ANOTHER LINE NUMBER IF
    E-O-L, ELSE FALL THROUGH AND GET ANOTHER STATEMENT
80  GOSUB 2:D = 1: GOTO 34
81  REM INITIALIZATION OF VARIABLES
82  DIM T$(127),H$(4),V(127)
83  REM INITIAL VARIABLE SETTING HAS AN 80-CHARACTER WIDE PRINT LINE AND
    82-LINE PAGE LENGTH (INCLUDING HEADER, EXCLUDING 'CONTINUED' INDICAT
    OR); CHANGE LL AND LP AS DESIRED FOR OTHER FORMAT SIZE.
84 PE =  PEEK (75) * 256 +  PEEK (74) - 1:P = PS: REM PS = INTEGER PROGRA
    M START ADDRESS MINUS ONE, PE = INTEGER PROGRAM STOP ADDRESS MINUS O
    NE
85 B = 0:LL = 80:LP = 82:IM = 4:TN = 0:TS = 0:TR = 0:S$ = " ":X$ = "01234
    56789ABCDEF":C$ = "":G$ = "":N$ = "":M$ = "":LB$ = "        ":BB$ =
    "                                                "
86  REM 'T$' ARRAY STRING CONSTANTS FOR PRINTING TOKENS
87  DATA " ","<<"," ",":","Load ","Save ","Con","Run ","Run","Del ",",","
    New","Clear","Auto ",",","Man ","Himem : ","Lomem : ","+","-","*","/
    "," = "," # "," >= "," > "," <= "," <> "," < "," And "," Or "," Mod
    ","^"," ","(",","," Then "
88  DATA " Then ",",",",","&","&","("," "," ","(","Peek","Rnd","Sgn","Abs
    ","Pdl"," ","(","+","-","Not ","("," = "," # ","Len(","Asc(","Scrn("
    ,",","(","$"," ","(",",",",",";",";",";",",",",",",","Text","Gr","Ca
    ll ","Dim ","Dim ","Tab "
89  DATA "End","Input ","Input ","Input ","For "," = "," To "," Step ","N
    ext ",",","Return"," Gosub ","*   "," "," GoTo ","If ","Print ","Pri
    nt ","Print","Poke ",",","Color = ","Plot ",",","HLin ",","," At ","
    VLin ",","," At ","VTab "
90  DATA " = "," = ",")"," ","List ",",","List","Pop","NoDsp ","NoDsp ","
    NoTrace","Dsp ","Dsp ","Trace","Pr # ","In # "
91  FOR K = 0 TO 127: READ T$(K): IF T$(K) = "&" THEN T$(K) =  CHR$ (34):
     REM ONE WAY TO GET A DOUBLE QUOTE INTO A STRING
92  NEXT
93  REM 'V' ARRAY CONSTANTS FOR TOKEN TESTING
94  DATA 2,0,2,0,0,0,0,1,0,1,1,0,0,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,2,1,1,1,0,0,0,0,0,1,2,2,1,0,0,0,0,0,2,1,1,1,1,1,0,0,0,0,1,1,1,0,2,
    1,1,1,0,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,0,0,0,1,0,0,1,1,0,1,0
95  DATA 1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,2,1,1,0,0,0,0,0,0,0,0,1,1
96  FOR K = 0 TO 127: READ V(K): NEXT
97  REM SCREEN PROMPTS AND OPERATOR ALTERNATES
98  HOME : TEXT : VTAB 2: HTAB 12: INVERSE : PRINT " ILISZTER ACTIVE ": NORMAL
    : VTAB 4:L = PS + 1: GOSUB 21: PRINT "START OF INTEGER PROGRAM: ";A$
    :L = PE: GOSUB 21: PRINT "  END OF INTEGER PROGRAM: ";A$: REM OPTION
    AL TO CHECK APPROXIMATE ADDRESS LOCATION
99  PRINT : INPUT "PROGRAM NAME: ";H$(1): INPUT "  PROGRAMMER: ";H$(2): INPUT
    "          DATE: ";H$(3): REM REQUIRED FOR HEADER ON EACH PAGE
100  PRINT : PRINT "WANT DIFFERENT START ADDRESS ?": GET A$: IF A$ <  > "
    Y" GOTO 103
101  INPUT "    START ADDRESS (HEX): ";A$:D = 1:BT = 0: FOR K =  LEN (A$) TO
    1 STEP  - 1: FOR E = 1 TO 16: IF  MID$ (A$,K,1) =  MID$ (X$,E,1) THEN
    BT = D * (E - 1) + BT
102  NEXT E:D = D * 16: NEXT K: PRINT :P = BT - 1:L = BT: GOSUB 21: PRINT
    "    HEX ADDRESS = ";A$;"    CHANGE ?": GET A$: IF A$ = "Y" GOTO 101
103  PRINT : PRINT "NO LEFT MARGIN, WANT ONE ?": GET A$: IF A$ = "Y" THEN
     INPUT "  MARGIN SPACES: ";K: IF K > 0 AND K < 49 THEN M$ =  LEFT$ (
    BB$,K):LL = LL - K
104  REM REMINDER FOR PRINTER SET-UP
```

(continued)

line 25. Integer does not allow a byte count larger than 127. (The actual number is 255. The 127-byte limit (line 26) is for print-line reconstruction, usually longer than source-code line length.) A byte count that is too large will jump to the binary-insert routine at line 114. Line numbers up to 65535 will output whether they are actual line numbers or a chance byte-pair in binary. A test of number magnitude was included in an earlier version but then disregarded due to the large number of starting prompts.

Remark checking in lines 30 to 32 is part of the blank-line separation for REMs. Removing separation would delete all but the "D=0" statement; D must remain for line number printing.

Statements begin parsing in line 34. ASCII characters are restored for printing but control characters are uppercase between vertical bars. Source code rubouts are included to fill out lines in certain programs.[2]

Token parsing begins at line 39 with a test for unused tokens. The added space to the gather string at line 40 prevents a crash during a binary code test; a rare condition, but it was found in two listings.

Three programs were found with a FOR loop starting on an IF-true condition. Line 42 solves indenting and restoration on this rare case. Integer normally executes only one IF-true condition but, apparently, a FOR loop will execute until completed.

Two or More Iterators

The printout indent restoration of statements such as "NEXT J,K" is solved by the search routine in lines 45 to 49. Of several comma tokens, only decimal value 90 is the comma in a multiple-variable NEXT statement. This search and find will restore global indenting of FOR loops. It can also be patched into the original LISZTER to solve an oversight.[7]

Numbers Following You?

Some tokens allow following numeric constants. Integer BASIC flags a numeric constant with a $B0 to $B9 prefix (ASCII numbers 0 to 9). The test in lines 51 and 52 check for token and prefix, ignoring the prefix if it exists.

Line 53 builds the numeric constant string and gathers it in G$. Flow must return to line 34 afterwards. The next byte can be either a token or a character; variable names are ASCII characters.

The Final Print Line

Lines 55 to 80 form the output print line, splitting and indenting as in the original LISZTER. First-priority split is still a space, but second-priority split has a vertical bar added to line 69. Control characters seem to be used more in Integer. At this point they have been converted to upper-case letters between bars and will not upset printer control.

The complex print statement group in line 77 is solely for the italics capability of the Epson printer. A single-character-set printer can substitute a simple "PRINT M$; C$" for both GOSUBs and PRINTs.

Possible Binary?

An IF-true test at lines 26 or 39 indicates something is wrong with the Integer source code. More than likely it is due to embedding binary code with integer. The routine at lines 114 to 120 checks this condition.

Variable LA is made up of the address of each new source line number start. That address is converted to hexadecimal and printed with the "Possible Binary From" indicator. A search now begins for any byte group meeting the following: the group is below HIMEM, the group is less than 128 bytes long, and the end-of-line byte value is found from the first-byte address plus value. A successful search will print the byte group *last* address in hex to complete the indicator, then return to line 25 for a new source line number.

The indicator may be printed several times before a correct source line is found. The number of prints will be dependent on binary content but a correct Integer source line will always follow embedded binary.

A possibility is a bit error in memory that can yield another possible binary print line. An advantage is that a printout will show beginning and ending addresses for closer examination.

An "attached" binary program will terminate at highest available memory. The possible binary last print will indicate this as $95FF with standard DOS.

Alternatives

A purely Integer version of ILISZTER can be written by translation of the general structure. Page zero locations $69 through $6D can be used for

(continued)

```
105  HOME : INVERSE : PRINT " SET PAPER TO TOP OF FORM ": PRINT "
        THEN            ": PRINT "       TURN ON PRINTER      ": NORMAL : PRINT
     : GET A$
106  REM SET SCREEN WIDTH, TURN ON PROPER PORT
107  HOME : POKE 33,30: PR# 1
108  REM CONTROL CHARACTERS FOR MX-80 WITH "GRAPPLER" CARD.  CHR$(9)=CTRL
     -I, CHR$(27)=ESC
109  PRINT  CHR$ (9)"82N" CHR$ (27)"0" CHR$ (9)"I"
110  REM
111  REM    SET-UP TO START FIRST PRINT PAGE
112 LC = 6:PC = 1:D = 0: GOSUB 11: GOTO 25
113  REM POSSIBLE-BINARY INSERT/ADDITION ROUTINE
114 RF = 1: GOSUB 18:L = LA: GOSUB 21: GOSUB 6: PRINT M$;LB$;" >>> Possib
     le Binary from ";A$;" to ";
115  IF P > PE GOTO 121
116  IF B > 127 THEN  GOSUB 2: GOTO 115: REM BYTE-COUNT TOO LARGE
117 PT = P + B - 1:BT =  PEEK (PT): IF PT > PE GOTO 121
118  IF BT <  > 1 OR B < 5 THEN  GOSUB 2: GOTO 115: REM NO E-O-L OR BYTE-
     COUNT TOO SMALL
119  IF LA = (P - 1) THEN  GOSUB 2: GOTO 115: REM AVOID REPETITION; SOMEH
     OW THE POINTER DIDN'T ADVANCE
120 P = P - 1:L = P: GOSUB 21: PRINT A$:D = 0:G$ = "": GOTO 25: REM RETUR
     N TO LINE-NUMBER START
121 L = PE: GOSUB 21: PRINT A$
122  REM ENDING ROUTINE
123  GOSUB 4: GOSUB 17: PRINT M$;LB$;"End of Listing"
124  REM OPTIONAL STATISTICS
125  GOSUB 4: PRINT M$;"Program Length = ";(PE - PS);" Bytes,  Total of "
     ;TN;" Line Numbers": GOSUB 4: PRINT M$;(TS - TR);" Total Non-Rem Sta
     tements,  ";TR;" Total Remarks"
126  REM TURN OFF PRINTER, RESET SCREEN AND SHOW COMPLETION
127  PR# 0: POKE 33,40: HOME : VTAB 12: HTAB 10: INVERSE : PRINT " END OF
      ILISZTING ": NORMAL : END
128  REM  "ILISZTER" program to re-format INTEGER BASIC listing prints
129  REM    by Leonard H. Anderson     Version 2.8.8,  15 May 1982
130  REM          lower case and italics for MX-80 & "GRAFTRAX"
131  REM   Possible-Binary routines added to 2.8.1 (21 March 1982)
132  REM
133  REM DESCRIPTION OF VARIABLES:
134  REM
135  REM A$  TEMPORARY STRING, PARTLY FOR HEX CONVERSION
136  REM B   PROGRAM BYTE VALUE IN DECIMAL
137  REM BB$ 'BIG BLANK' STRING OF 48 SPACES
138  REM BC  BYTE-COUNT OF A LINE, DECIMAL
139  REM BT  TEMPORARY PROGRAM BYTE VALUE IN DECIMAL
140  REM CF  "IF" FLAG: SET ONLY ON "IF" FOLLOWED BY "FOR"
141  REM C$  CHARACTER AND TOKEN STRING TO BE PRINTED
142  REM D   TEMPORARY, PARTLY FOR 'DIRECTION'
143  REM E   TEMPORARY, PARTLY FOR SPLIT-LINE LIMITS
144  REM FF  "FOR" FLAG: 1 = "FOR" STARTED, 0 = NO "FOR"
145  REM FS  "FOR" INDENT SPACE COUNTER
146  REM G$  'GATHER' STRING TO BUILD A STATEMENT
147  REM H$  HEADER ARRAY FOR PRINT-PAGE TITLE
148  REM IM  INDENT SPACE MULTIPLIER
149  REM K   TEMPORARY
150  REM L   TEMPORARY, PARTLY FOR LOW-BYTE VALUE
151  REM LA  LINE NUMBER BEGINNING ADDRESS
152  REM LC  LINE COUNTER FOR PAGINATION
153  REM LL  LINE-LENGTH CONSTANT
154  REM LB$ 'LITTLE BLANK' STRING OF 8 SPACES
155  REM M$  LEFT MARGIN SPACING STRING
156  REM N$  LINE NUMBER STRING
157  REM P   POINTER TO PROGRAM BYTE, DECIMAL
158  REM PC  PAGE COUNTER FOR PRINT-PAGE HEADER
159  REM PE  INTEGER PROGRAM END ADDRESS, DECIMAL
160  REM PS  INTEGER PROGRAM START ADDRESS, DECIMAL
161  REM PT  TEMPORARY POINTER TO PROGRAM BYTE, DECIMAL
162  REM RF  "REM" FLAG: 1 = "REM" STARTED, 0 = NO "REM"
163  REM RS  "REM" INDENT SPACE COUNTER
164  REM SF  SPLIT-LINE FLAG: SET IF PRINT LINE MUST BE SPLIT
165  REM S$  SINGLE-SPACE STRING
166  REM TN  TOTAL LINE NUMBER COUNTER
167  REM TR  TOTAL REMARKS COUNTER
168  REM TS  TOTAL STATEMENTS COUNTER
169  REM T$  TOKEN STRING ARRAY
170  REM V   ARRAY FOR TOKEN EVALUATION:
171  REM         0 = NO BINARY NUMBER FOLLOWS TOKEN
172  REM         1 = A 3-BYTE BINARY NUMBER FOLLOWS
173  REM         2 = UNUSED/INTERNAL, DO NOT PRINT
174  REM X$  HEX CHARACTER STRING FOR CONVERSIONS
```

Make ILISZT

```
200      *    TEXT FILE GENERATOR FOR "ILISZT"
210      *    VERSION 3.0, 16 APRIL 1982  LHA

220  D$ = "|D|"
230  Print D$;"OPEN ILISZT"
240  Print D$;"WRITE ILISZT"

250      *    MAKE INTEGER LOMEM POINTER HOLD ENDING OF INTEGER PROGRAM

260  Print "POKE74,PEEK(76)"
270  Print "POKE75,PEEK(77)"

280      *    MAKE INTEGER HIMEM POINTER HOLD START OF INTEGER PROGRAM

290  Print "POKE76,PEEK(202)"
300  Print "POKE77,PEEK(203)"
310  Print "RUN ILISZTER"
320  Print D$;"CLOSE"
330  End
```

pointer re-arrangement as in the LISZT predecessor. Total code will probably exceed the 4.5K bytes of a "REM-less" ILISZTER in Applesoft. MAKE ILISZT can be either language; the created text file will be the same.

ILISZTER has successfully handled a 23K Integer program printout plus one program with two embedded binary code sections.

References

1. Apple Pugetsound Program Library Exchange "public domain" disks (members only). Printouts of 1057 programs fill three large loose-leaf notebooks; about a quarter are Integer.
2. "Higher Text" by Ron and Darrell Aldrich, *Call —A.P.P.L.E.* version. One Integer program has two binary embedments.
3. *MICRO on the Apple*, Volume 1, MICRO INK, pages 198-203.
4. *PEEKing at Call —A.P.P.L.E.*, Volume 2, pages 44-61, Apple Pugetsound Program Library Exchange, 1979.
5. *What's Where in the Apple!*, William F. Luebbert, MICRO INK. For address locations only.
6. "The Inspector," Omega Microware, Inc., is one example of a disk or memory byte-changer utility. Although the author has upper-/lower-case conversion on the keyboard, this utility was used to correct typos in ILISZTER's DATA statements.
7. "LISZT with Strings," Richard F. Searle, Don Cohen, Leonard H. Anderson, MICRO, May 1982, listing 2 on page 41. The easiest patch is a GOSUB in line 45 just after the "CF = 1" statement; the subroutine would look for a delimiter comma in ASCII, such as "BT = 44", to decrement the FOR spacer.

You may contact Mr. Anderson at 10048 Lanark St., Sun Valley, CA 91352.

MICRO

BASIC Macro Function for Cursor Control

by Kerry Lourash

BASIC Macro is a machine-language program similar in function to the macro option of some assemblers. It enables Cursor Control users to insert often-used statements with only two keys when typing BASIC programs. ERGO, a routine for all C1P users, eliminates the graphic character in error messages.

BASIC Macro and ERGO
require:
OSI C1P

As a C1P owner, I type in a lot of BASIC programs, mainly because neither OSI nor independent vendors have the programs I want. While I pounded my fingers to the bone and cursed my two-fingered typing speed, I wished for a utility similar to the macro function of some assemblers. After punching out "GOSUB8000:GOTO650" for the 20th time in a program, I was inspired to write BASIC Macro.

Macro is an extension of the Cursor Control program (MICRO 36:75). It lets you insert one of ten macros up to 70 characters long in a BASIC line with only two keystrokes (three, if you count CTRL R as two keys). If a phrase (such as GOSUB8000:GOTO650) occurs frequently in a program you're typing, store it in a BASIC line 0-9 (1 GOSUB8000:GOTO650). Now, as you encounter that phrase, hit CTRL R. A white block will appear. Type '1' and the phrase will be printed on the screen and stored in the input buffer. Should you type a line number that doesn't exist, Macro will wait for another number. If you type a letter, Macro assumes you've changed your mind about calling a macro, and exits. CTRL R stands for repeat.

When designing Macro, I had plans for a sophisticated phrase storage area with variable-length storage space. After I'd written the code to find and print the phrases, which was the lesser half of the program, I found that I'd used over half a page of memory. This approach was going to cost me well over the page of memory I had allotted for program and storage space! So I let BASIC keep track of the phrases.

To patch Macro into Cursor Control, change the input routine PATCH at location $1E0F to JMP $0222 instead of JMP $1E12.

Macro finds the BASIC line you specify, prints it on the screen, and stores it in the input buffer. If the addition of the phrase makes the line too long, the 'BEL' character is printed. To use BASIC lines 0-9 as storage space, it was necessary to teach Macro how to convert tokens to keywords, but the final program is still much shorter than my first attempt. The WINDUP routine finds the buffer count in the stack,

BASIC Macro Listing

```
0000                ;BASIC MACRO FOR CC
0000                PATCH=$1E0F
0000                   OK=$1F10
0222                   *=$0222
0222 C912    MACRO  CMP #$12      ;CTRL R?
0224 D061           BNE RESUME
0226 20101F         JSR OK        ;PRINT WHITE BLOCK
0229 2000FD  MAC    JSR $FD00     ;GET MACRO NUMBER
022C C93A           CMP #$3A      ;IF NOT A NUMBER
022E B057           BCS RESUME    ;THEN EXIT
0230 C930           CMP #$30
0232 9053           BCC RESUME
0234 E930           SBC #$30      ;ASCII TO BINARY
0236 8511           STA $11       ;LOOK FOR LINE #
0238 A900           LDA #0
023A 8512           STA $12
023C 2032A4         JSR $A432
023F 90E8           BCC MAC       ;TRY AGAIN
0241         ;
0241 A003           LDY #3        ;TO START OF LINE
0243 C8      FOUND  INY           ;NEXT CHAR.
0244 8497           STY $97       ;SAVE Y REGISTER
0246 B1AA           LDA ($AA),Y   ;GET CHAR.
0248 F035           BEQ WINDUP    ;QUIT IF NULL
024A 3007           BMI TOKEN     ;CONVERT IF TOKEN
024C A497    FND    LDY $97       ;RESTORE Y REGISTER
024E 206F02         JSR STORE
0251 D0F0           BNE FOUND     ;BRANCH ALWAYS
0253         ;
0253 38      TOKEN  SEC           ;FIND & CONVERT TOKEN
0254 E97F           SBC #$7F      ;TOKEN MINUS 7F
0256 AA             TAX           ;TOKEN INDEX IN A REG
0257 A0FF           LDY #$FF
0259 CA      TO     DEX
025A F008           BEQ T2        ;FOUND TOKEN IN TABLE?
025C C8      T1     INY           ;NO, NEXT LETTER
025D B984A0         LDA $A084,Y
0260 10FA           BPL T1        ;LOOP & GET NEXT CHAR.
0262 30F5           BMI TO        ;LOOP TO NEXT TOKEN
0264 C8      T2     INY
0265 B984A0         LDA $A084,Y   ;GET LETTER
0268 30E2           BMI FND       ;LAST LETTER OF TOKEN?
026A 206F02         JSR STORE
026D D0F5           BNE T2
026F         ;
026F A60E    STORE  LDX $0E       ;STORE CHAR. IN BUFFER
```

where it was stored at the start of the INPUT routine (the X register). Location $0E, the screen character counter, is loaded into the stack to update the buffer count.

For those unfortunates who have not been converted to Cursor Control, I whipped up a short patch to the stock output routine that prints C1P error messages correctly. As the output routine prints characters on the screen, ERGO checks every carriage return to see if it comes from the error message routine. If so, ERGO steps in and prints the second letter of the error message as a letter, not a graphics character. The stock carriage return/line feed is omitted to save space on the screen. To patch ERGO into the output routine, change the contents of the output vector to the start of ERGO ($021A = 22, $021B = 02).

You may contact Kerry Lourash at 1220 North Dennis, Decatur, IL 62522.

MICRO

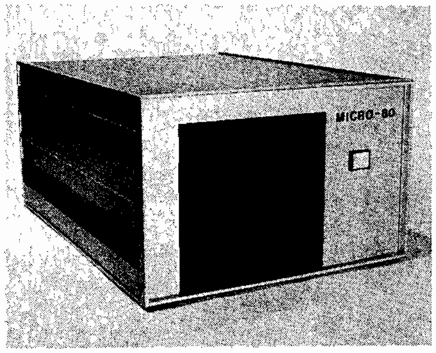
BASIC Macro Listing *(Continued)*

```
0271 E047              CPX  #$47
0273 B005              BCS   STO+1
0275 297F              AND  #$7F         ;ZERO HI BIT
0277 9513              STA   $13,X
0279 2CA907    STO     BIT   $07A9       ;BEL CHAR. IF >71
027C 4CE5A8            JMP   $A8E5       ;PRINT CHAR.
027F           ;
027F BA        WINDUP  TSX               ;UPDATE BUFFER COUNT
0280 A50E              LDA   $0E         ;LINE COUNT IN STACK
0282 9D0201            STA   $0102,X
0285 A901              LDA  #1           ;NON-PRINTING CHAR.
0287 4C121E    RESUME  JMP   PATCH+3     ;BACK TO CC
```

ERGO Listing

```
0000         ;       ERGO ROUTINE

0222         *=$0222
0222 C90D            CMP     #13      IS CHAR A  CR  ?
0224 D015            BNE EXIT
0226 8650            STX      $50     SAVE X REG.
0228 BA              TSX              GET STACK POINTER
0229 BD0501          LDA     $105,X   CALLING ADDRESS $A252?
022C C952            CMP     #$52
022E D007            BNE     NOERR
0230 BD0601          LDA     $106,X
             0233 C9A2       CMP      #$A2
0235 F007            BEQ     ERGO     YES, PRINT ERR MESS.
0237 A650    NOERR   LDX     $50      RESTORE A&X REGS.
0239 A90D            LDA     #13
023B 4C69FF  EXIT    JMP     $FF69    TO REGULAR OUTPUT

023E A650    ERGO    LDX     $50      RESTORE X REG.
0240 20E3A8          JSR     $A8E3    PRINT '?'
0243 BD64A1          LDA     $A164,X  FIND 1ST LETTER
0246 20E5A8          JSR     $A8E5    PRINT IT
003F BD65A1          LDA     $A165,X  FIND 2ND LETTER
024C 297F            AND     #$7F     ZERO HI BIT
024E 4C5FA2          JMP     $A25F    TO REG. ERR ROUTINE
```

NEW FROM D & N MICRO PRODUCTS, INC.



MICRO-80 COMPUTER

Z80A CPU with 4MHz clock and CP/M 2.2 operating system. 64K of low power static RAM. Calendar real time clock. Centronics type parallel printer interface. Serial interface for terminal communications, dip switch baud rates of 150 to 9600. 4″ cooling fan with air intake on back of computer and discharge through ventilation in the bottom. No holes on computer top or side for entry of foreign object. Two 8″ single or double sided floppy disk drives. IBM single density 3740 format for 243K of storage on each drive. Using double density with 1K sectors 608K of storage is available on a single sided drive or 1.2 meg on a double sided drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane for expansion. 48 pin buss is compatible with most OSI boards. Uses all standard IBM format CP/M software.

Model 80-1200 — $2995
2 8″ single sided drives, 1.2 meg of storage

Model 80-2400 — $3495
2 8″ double sided drives, 2.4 meg of storage

Option 001 — $ 95
Serial printer port, dip switch baud rate settings

Software available in IBM single density 8″ format.

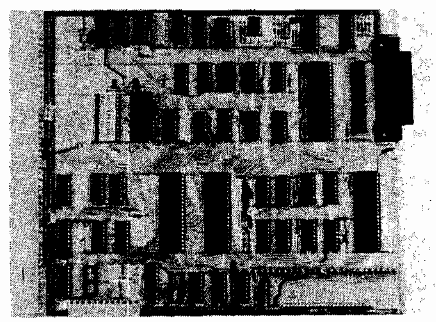
Microsoft		Digital Research		Micropro	
Basic-80	$289	PL/1-80	$459	Wordstar	$299
Basic Compiler	$329	Mac	$ 85	Mail-Merge	$109
Fortran-80	$410	Sid	$ 78	Spellstar	$175
Cobol-80	$574	Z-Sid	$ 95	Super Sort I	$195
Macro-80	$175	C Basic-2	$110	**Pascal**	
Edit-80	$105	Tex	$ 90	Pascal/MT +	$429
Mu Simp/Mu Math	$224	DeSpool	$ 50	Pascal Z	$349
Mu Lisp-80	$174	**Ashton-Tate**		Pascal M	$355
		dBase II	$595		

Convert almost any static memory OSI machine to CP/M® with the D & N-80 CPU Board.

Z80A CPU with 4MHz clock. 2716 EPROM with monitor and bootstrap loader. RS-232 serial interface for terminal communications or use as a serial printer interface in a VIDEO system. Disk controller is an Intel 8272 chip to provide single or double density disk format. 243K single density or 608K double density of disk storage on a single sided 8″ drive. A double sided drive provides 1.2 meg of storage. DMA used with disk controller to unload CPU during block transfers from the disk drives. Optional Centronics type parallel printer port complete with 10 ft. cable. Optional Real Time Calendar Clock may be set or read using 'CALL' function in high level languages. Power requirements are only 5 volts at 1.4 amps. Available with WORDSTAR for serial terminal systems.
INCLUDES CPM 2.2

D & N-80 serial	$695
D & N-80 serial w/Wordstar	$870
D & N-80 video	$695
Option001	$ 80

parallel printer and real time calendar clock



D & N-80 CPU BOARD

OTHER OSI COMPATIBLE HARDWARE

IO-CA10X Serial Printer Port — $125
Compatible with OS-65U and OS-65D software

IO-CA9 Parallel Printer Port — $175
Centronics standard parallel printer interface with 10 ft. flat cable

BP-580 8 Slot Backplane — $ 47
Assembled 8 slot backplane for OSI 48 pin buss

24MEM-CM9	**$380**	**24MEM-CM9F**	**$530**
16MEM-CM9	**$300**	**16MEM-CM9F**	**$450**
8MEM-CM9	**$210**	**8MEM-CM9F**	**$360**
BMEM-CM9F	**$ 50**	**FL470**	**$180**

24K memory/floppy controller card supports up to 24K of 2114 memory chips and an OSI type floppy disk controller. Available fully assembled and tested with 8, 16, or 24K of memory, with floppy controller (F). Controller supports 2 drives. Needs separated clock and data inputs. Available Bare (BMEM-CM9F) or controller only (FL-470). Ideal way to upgrade cassette based system

C1P-EXP Expansion Interface — $ 65
Expansion for C1P 600 or 610 board to the OSI 48 pin buss. Requires one slot in backplane. Use with BP-580 backplane

BIO-1600 Bare IO card — $ 50
Supports 8K of memory, 2 16 bit parallel ports may be used as printer interfaces. 5 RS-232 serial ports, with manual and Molex connectors

DSK-SW Disk Switch — $ 29
Extends life of drive and media. Shuts off minifloppy spindle motor when system is not accessing the drive. Complete KIT and manual

Disk Drives and Cables

8″ Shugart SA801 single sided — **$395**

8″ Shugart SA851 double sided — **$585**

FLC-6 6ft. cable from D & N or OSI controller to 8″ disk drive — **$ 69**

5 1/4″ MPI B51 with cable, power supply and cabinet — **$450**

FLC-5 1/4 8 ft. cable for connection to 5 1/4 drive and D & N or OSI controller, with data separator and disk switch — **$ 75**

Okidata Microline Printers

ML 82A Dot Matrix Printer — **$534**
120 CPS, 80/120 columns, 9.5″ paper width, friction or pin feed

ML 83A Same as 82A except — **$895**
16″ paper width, 132/232 columns with tractor feed

ML 84 Same as 82A except 200 CPS, — **$1152**
16″ paper width, 132/232 columns, 2K buffer, dot addressable graphics, with tractor feed

D & N Micro Products, Inc.
3684 N. Wells St.
Fort Wayne, Ind. 46808
(219) 485-6414

TERMS $2.50 shipping, Foreign orders add 15%.
Indiana residents add 4% sales tax.

ATARI Character Graphics from BASIC, Part 3

by Paul Swanson

You can remove the screen flicker by adding a short machine-language program to Atari's vertical blank interrupt routine.

Character Graphics
requires:
Atari 400/800

Last month I explained how to enable and use Atari's fine scrolling function (:). The only big problem was that the screen flickered a little because you had to shut off ANTIC, along with the display, in order to alter the horizontal scroll register.

There are several registers like that—you can't write to them while ANTIC is displaying a screen or you get strange effects. Most of these are taken care of by shadowing. However, the horizontal scroll register is not shadowed, so we need a different technique.

Shadowing

Shadowing is a method of updating video-related registers without interrupting the display in progress. Certain memory locations ("shadow" registers) are set aside to represent the actual video registers. When ANTIC completes the job of displaying one screen, it sends an interrupt signal to the 6502. Since ANTIC is not doing anything but waiting for the electron beam to return to the upper left corner of the screen, the 6502 has time to execute many instructions. Among the things accomplished during this vertical blank period is an update of the actual video registers from the contents of the shadow reigsters. This guarantees that all of the hardware registers are written while ANTIC is not drawing on the screen. At the end of the interrupt routine, the 6502 automatically returns to whatever it was doing before the interrupt occurred, so this process is almost invisible to the main program. This interrupt routine happens at the end of every sweep of the electron beam, or exactly sixty times per second.

The Vertical Blank Interrupt Routine

Every sixtieth of a second your program, whether in BASIC or machine language, gets interrupted for this special routine. Actually, there are two routines. The first one, which almost always runs, is called the immediate vertical blank interrupt routine. It takes care of all of the timers in the system, which includes the real time clock in locations 18 through 20 (decimal). It adds one each frame so that PEEK(20) + PEEK(19)*256 + PEEK (18)*65536 always reveals the elapsed time in sixtieths of a second.

The second routine is tacked on to the end of the first one. This second part is called the deferred vertical blank interrupt routine. You can easily stop this routine from running by setting the critical flag (a 1 into location 66). In addition to writing the shadowed information to the hardware registers, this second part also updates a few other timers, maintains the keyboard autorepeat and debounce functions, and reads and interprets the game controllers into special memory locations.

By altering two vector locations, you can replace or add to the existing interrupt routines. Each vector is a two-byte address stored in low, high order.

The vertical blank interrupt starts with a signal generated by ANTIC at the end of the display. This signal can be masked by the hardware register NMIEN (decimal location 54286). If the contents last written here were 64,

Listing 1: **Routine to shadow the fine scrolling registers. The JMP location xxxx will be the vector value at location $224. The shadow registers will be at locations $610 and $611.**

```
0600   AD 11 06    LDA   $611
0603   8D 05 D4    STA   $D405
0606   AD 10 06    LDA   $610
0609   8D 04 D4    STA   $D404
060C   4C xx xx    JMP   $xxxx
```

Listing 2

```
1 REM *** Custom Character Set ***
2 REM *** Vertical Blank       ***
3 REM ***   Interrupt routine  ***
4 REM
5 REM *** Program by...        ***
6 REM ***         Paul S. Swanson ***
7 REM
8 REM
9 REM --- Calc. position in mem. ---
10 DIM S$(1024)
20 A=ADR(S$)
30 B=INT(A/512+1)*2
40 CBASE=B*256-A+1
47 REM
48 REM
49 REM --- Clear S string ---
50 S$(1)=CHR$(0)
60 S$(1024)=CHR$(0)
70 S$(2)=S$(1)
77 REM
78 REM
79 REM --- Move standard set down ---
80 FOR I=0 TO 511
90 S$(CBASE+I,CBASE+I)=CHR$(PEEK(I+57344))
100 NEXT I
107 REM
108 REM
```

(continued)

Listing 2 *(continued)*

```
109 REM --- Set # to character ---
110 FOR I=24 TO 31
120 READ N
130 S$(I+CBASE,I+CBASE)=CHR$(N)
140 NEXT I
147 REM
148 REM
149 REM --- GR.2 - No text window ---
150 GRAPHICS 18
152 GOSUB 500
157 REM
158 REM
159 REM --- Find Display List ---
160 DLIST=PEEK(560)+PEEK(561)*256
162 SLOC=PEEK(DLIST+4)+PEEK(DLIST+5)*256
167 REM
168 REM
169 REM --- Set scroll enables ---
170 POKE DLIST+3,PEEK(DLIST+3)+48
180 FOR I=6 TO 16
190 POKE DLIST+I,PEEK(DLIST+I)+48
200 NEXT I
207 REM
208 REM
209 REM --- Initialize position ---
210 VPOS=96
220 HPOS=80
222 POKE 756,B
224 WING=1
226 S=14
227 REM
228 REM
229 REM --- Draw character in position ---
230 V=INT(VPOS/16)
232 IF WING=1 THEN SOUND 0,10,0,6
240 VSCROL=VPOS-V*16
250 H=INT(HPOS/8)
260 HSCROL=HPOS-H*8
262 IF WING=1 THEN WING=2:S$(CBASE+25,CBASE+25)=CHR$(0):S$
(CBASE+26,CBASE+26)=CHR$(231):GOTO 266
264 WING=1:S$(CBASE+25,CBASE+25)=CHR$(195):S$(CBASE+26,CBASE+26)
=CHR$(36)
266 P1=V*24+H
270 IF P<>P1 THEN POKE SLOC+P,0
280 POKE 1552,HSCROL
290 POKE 1553,15-VSCROL
291 IF P<>P1 THEN P=P1:FOR I=1 TO 3:NEXT I
292 POKE SLOC+P,3
294 SOUND 0,10,0,2
297 REM
298 REM
299 REM --- Read Joystick ---
300 OLDS=S:S=STICK(0)
310 IF S=15 THEN S=OLDS
320 VMOVE=0
330 HMOVE=0
340 IF S=9 OR S=13 OR S=5 THEN VMOVE=2
350 IF S=10 OR S=14 OR S=6 THEN VMOVE=-2
360 IF S>4 AND S<8 THEN HMOVE=1
370 IF S>8 AND S<12 THEN HMOVE=-1
380 IF VMOVE+VPOS>=0 AND VMOVE+VPOS<191 THEN VPOS=VPOS+VMOVE
390 IF HMOVE+HPOS>=0 AND HMOVE+HPOS<192 THEN HPOS=HPOS+HMOVE
400 IF VMOVE=2 THEN WING=2
410 GOTO 230
497 REM
498 REM
499 REM --- SET UP VBLANK ROUTINE ---
500 FOR I=1 TO 13
510 READ N
520 POKE 1535+I,N
530 NEXT I
540 POKE 66,1
550 POKE 1549,PEEK(548)
560 POKE 1550,PEEK(549)
570 POKE 548,0
580 POKE 549,6
590 POKE 66,0
600 RETURN
1000 DATA 0,195,36,24,24,36,0,0
1010 DATA 173,17,6,141,5,212,173,16,6,141,4,212,76
```

the interrupt will happen. Writing a zero will prevent the interrupt.

If the signal is not masked by NMIEN, the 6502 is interrupted and a branch to the immediate vertical blank interrupt routine occurs. This updates the real time clock, processes the attract mode, and maintains a special system timer, CDTMV1 (refer to Atari manuals).

When the immediate mode vertical blank routine is completed, the flag CRITIC (memory location 66) is checked, as is the processor interrupt bit I. If either is non-zero, the interrupt sequence is terminated with a return to the main program 6502 instruction RTI. Otherwise, the interrupt routine continues with the deferred portion.

This second part moves all the shadow registers into the hardware registers, updates a few other system timers, and decodes the results read from the game controllers. When it has finished, it branches through the vector at location 548 (decimal — 2 bytes). Unless you alter it, this location points to an RTI routine.

Every time there is a vertical blank interrupt, the computer uses the address at location 546 to find the immediate vertical blank interrupt routine. It uses the address at location 548 only when the critical flag and the I bit are not set. BASIC cannot access the I bit directly, but it can write to the critical flag with a POKE.

Your Own Routine

To shadow your fine scrolling values so that you don't interrupt the screen while it is being drawn, you must add on your own machine-language routine. This can be done by altering the pair of memory locations called VVBLKD (Vector for Vertical BLanK Deferred routine — this is the one at location 548).

First you must write your routine in machine language and store it in a fixed place in memory. In the sample program, the routine requires 15 bytes and starts at location $600 (1536 in decimal). A BASIC POKE routine may be used to install this code.

Since BASIC is so slow, you must make allowances for certain odd occurrences. What happens if a vertical blank routine tries to use a vector between the time you write one byte and the time you write the next byte? Your program crashes! To get around this potential catastrophe, you can shut the second part of the vertical blank interrupt routine off so that it does not even look at this vector. This is accomplished by setting the critical flag (a 1 into location 66). You then make the changes to the vector at location 548, then restore the critical flag with a zero into location 66. This needs to be done only once — while you change the contents of the vector.

If you want to add to the beginning of the immediate vertical blank interrupt, first POKE 54286 (NMIEN) with a zero. This disables the vertical blank interrupt. Next, make the appropriate changes to the vector at 546, and then POKE 54286 with a 64 to re-enable the vertical blank interrupt.

Listing 1 shows the routine used to form shadow registers for the fine scrolling hardware registers. You must POKE the first 13 bytes into memory, then copy locations 548 and 549 into bytes 14 and 15. This causes the routine to jump to the location that the vertical blank interrupt routine normally jumps to on completion. To get the normal interrupt routine to jump to your routine in the first place, POKE a zero in location 548 and a 6 in location 549. This puts 1536 ($600) into the VVBLKD locations.

The machine-language program takes the values in locations $610 and $611 (decimal 1552 and 1553) and stores them into the horizontal and vertical scroll hardware registers. Then it jumps back into the vertical blank interrupt routine where we first interrupted it. Locations 1552 and 1553 (decimal) now act as shadow registers for horizontal and vertical scroll values, respectively.

The BASIC Program

Listing 2 enhances the program presented in last month's article by adding the shadowing routine. The machine-language routine is converted to decimal and included as line 1010 in a DATA statement. A new subroutine, called at line 152, has been added at line 500. It first READs the machine-language routine into the locations chosen. Line 540 turns off the deferred vertical blank interrupt routine so that the computer will not try to branch through the vector that needs changing. Lines 550 and 560 copy the current contents of that vector into the JMP instruction of our machine-language routine and then change the vector to point to location $600 (1536 decimal). Line 590 turns off the flag, enabling the new routine, and RETURNs.

Note that the second DATA statement READ happens after the READ for the first one. If you rearrange the program, make sure you pay attention to the DATA pointer so that you don't insert the shape of the bird where the machine-language routine should go.

There are a few other changes made to the portion that scrolls the bird. Lines 266 through 292 are altered. Line 266 now calculates the new position. If it is the same as the old position except for the scrolling values, the character is not erased. It is erased only when the position value has changed; this limits the flickering substantially.

Lines 550 and 560 are altered to POKE into the new shadow registers. ANTIC is not turned off at all. Line 291 is added to update the position value P and cause a slight delay if the position value were changed. This delay guarantees that there has been at least one vertical blank interrupt routine since the new values were written to the shadow registers. The hardware registers are updated before line 292 is executed. Line 292 puts the bird on the screen in the position indicated by P. If the position were not altered, this line doesn't actually do anything. If the position value has been changed, it draws the bird in the new position.

There is still a slight flicker every once in awhile, but this will not be noticeable if other things are happening at the same time. The only way to eliminate the flicker altogether is to use machine language to update the bird as well. By using shadow registers you could write a vertical blank interrupt routine that would take your position values and reduce them to the screen position and the fine scrolling values. BASIC is a much easier language in which to create programs, but a little machine language now and then can help smooth out the rough edges. If you can get away with routines as short as the one in listing 1, it is certainly worth it.

What To Do With This Information

The character graphics example here was intended for instruction only. However, the shadowing described in this article, combined with the custom character set and fine scrolling described in parts 1 and 2, needs only to be combined with a little imagination to produce some elegant software.

Paul Swanson is our Atari columnist. You may contact him at 97 Jackson Street, Cambridge, MA 02140.

MICRO™

APPLESOFT GOTO/GOSUB Checking Routine

by Peter J.G. Meyer

This 194-byte machine-language routine will check all GOTO and GOSUB references in an Applesoft program and display any that refer to non-existent lines. The source program also demonstrates how to make use of the machine-language subroutines available in the Applesoft Interpreter.

GOTO/GOSUB Checker
requires:
Apple II with Applesoft

In a previous article [MICRO 43:101] I presented a short assembly-language program for a utility that would display the bytes constituting a specified line in an Applesoft program. That utility was constructed using eight machine-language subroutines available in the Applesoft Interpreter and the Apple Monitor.

In this article I will use two of those routines [LINGET and FNDLIN] together with six others to construct a utility for checking the GOTO and GOSUB references in an Applesoft program. This utility does the useful task of going through an Applesoft program looking for GOTOs and GOSUBs. When it finds one, it searches the program for the referenced line. If the line does not exist, it displays the offending statement with the line number in which it occurs.

To understand the assembly-language program presented here, it is necessary only to understand the structure of an Applesoft line in RAM and the function of the eight Applesoft subroutines that are employed. Of course, it also helps to know a little about 6502 assembly-language programming, but novices should not be deterred.

An Applesoft program line, as it exists as bytes in RAM, consists of four consecutive parts:

1. Two bytes containing the address of the following line (low byte then high byte, as usual).
2. Two bytes containing the line number in hexadecimal.
3. The tokenized text of the line (in which, for example, GOTO is represented by the token byte $AB).
4. The end-of-line token, $00.

The text of the line may consist of several statements. In this case each statement (except the last) is followed by the end-of-statement token, $3A (which is the byte used as the ASCII representation of the colon, ':'). The final statement in the line is followed, not by an end-of-statement token, but by the end-of-line token.

For example, suppose the program line "10 IF A = 0 THEN GOSUB 120: ON B GOTO 340,560" is the first in a program. It will (normally) occur at $0801 and be represented in RAM as shown in figure 1.

Good programming style is simply knowing what you want to do, and stating clearly how to do it. In this case, what we want to do is as follows. For each line in the Applesoft program:

1. Inspect the line for GOTOs ($AB tokens), THENs ($C4 tokens), and GOSUBs ($B0 tokens).

Figure 1

801 - 1A 08	pointer to next line
803 - 0A 00	"10" in hexadecimal
805 - AD 41 D0 30	"IF A = 1"
809 - C4 B0 31 32 30 3A	"THEN GOSUB 120:"
80F - B4 42 AB 33 34 30 2C 35 36 30	"ON B GOTO 340,560"
819 - 00	end-of-line token

Listing 1

```
;*******************************************
;*
;*      GOTO/GOSUB CHECKER
;*
;*      BY PETER MEYER
;*
;*      APRIL 1982
;*
;*******************************************
;
;*      APPLESOFT SUBROUTINES
;
CHRGET   EPZ $B1
CHRGOT   EPZ $B7
FNDLIN   EQU $D61A
STXTPT   EQU $D697
LINGET   EQU $DA0C
CRDO     EQU $DAFB
STROUT   EQU $DB3A
LINPRT   EQU $ED24
;*      STANDARD ZERO PAGE LOCATIONS
;
LINNUM   EPZ $50
TXTTAB   EPZ $67
TXTPTR   EPZ $B8
;
;*      SPECIAL ZERO PAGE LOCATIONS
;
TOKEN    EPZ $F9
```

2. If none are found, continue with the next line, until the end of the program is reached.
3. If a GOTO, THEN, or GOSUB token is found, read the line number following the token.
4. Search through the program for a line so numbered.
5. If the line is found, continue inspecting the current line for GOTOs, THENs, and GOSUBs.
6. If no such line is found, report this fact by displaying the current line number and the offending GOTO, THEN, or GOSUB statement (then continue the inspection).

To go through RAM one byte at a time, Applesoft has the subroutine CHRGET, which is located on page zero (at $B1). This routine makes use of the two-byte pointer called TXTPTR (at $B8,B9). TXTPTR is usually pointing to a byte somewhere in the Applesoft program in RAM. The effect of CHRGET is to advance TXTPTR to the next byte and to load that byte into the accumulator (setting certain flags along the way). Thus, by repeatedly invoking CHRGET we can go through each program line looking for GOTO and GOSUB tokens. (CHRGOT, at $B7, is CHRGET without the initial advance of TXTPTR. It simply loads the accumulator with whatever byte TXTPTR is pointing to.)

Having found a GOTO, THEN, or a GOSUB token, we can then use the subroutine LINGET (at $DA0C) to read the line number and place it (in hexadecimal form) in the zero-page location LINNUM ($50,51). We can use LINGET for this purpose because this is precisely what LINGET was designed to do.

To help you search through a program to find a line whose number is at LINNUM, there is the routine FNDLIN (at $D61A). When this routine returns, the carry flag is set if such a line was found, otherwise the carry flag is clear. In the latter case we procede using CHRGET to look for further GOTOs and GOSUBs.

If FNDLIN returns with the carry flag set, then we have found a reference to a non-existent line and a report to this effect is in order. This report only needs to consist of 1. the number of the line containing the offending statement, 2. the word GOTO, THEN, or GOSUB, followed by 3. the number of the non-existent line referred to.

For printing numbers we have the

Listing 1 *(continued)*

```
             LN1      EPZ $FA
             LN2      EPZ $FC
             ;
             ;*     OTHER LOCATIONS
             ;
             DOS'WS   EQU $3D0            ;DOS WARM START VECTOR
             SPEAKER  EQU $C030
             ;
             ;******************************************
             ;
                      ORG $300            ;OR ANYWHERE CONVENIENT
0300         BEGIN:
0300 20 FB DA         JSR CRDO            ;PRINT <CR>
0303         ;SET TXTPTR TO BYTE PRECEEDING LINK FIELD OF FIRST LINE
0303 20 97 D6         JSR STXTPT
0306         NEXTLINE:
0306 20 B1 00         JSR CHRGET
0309 A0 01            LDY #1              ;END-OF-PROGRAM DOUBLE 00
030B B1 B8            LDA (TXTPTR),Y      ;REACHED YET?
030D D0 06            BNE SAVLINNO        ;IF NOT
030F 20 FB DA         JSR CRDO            ;PRINT FINAL <CR>
0312 4C D0 03         JMP DOS'WS          ;BACK TO BASIC
0315         SAVLINNO:
0315         ;IN CASE WE NEED TO PRINT IT LATER
0315 C8               INY
0316 B1 B8            LDA (TXTPTR),Y
0318 85 FA            STA LN1
031A C8               INY
031B B1 B8            LDA (TXTPTR),Y
031D 85 FB            STA LN1+1
031F         ;ADVANCE TXTPTR TO FIRST BYTE IN TEXT OF LINE

031F A5 B8            LDA TXTPTR
0321 18               CLC
0322 69 03            ADC #3
0324 85 B8            STA TXTPTR
0326 90 02            BCC GOTHRULN
0328 E6 B9            INC TXTPTR+1
032A         GOTHRULN:
032A         ;INSPECTING EACH BYTE IN TURN
032A 20 B1 00         JSR CHRGET
032D C9 00            CMP #0              ;END-OF-LINE TOKEN?
032F F0 D5            BEQ NEXTLINE        ;IF SO
0331 C9 C4            CMP #$C4            ;'THEN' TOKEN
0333 D0 0F            BNE NEXT
0335 A0 01            LDY #1
0337 B1 B8            LDA (TXTPTR),Y
0339 38               SEC
033A E9 30            SBC #$30
033C C9 0A            CMP #$0A
033E B0 EA            BCS GOTHRULN
0340 A9 C4            LDA #$C4            ;'THEN' TOKEN
0342 D0 08            BNE STORE           ;ALWAYS
0344 C9 AB   NEXT     CMP #$AB            ;'GOTO' TOKEN
0346 F0 04            BEQ STORE
0348 C9 B0            CMP #$B0            ;'GOSUB' TOKEN
034A D0 DE            BNE GOTHRULN
034C 85 F9   STORE    STA TOKEN
034E         READLNNO:
034E 20 B1 00         JSR CHRGET          ;ADVANCE TXTPTR TO LINE NO.
0351 20 0C DA         JSR LINGET          ;READ LINE NO.,STORE IN LINNUM
0354 A5 50            LDA LINNUM
0356 A4 51            LDY LINNUM+1
0358 85 FC            STA LN2             ;SAVE LINNUM IN LN2
035A 84 FD            STY LN2+1
035C AD 30 C0         LDA SPEAKER         ;EACH CLICK MEANS A PROG SEARCH
035F 20 1A D6         JSR FNDLIN          ;SEARCH PROGRAM FOR A LINE
0362 B0 30            BCS CHKCOMMA        ;IF LINE FOUND
0364         LINNOTFD:
0364 20 FB DA         JSR CRDO            ;PRINT <CR>
0367 A5 FB            LDA LN1+1
0369 A6 FA            LDX LN1
036B 20 24 ED         JSR LINPRT
036E A5 F9            LDA TOKEN
0370 C9 C4            CMP #$C4            ;'THEN' TOKEN
0372 D0 07            BNE NEXT1
0374 A9 B9            LDA #THEN
0376 A0 03            LDY /THEN
0378 4C 8A 03         JMP PRINT
037B C9 B0   NEXT1    CMP #$B0            ;'GOSUB'
037D F0 07            BEQ NEXT2
037F A9 A6            LDA #GOTO
0381 A0 03            LDY /GOTO
0383 4C 8A 03         JMP PRINT
0386 A9 AF   NEXT2    LDA #GOSUB
0388 A0 03            LDY /GOSUB
038A 20 3A DB PRINT   JSR STROUT          ;PRINT GOTO OR GOSUB
```

Applesoft routine LINPRT (at $ED24), which prints, in decimal form, the hexadecimal number whose high byte is in the accumulator and whose low byte is in the X-register. For printing text we have the routine STROUT (at $DB3A), which will print the string pointed to by the Y-register (high byte) and the accumulator (low byte). (The string must be terminated by a $00 or a $22.)

Thus, Applesoft provides us with all the routines we need for the job. With a good assembler and some attention to detail, these can be put together to produce a machine-language routine to perform the required task. The source program in listing 1 demonstrates how this can be done.

Once assembled and BSAVEd, this utility is used as follows: LOAD your program into RAM and BRUN the routine or, if it is already installed, simply CALL it. Line references in ONERR GOTOs and GOSUBs will also be checked, as will all line references (not just the first) in ON X GOTOs and GOSUBs.

Listing 1 *(continued)*

```
038D A5 FD              LDA LN2+1
038F A6 FC              LDX LN2
0391 20 24 ED           JSR LINPRT         ;PRINT LINE REFERRED TO
0394           CHKCOMMA:
0394           ;IN CASE OF MULTIPLE GOTO,OR GOSUB
0394 20 B7 00           JSR CHRGOT
0397 C9 2C              CMP #$2C           ;COMMA?
0399 F0 B3              BEQ READLNNO       ;IF SO
039B A5 B9              LDA TXTPTR+1       ;DECREMENT TXTPTR IN PREP
039D D0 02              BNE NEXT3          ;FOR NEXT USE OF CHRGET
039F C6 B9              DEC TXTPTR+1
03A1 C6 B8     NEXT3    DEC TXTPTR
03A3 4C 2A 03           JMP GOTHRULN
03A6           ;
03A6           ;*******************************************
03A6           ;*  STRINGS
03A6 20 20 20  GOTO     .DA '   GOTO "'
03A9 47 4F 54
03AC 4F 20 22
03AF 20 20 20  GOSUB    .DA '   GOSUB "'
03B2 47 4F 53
03B5 55 42 20
03B8 22
03B9 20 20 20  THEN     .DA '   THEN "'
03BC 54 48 45
03BF 4E 20 22
03C2                    END
```

Peter Meyer is the author of Agenda Files, from Special Delivery Software, and Routine Machine, recently released by Southwestern Data Systems. He is currently designing applications software in Europe. You may contact him at 55 Sutter St., Suite 608, San Francisco, CA 94104.

MICRO

Chances are, when you bought your first disk drive, it was an Apple. Now that you're ready for a second, take a look at Quentin.

Our Apple*-Mate™ 5¼" Disk Drive is fully software transparent with Apple's DOS 3.3 operating system in full and half track operation.

Add it to your present drive for greater capacity and faster access. Just plug it in and go to work.

And the Apple-Mate has these High Performance advantages:

ON TRACK HEAD SEEK

A precision lead screw positions the head onto the correct track. Time-consuming retries and disk-to-disk copying errors are virtually eliminated.

SIEMENS† DISK DRIVE

The apple-beige unit is built around the highly reliable Siemens system with over 10,000 lifetime hours. Shielded connecting cable also attached.

LONG TERM DEPENDABILITY

MTBF (Mean Time Between Failures)—8,500 power-on hours, and the unit has a one-year warranty.

COUNT ON QUENTIN FOR QUALITY

Quentin Research was building disk systems for the computer industry when Apple was a little bud on the big computer tree. We're known for product reliability and stand behind every system we sell you.

But the best news may be the price—only $335.00 (40 tracks).

A special introductory offer when you order Apple-Mate directly from us.

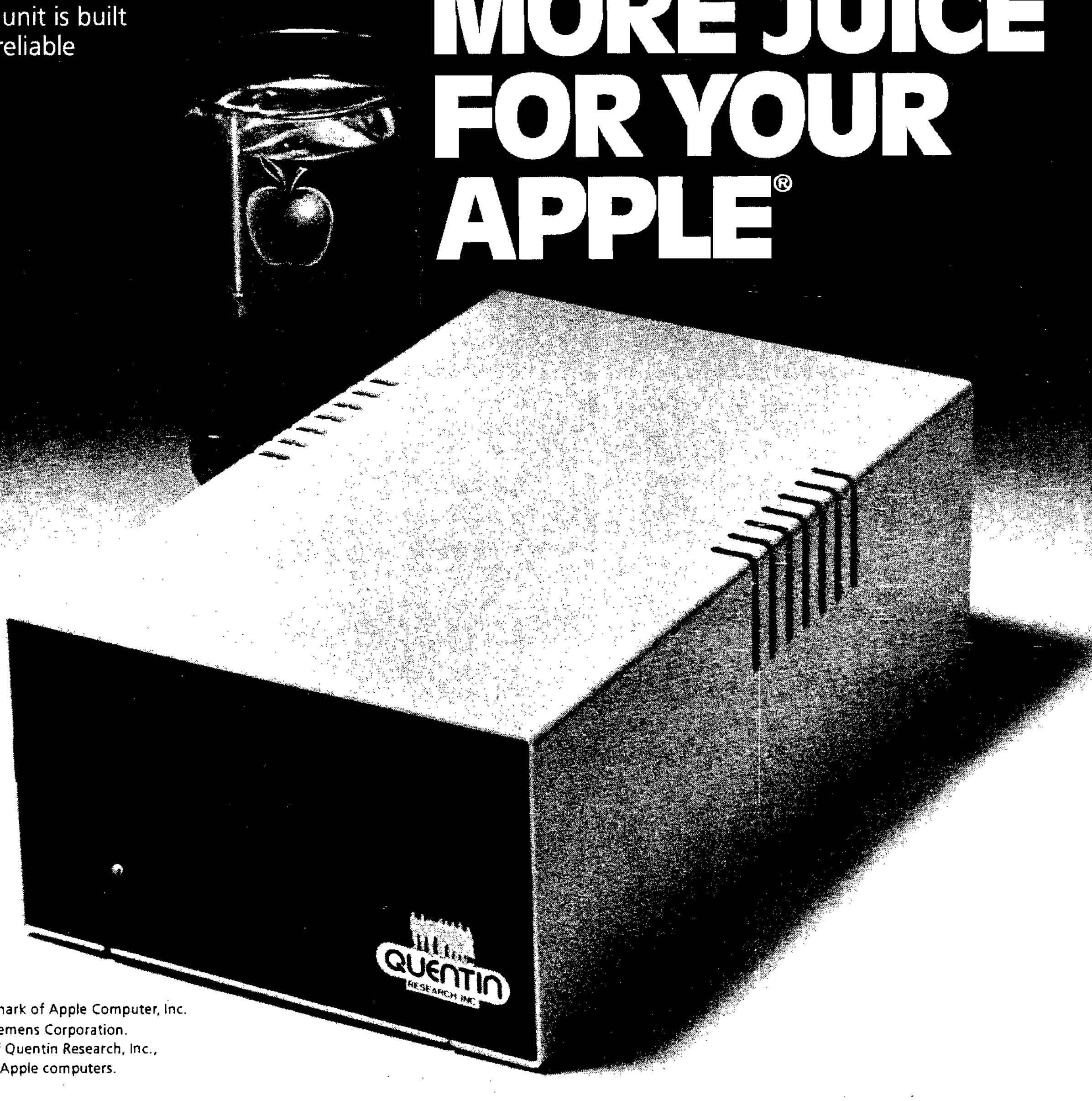
So when you're ready to boost the juice on your Apple, add-on the Quentin Apple-Mate.

To order: Check, money order, Visa or Mastercard number. Calif. residents add 6% sales tax. Allow one week delivery.

QUENTIN RESEARCH, INC.

19355 Business Center Drive
Northridge, California 91324
(213) 701-1006

MORE JUICE FOR YOUR APPLE®



Special Introductory Price: $335.00

® Apple is a registered trademark of Apple Computer, Inc.
† Siemens is a trademark of Siemens Corporation.
*Apple-Mate is a trademark of Quentin Research, Inc., which does not manufacture Apple computers.

MICRO™

CoCo Bits

By John Steiner

This month's CoCo Bits re-examines the single disk COPY command. In addition, I have noted a few CoCo-related news items. One item I did not mention last month regards the transfer of machine-language files to disk. Before loading the routines into memory, be sure to reserve enough memory space so BASIC will not overwrite your program. Also, before loading and executing the modified BEDLAM from disk, a CLEAR 200, 16384 will protect the program from BASIC. Without this command, the program seems to execute properly but does not print the opening message.

As I mentioned last month, the single disk COPY command *is* available and will *not* destroy a program that is in memory (like DSKINI and BACKUP). This opens the door to a useful routine for selective backup of program and data files. The backup command is appropriate for archives and duplication purposes. COPY is useful when only a few files require transfer, or if program data must be transferred to a disk without destroying already existing files.

If several files must be transferred, however, it is tiresome to enter the files one by one using COPY "filename/ext". The program in listing 1 provides a selective backup routine. It reads the disk directory track and stores all the program names in a string array. The array holds up to 68 file names, the maximum number a CoCo disk can hold. After reading the filenames, each name is presented. Pressing "Y" invokes the COPY command and the file is read into memory. You are prompted to switch disks, and if all goes well, told that the copy is complete. If you don't wish to copy a file press any other key. The next file in line is then presented for your decision. Be sure to reinstall your source disk before pressing "Y".

In addition to the COPY command, the simple program makes use of another powerful disk command. DSKI$ is used in a loop to read the sectors in the directory track. It is the only BASIC command that can directly read the directory. The routine that reads and stores the filenames is modified from the routine provided on page 62 of the COCO disk manual. By the way, there is a slight error in the routine that will cause it to miss several files. Line 60 reads FOR N = 1 TO 7; it should read FOR N = 0 TO 7.

The selective backup program routine uses several small arrays to read and identify the files that exist on a particular disk. Upon execution of line 160, the array FI$ contains the filenames of the program on the disk. Lines 170 to 230 present the filenames and invoke the copy command if necessary. This routine has saved me a lot of time and hassle.

A Color Computer user's group has been formed in the Toronto, Ontario, Canada area. If you are interested in joining, you may contact Patricia Jackson at (416) 425-1116. Call week days after 6:00 p.m., or on the weekend. There is also a user's group in the Fargo, North Dakota area. Contact me and I will put your name on the meeting notice mailing list. Anyone wishing to pass along similar information can contact me directly at the address shown below. It will take two to three months for your notice to appear in MICRO.

Rumors are that Tandy has signed an agreement with a group of RCA distributors to market the Color Computer in retail outlets not handling Radio Shack products. The new Color Computer will have a different color case and new name. If you have more details on this, or any other news regarding CoCo, pass it along.

Recently, I received an interesting musical program cassette. The classical rendition with four-voice organ music is the highest quality music routine I have heard, and I was impressed with the thought that most programmers are not using CoCo's sound abilities to their fullest. Several musical selections are available from Classical Software, 8931 Comanche Road, Longmont, Colorado 80501. They plan to announce a music editor with four-part tonal structure that will allow the user to enter and play notes directly from sheet music.

I own one of the early model Color Computers (serial number 337) and follow news about the Radio Shack 32K

Listing 1: COPY

```
10 CLS : PRINT@4,"SELECTIVE BACKUP PROGRAM"
20 PRINT@40,"BY JOHN STEINER"
30 PCLEAR 1
40 CLEAR 2000 : DIM FI$(67)
50 FOR X = 3 TO 11
60 DSKI$ 0,17,X,A$,B$
70 C$=A$ + LEFT$(B$,127)
80 N$(0)=LEFT$(C$,8)
90 EX$(0)=MID$(C$,9,3)
100 FOR N=0 TO 7
110 N$(N)=MID$(C$,N*32+1,8)
120 EX$(N)=MID$(C$,9+N*32,3)
130 IF LEFT$(N$(N),1)<>CHR$(0) AND LEFT$(N$(N),1)<>CHR$(255)
 THEN FI$(K)=N$(N)+" /"+EX$(N) : K=K+1
140 NEXT N
150 NEXT X
160 CLS:PRINT@64,"ENTER Y TO COPY"
170 FOR J=0 TO K
180 PRINT@224,FI$(J)
190 Z$=INKEY$ : IF Z$="" THEN 190
200 IF Z$="Y" THEN COPY FI$(J)
210 IF Z$="Y" THEN CLS : PRINT@224, FI$(J) " COPIED" : FOR I=1 TO 400
 : NEXT I
220 IF Z$="Y" THEN PRINT@0,"PLEASE REINSERT SOURCE DISK"
230 NEXT J.
```

CoCo Bits *(continued)*

modifications. I have wanted to upgrade to the new version for a while, but have not wanted to be without CoCo for the time it would take to make the change. I did increase memory capacity by piggy-backing existing memory with 16K chips. It is a relatively inexpensive procedure and works well, giving fewer OM errors. One of the major disadvantages of this modification is that Radio Shack is replacing the early boards with an updated processor board and 64K RAM chips. The 64K chips are permanently wired making the upper 32K bank inaccessible. A few simple changes allow you to restore the upper bank and deselect the ROMs that normally reside there. The user can then load another DOS, modify BASIC, or change the entire character of CoCo. When Radio Shack changed the memory chips, the company had to issue a new Color BASIC ROM. Color BASIC 1.1, in addition to checking for and using 32K, has a few of the previous bugs removed. The 1.1 ROM will send 8-bit serial data to the printer port. This allows CoCo to send graphics or special characters to the printer without loading Tandy's PTFX program.

I am interested in hearing from anyone who has modified a Color Computer to 64K without converting to the E board. I would also like to hear from FLEX and OS-9 users who successfully run their programs on CoCo. The added power and software compatability is a major step for Color Computer programmers.

Next month, in addition to CoCo news, I will discuss some books available for Color Computer users. I will also take a look at medium- and high-resolution graphics modes available in Extended BASIC.

You may contact the author at 508 Fourth Avenue NW, Riverside, ND 58078.

MICRO

MICRO™

From Here to Atari

By Paul S. Swanson

Atari News

I was pleased to see that Atari, Inc., recently established two regional software acquisition centers located in Cambridge, Massachusetts and London, England. The centers were set up to acquire software by contracting out for specific programs, or by buying software that has already been developed independently. more centers are planned for the future; I'll let you know where they will be as soon as Atari annouces that information.

Technical Tidbits

Code conversion is required in two areas when you're programming the Atari. The "normal" character code, called ATASCII, is a variation of ASCII. There are two other character codes used by the system. One is used to write characters to the screen. The screen handler does this conversion automatically when you PRINT to the screen, but if you use your own routines and put the characters directly on the screen with POKE or a similar method, you need to convert to this screen code.

The operating system manual includes a table that shows you the correspondence between ATASCII and the screen code (which they call the "Internal Code"). You can form a look-up table if you want by using a 256-byte string. Set it up so the value to POKE is the ASC(value of the byte in the string found at AVAL + 1, where AVAL is the ASC(value of the ATASCII character to be displayed.

An alternative approach, which consumes less memory than the look-up table, is using dependent IF statements. Using N as the ATASCII value to display:

```
FLAG = INT(N/128):N = N − FLAG + 64:
If N > 95 THEN N = N − 96: IF N > 64
THEN N = N + 32
```

After you execute that one line of code (it must be in one program line), POKE the screen location with N + FLAG. FLAG will equal 128 for inverse video characters and will equal zero for normal video characters in mode 0. There are two bits in modes 1 and 2 that determine the color, but the conversion routine in the above IF statements will interpret them both correctly.

The other code conversion would be for characters read from the keyboard. Several people have asked me how to eliminate the keyboard click. The only way to completely eliminate it would be to disconnect the keyboard speaker, but you can use another method if you write your programs to accommodate it. Instead of using INPUT and GET to obtain information from the keyboard, you can PEEK location 764. This location contains the keyboard code of the last key pressed on the keyboard. You must read this location, then POKE 764,255. If the location contains 255 you know that no key has been pressed since the last time you read it.

The problem with this method is that the code you read is neither ATASCII nor the internal code. You can get the values of all of these codes by running the following program:

```
10 REM ** KEYBOARD CODES **
11 REM ** STOP BY PRESSING BREAK **
12 REM **
13 REM **
20 PRINT "PRESS KEY AND THIS PROGRAM
30 PRINT "WILL DISPLAY THE
40 PRINT "CORRESPONDING KEYBOARD
   CODE AS A DECIMAL VALUE:"
50 N = PEEK(764)
60 IF N = 255 THEN 50
70 POKE 764,255
80 ? N;" ";
90 GOTO 50
```

If you use this program as a subroutine by itself, it will act as a GET statement. Putting the subroutine in a loop that stacks the codes in a string until it gets a RETURN code will act as an INPUT statement for alphanumeric input. For this, remember to display the characters on the screen and to make allowances for backspaces. Now your program will not produce a click with each keystroke.

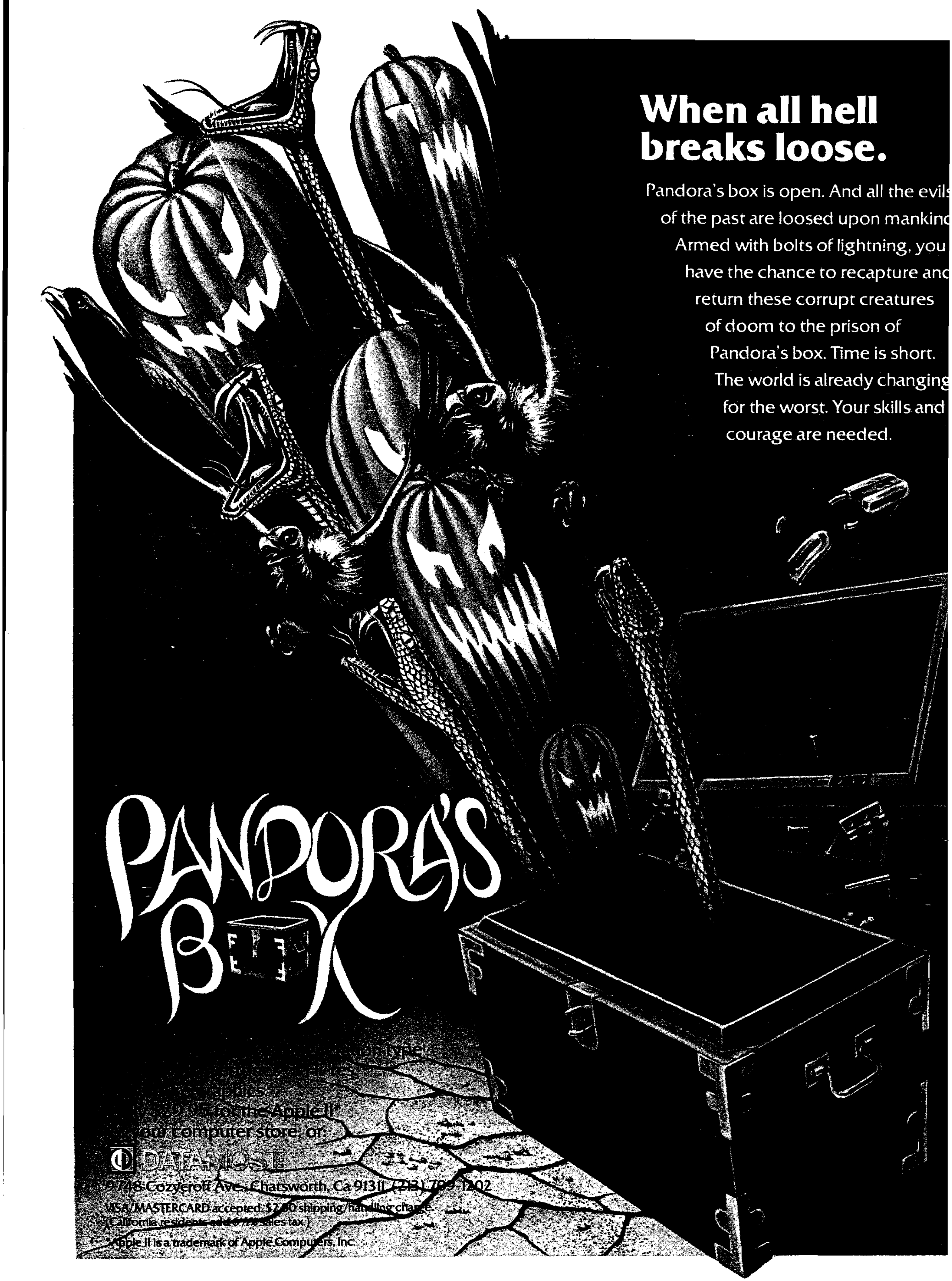
The only other common code conversions required are for the graphics screens. Those are simpler than the other conversions. If you are using the standard screen set up by BASIC, it is much easier to use standard BASIC statements like PLOT and DRAWTO. If you want to set up a specific shape that would require a lot of DRAWTO commands for a relatively small area, you may want to use PRINT.

Although converting to exact byte values to POKE onto the screen is possible, PRINT allows you to address each individual pixel on the screen. You PRINT an alphanumeric string to the screen through channel six. In mode 3, POSITION the graphics cursor at the beginning of one of the lines in the image, then PRINT #6;"112233" for two pixels each of colors 1, 2, and 3. To print the background color, which will allow you to erase an image, use zero, four, or a space. In two-color modes, use only zero and one. This method will save you substantial conversion over PEEKing and POKEing and will, in some cases, run much faster than the equivalent PLOT and DRAWTO statements. You don't need a COLOR statement for the PRINT method because you specify the color register directly, and there is an additional advantage to providing a version of the image right in the program (invaluable in debugging).

Next Month

My January column will introduce the Operating System and Hardware manuals and a few other sources of more technical information on the Atari. I plan to make the Technical Tidbits a regular feature, so send in your questions.

MICRO


When all hell breaks loose.
Pandora's box is open. And all the evils of the past are loosed upon mankind. Armed with bolts of lightning, you have the chance to recapture and return these corrupt creatures of doom to the prison of Pandora's box. Time is short. The world is already changing for the worst. Your skills and courage are needed.
PANDORA'S BOX
your computer store, or:
DATAMOST
9748 Cozycroft Ave., Chatsworth, Ca 91311
VISA/MASTERCARD accepted. shipping/handling charge.
Apple II is a trademark of Apple Computers, Inc.

MICRO News

by Phil Daley, MICRO Staff Editor

Apple Bits and Pieces

As the release date for a new APPLE approaches, rumors fly fast and furious. Apple is securing sources for one million 68000 microprocessors, leading me to believe that the "Lisa" model (APPLE IV?) will be the first out, probably this Spring. It is to sell for approximately $8000 and to be pitched at the business person who knows little about computers. At least, those are the rumors.

•••

The "Seem alike" Franklin ACE 1000 may prompt Apple to release the Super Apple II sooner than originally anticipated. In addition to having 64K standard, rumor has it that the Super Apple II will contain far fewer chips on the mother board and will sell for substantially less.

The Franklin looks like an Apple II, especially when you take the cover off (the only noticeable difference is the larger power supply). The mother board looks almost identical, although somewhat enlarged. The chips are all the same and the I/O slots are similar. The Franklin is delivered with Applesoft and the Apple monitor ROMs installed. The other principal differences are that the Franklin accepts and displays lower case and has no color capabilities, soon to be remedied according to the manufacturer.

Having lost the preliminary injunction ruling against Franklin, Apple is asking for a reconsideration due to a similar case that ruled in favor of the manufacturer. Apple's position is that object code is copyrightable, and therefore proprietary and not usable by others.

Just to make the issue more complicated, Franklin is suing Apple for price manipulation and threatening Apple dealers who want to carry Franklin products.

Also pushing on the retail price are the Far East imitations, yet to be seen in the U.S., which are selling at one-fifth the normal European selling price.

•••

There are rumors that the Mackintosh (also from Apple), a cheaper, simpler version of Lisa, is still in the developmental stage and is not expected until the end of next year at the earliest.

•••

MICRO Bulletin Board

MICRO has instituted a sophisticated Bulletin Board/ Information Service System on our Apple II, which will be available to subscribers Monday through Thursday nights from 5:00 PM to 8:00 AM Eastern Time. The MICRO Bulletin Board System is using software developed by Computer Stations, Inc., of Granite City, IL, and a D.C. Hayes Associates, Inc., microcoupler. Our telephone number is (617) 256-1446.

After dialing into the Net-Works program, a self- explanatory menu is presented. The option (C)hat will not be supported. The first time that you log on you will be expected to leave your name, etc., for subscription verification.

This check will take at least one day. You will have only limited access to the system until your name has been verified and added to the queue of valid users. Please write down the password that the system assigns to you so that you can use it for future calls. A "< ctrl >S" will temporarily stop the system in case it is scrolling too fast to read. Generally, new users may read the system, but not write to the system until verified. We are planning a communications issue for April with articles on all aspects of computer communications. If you have written an article or have any suggestions or criticisms, please send them to us here at MICRO.

•••

A Computer Center

A new resource center has been opened in Newton, MA, to meet the educational and instructional needs of executives who are interested in learning how to make effective use of desktop computers. Called The Computer Forum, this educational institution will offer integrated courses, software selection, continuing help, and customized seminars to interested individuals and businesses. Course offerings will include *How to Make Computers Work for You, Using Your Apple, Programming in BASIC, Data Bases, Using Business Graphics, The Electronic Spreadsheet, Advanced VisiCalc Techniques,* and *Management and Analysis Using VisiCalc.* The Forum has several classrooms, one for each system. Currently, only the Apple room is fully equipped, but plans call for an IBM PC room and possibly a XEROX room. Sign-up for the first schedule of courses has been brisk. We wish the Forum much success and hope that additional centers can be opened around the country.

MICRO

Statement of ownership, management, etc., required by the act of Congress of October 23, 1962, of MICRO, published monthly at Chelmsford, Massachusetts, for November 1982.

The name and address of the publisher is MICRO INK, 34 Chelmsford Street, Chelmsford, Massachusetts. The President/ Editor-in-Chief is Robert M. Tripp of Chelmsford, Massachusetts.

The owner is THE COMPUTERIST, Chelmsford, Massachusetts and the names and addresses of stockholders owning or holding one percent or more of the total amount of stock are: Robert M. Tripp and Donna M. Tripp of Chelmsford, Massachusetts.

The known bondholders, mortgagees and other security holders owning one percent or more of the total amount of bonds, mortgages or other securities are: none.

The average number of copies of each issue of this publication sold or distributed through the mails or otherwise to paid subscribers during the twelve months preceding the date shown above is: 23,292.

I certify that the statements made by me above are correct and complete.

Signed: Robert M. Tripp
President/Editor-in-Chief

It's All Relative— CBM Disk Techniques, Part I

by Jim Strasma

Contributing editor Jim Strasma begins a series that explains how to get the most from CBM's powerful disk operating system. Examples are drawn from a well-written mailing list package that is both inexpensive and widely available. In Part 1 Jim covers global variables, combining BASIC with machine language, and chaining of program modules.

Editor's Note: To implement all of these techniques you should have a DOS 2.0 (or later) disk drive. BASIC 4.0 is also assumed. However, ways to emulate BASIC 4.0 disk commands from Upgrade BASIC and VIC BASIC are summarized.

One of the best features of Commodore's BASIC 4.0 and DOS 2 is its use of relative records for data files. This is a very powerful technique, not well matched by competing computers in Commodore's price range. However, relative records can be quite confusing, and though they have been around for two years now, are largely used in commercial programs. However, there is one large program package freely available that uses relative records — Chris Bennett's "Mail List 4040." In one form or another it has been around for about two years. For much of that time I have been modifying and documenting it.

With the help of the mail list, this series of six articles will thoroughly explain the use of relative records. It will also cover some programming techniques for large packages and a machine-language program that takes much of the drudgery out of data entry programming.

In this first article I will prepare the computer to run the mail list. In the process, I will: 1. show how to mix BASIC and machine language, 2. have one program load another without stopping or losing variables (called *chaining*), and 3. explain the use of global variables (called *soft coding*).

Because of the general availability of Bennett's "Mail List," a full listing will not be presented here. However, you don't need the program to understand the articles. If you do wish to obtain the program, see the box on page 41.

Mixing BASIC and Machine Language

One of the more difficult tasks in programming is mixing BASIC and machine-language code gracefully. When first released, the mail list used one common method, reading the machine-language portion from data statements and POKEing it into working locations. This method easily allows changes to the BASIC program. However, if the machine-language portion is sizeable it can be slow; incorporating substantial changes from a new assembly of the machine-language portion would be tedious at best.

Next, I tried attaching the machine-language portion to the end of the BASIC code and using a machine-language SYS call to boot it into working location. This method is fast. However, it makes modifications to the BASIC program difficult, as any change in the length of the program also moves the machine code, guaranteeing a crash when the new version is used.

Now I use a small trick to load the machine-language portion separately from the BASIC part. This method is quick and allows easy changes to both the BASIC and machine-language portions of the program.

Line 1040 checks to see whether a key location contains the value it does when the machine code has been loaded. If not, MEMSIZ, the zero-page location that controls top-of-memory pointers, is lowered along with FRETOP, the top-of-dynamic strings pointer. (On the VIC, MEMSIZ is at $37 and FRETOP is at $33.)

The two POKEs protect the machine code from BASIC's dynamic string variables. Note that if only MEMSIZ were altered, BASIC would think it had a negative amount of memory free. Since changing these pointers ruins any variables already in the top of memory, it is essential to do it only at the beginning of the first program module.

```
1030 REM LOAD OBJECT PORTION
     IF HAVEN'T
1040 IF PEEK(31232) < > 76 THEN
     POKE 53,122:POKE 49,122
     :DLOAD "OBJECT CODE"
```

After resetting the memory pointers, line 1040 loads the machine-language portion from disk as a program named "object code." Usually loading a new program destroys the old one, but not this time. "Object code" loads very high in memory, beginning at location 31232, ($7A00). It will overwrite anything else up there, such as Universal DOS support, but not BASIC programs located lower in memory.

Since the DLOAD command was part of a running program, BASIC attempts to execute "object code" as soon as it is fully loaded. However, BASIC assumes its programs begin where another pointer, TXTTAB points. In this case, we've left it alone. This means that BASIC will execute "mail list 4040" again. That is the main reason for checking to see whether "object code" has already been loaded. Otherwise we would never get past line 1040.

After the load the IF test in line 1040 fails and the program continues.

Chaining

Line 1060 is another line that must appear at the beginning of the first program module. For program chaining to work correctly, we must either make the first program the largest one, or else convince BASIC that this is so. We could do this by adding dozens of long lines to the program as ballast. However, this would add to its loading time, and take up more storage space on the disk. I have only followed that idea to the extent of coding this module very loosely, with mostly single-statement lines and lots of REMark statements. The added clarity is worth the slight waste. I also started with line number 1000 to keep all line numbers the same length, again for clarity.

In early versions of the mail list, chaining worked by altering the file size pointer, VARTAB at location 42 ($2A), as each module began. This worked because BASIC keeps track of the actual file size in pointer EAL, at location 201 ($C9), during a load. (On VIC, VARTAB is at $2D and EAL is at $AE.) We simply had a line like the one below at the start of each module.

```
10 POKE 42,PEEK(201):POKE 43,
   PEEK(202):CLR
```

Unfortunately, it won't work without the CLR, and once CLR is used, the old variables are gone. This means that a separate disk file has to be established and loaded by each module to remember global variables, or the variables have to be hidden from BASIC and PEEKed. Either method is slow.

By POKEing VARTAB with a value at least as large as it would need to run the largest module, we can use line 1060 instead of line 10, and need it only in the first module.

```
1060 POKE 42,0:POKE 43,53:CLR
```

To determine the correct values to use here, load the longest module in your program, and enter:

```
?PEEK(43)
```

Add two to the result and write it down. Use that number in place of 53 in line 1060. Note that we could have also PEEKed at 42, but I prefer to overstate slightly the required memory. This allows minor additions to that longest module without also requiring a change here.

Don't make program changes to any module after loading it *via* a chain. BASIC no longer knows the module's true size. Instead, reload the module from disk in immediate mode and then make the changes. This is especially important if you have used line 10 above. EAL isn't changed by line editing. If EAL points lower than the end of a modified BASIC program, line 10 would force the variables to begin being stored on top of the last lines, ruining them. To prevent such disasters, it's always a good idea to save a modified program to disk before trying to run it.

The actual chaining happens in line 2060:

```
2060 DLOAD D(PD), "4040 MENU"
     ON U(UN)
```

For BASIC 2.0 and the VIC use:

```
2060 LOAD STR$(PD) + ":4040
     MENU",UN
```

Soft Coding

Notice the variables used in line 2060 above: PD and UN (program drive and disk unit number). They are set earlier in the program, in lines 1220 and 1230:

```
1220 UN = 8:REM DISK UNIT
1230 PD = 0:REM PROGRAM DRIVE
```

By setting them there and using only the variable names everywhere else in the program package, it is easy to change the package to work with different equipment, such as a disk drive that answers to device 9 instead of 8. We will have more to say about soft coding shortly, but first we need to finish setting up.

Setting Text Mode

One other task awaits us in preparing the machine. Commodore computers have two character sets, one for graphics and one for upper- and lowercase text. Since this program uses text, we must enable the text character set. A method that works for all CBM and PET models is given in lines 1080 and 1090 below. (On the VIC, leave out line 1080.)

```
1070 REM SET TEXT MODE
1080 POKE 59468,14
1090 IF PEEK(57345) < > 54 THEN
     PRINT CHR$(14):REM UNLESS
     FAT 40
```

For reasons that make sense only to Commodore, Fat 40's, (the 4016 and 4032 with 12" monitor), are adjusted on the assembly line so that printing CHR$(14) zooms the top and bottom lines off the screen. The IF test in line 1090 prevents this. However, there is also a hardware fix. On the underside of the video display board is a hole labeled "height." Your dealer can adjust your display in about 30 seconds to restore the lost top and bottom lines permanently. If you do it yourself, remember that metal screwdrivers are good conductors and the video board carries 10,000 volts. One slip could do more than violate your warranty.

The CHR$(14) is especially needed by 80-column models. If you leave it out and the machine was previously in graphic mode, lines will appear squished together.

The matching lines to enable the graphic character set are:

```
1070 REM SET GRAPHIC MODE
1080 POKE 59468,12
1090 PRINT CHR$(142)
```

Leaving out the CHR$(142) on 80-column models leaves them with a venetian blind effect, separating lines of graphic characters. No Fat 40 fix is needed this time. (Line 1080 should still be omitted on the VIC.)

Always establish one character set or the other at the start of any program package. CBM models start up in text mode, but PET models start in graphic mode.

Initialization

At this point the machine is ready. The machine-language portion is in and protected. The file pointers have been set for successful chaining and the character set is correct. Now the program begins a long process of initializing variables. Because this takes about five seconds, it is wise to give the user something to look at meanwhile. The mail list starts with a copyright message and then a status line:

```
1200 PRINT "                INITIALIZING
```

This assures the user that the program hasn't died. If the delay will be more than half a minute, also give the user an estimate as to how long the task should take and an occasional progress report.

More on Soft Coding

In the lines following 200 in this first module, the global variables are defined. Because they are not cleared by later modules, the way the entire package works can be modified drastically by changing a single line in this module. Naturally, the other modules have to be carefully written to take advantage of this power. We will see how this is done later in this series of articles.

The global variables used tend to fall into three categories: those that define messages, those that define special characters, and those that act as flags to control the program. The first category allows easy changes to such things as field names or default field contents. These messages may also include cursor control characters to be sure they appear at the correct location on the screen. To ease this task, the mail list predefines a position string of cursor controls in line 1880:

```
1880 PO$ = "[HOME,23DOWN,
     7RIGHT]" + ""
```

The characters shown in square brackets represent literal cursor characters. The codes stand for one home character, followed by 23 cursor downs, followed by seven cursor rights. In the actual mail list, the literal characters are used and the codes are in a REMark statement at the end of the line. Always try to explain lengthy strings made up of cursor controls, especially if anyone will ever need to list your program to a non-Commodore printer.

Later lines select needed portions of the program with LEFT$, as in line 1940:

```
1940 M2$ = LEFT$(PO$,8) + "START
     POSITION :"
```

However, we must be sure the messages are stored in high memory where they will chain correctly. To do this, we concatenate a null string to each literal string in the program, as shown at the end of line 1880.

If we didn't add the null string, BASIC would save space by pointing variable PO$ at its original memory location in line 1880. After chaining, this location would likely contain something quite different, and the string would be ruined. Adding the null string forces it into high memory where it is safe.

The second category of variables is illustrated by line 1830:

```
1830 QT$ = CHR$(34)
```

This is the quote character. It is needed later to allow INPUT# statements to read past troublesome characters like commas. We could use CHR$(34) everywhere instead, but CHR$ is a slow command in BASIC. Predefining QT$ is at least ten times faster overall. Other characters the mail list predefines include RETURN, SHIFTED-RETURN, and SHIFTED SPACE. We will explain how each is used later in this series of articles.

The third class of global variables is the controllers. These include both numeric and string variables, used in IF tests and within expressions later in the program. For instance, line 1210 flags whether or not you want to allow the user to get out of the program by pressing STOP:

(continued)

```
1210 NS=0:REM NON-STOP?
```

If NS=1, the program becomes non-stop; a great idea when using untrained operators, but a terrible idea when a skilled user is trying to modify the package.

An example of a string variable used as a control is PZ$, defined in line 1310:

```
1310 PZ$="A":REM ASCII, P=PET
```

One of the skills of the machine-language portion of the package is that it can convert strings from PET ASCII to true ASCII codes and back again. This is useful when working with a modem or a non-Commodore printer. Line 1760 shows how this feature is used or skipped, depending on the contents of PZ$:

```
1750 REM FLIP CASE OF ASCII
     PRINTER PROMPTS
1760 IF PZ$ < > "A" THEN 1830
1770 SYS SM,1,NA$
1780 C3$=C1$
1790 SYS SM,2,C3$
1800 C4$=C2$
1810 SYS SM,2,C4$
```

My personal copy of the mail list carries the control variable idea a step further by using the variable TY to select between using the package as a church mail list, a computer users' mail list, and a sermon file, depending on whether TYpe=1, 2, or 3 in a new line added to this module.

The other special options set by the global variables are explained in the instructions that come with the mail list package, so I won't take space for them here. However, if you do get the program, notice that all the simple variables are defined before the arrays are defined. Doing things in this order cuts the initialization delay by 2.5 seconds. Further speed gains are possible by arranging the lines so the most-used variables and arrays are defined before those used less often. The ones most heavily used are usually inside nested loops and often-used subroutines.

Using Program Intelligence

The program selects either an ASCII or a PET printer, as we saw in line 1310. However, it doesn't simply assume the printer is on, but goes to the trouble of checking, in lines 1350-1380:

```
1300 DV=4:REM PRINTER
.
.
.
1340 REM BE SURE PRINTER IS ON
1350 OPEN 4,DV
1360 PRINT#4,CHR$(7);:REM BELL
1370 IF ST THEN PZ$="N":
     PRINT "          PRINTER IS OFF
1380 CLOSE 4
```

Line 1360 tries to print a BELL character to the selected printer device. If it succeeds, the IF test of the status variable will fail in line 1370. Otherwise, a warning is printed and the printer control variable is set to show no printer is on line. This allows users without a printer to safely use the package.

A similar technique is used in lines 1250-1290:

(continued)

```
1240 REM SELECTS DATA DRIVE
1250 DD=1
1260 OPEN 15,UN,15
1270 PRINT#15,"INITIALIZE"+
     STR$(DD)
1280 IF DS=74 THEN DD=0:REM IF
     SINGLE DRIVE
1290 CLOSE 15
```

As these lines initialize disk drive one, they identify single drive units and prepare the program to work with either single or dual drives.

An earlier version of the program had the user select one or two drives manually by changing line 250. However, I use both single and dual drives often, and decided it made more sense to let the computer use its own intelligence to work with all Commodore disk drives. This kind of intelligence in a program means more work for the programmer once, but less work for all the users for years to come. Programs you expect to give or sell to others should work on all existing and likely models. (If I followed that advice fully, this program would have used BASIC 2.0 disk commands, at some cost in speed and a great cost in clarity.)

Next time we will begin working with relative records — creating the files needed by the mail list package.

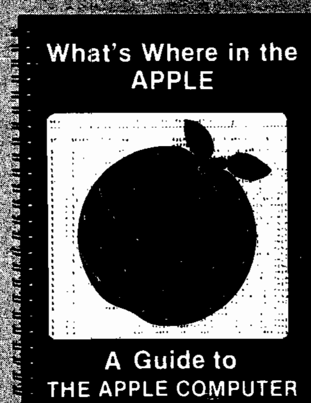
MICRO

How to Obtain Bennett's "Mail List"

Many users' groups will have this program in their libraries. It is also available from ATUG (200 S. Century, Rantoul, IL 61866), TPUG (381 Laurence Ave W., Toronto, Ontario M5M 1B9, Canada), or from the author as part of his HELP disk. The HELP disk is a companion to the third edition of Osborne/McGraw-Hill's *CBM and PET Computer Guide* (edited by the author).

To obtain the HELP disk send $15 to the address below. Specify 4040/2031 or 8050 format.

HELP Disk
Jim Strasma
1280 Richland Ave.
Lincoln, IL 62656



Squeeze for PET BASIC Program

by Hans Hoogstraat

This short routine removes the unnecessary spaces, REMs, and blank lines from a BASIC program. It is relocatable and does not require maintaining two versions of the BASIC program.

SQUEEZE
requires:
PET/CBM — original, upgrade, or 4.0 ROMs

This routine squeezes all the imbedded blanks, line separators, and comments from a BASIC program. In addition, the following syntax corrections are made:

1. GO TO .... = GOTO
2. IF .... GOTO = IF .. THEN
3. IF .. THEN GOTO = IF .. THEN

SQUEEZE is relocatable and can be stored in either cassette buffer. It is designed to be called with a SYS command in the first line of your BASIC program. This means that you need to store only one copy — fully commented and expanded — of your program on tape or disk. When you run the program, it is automatically compressed first.

BASIC Example Program:
(XXX = ADDRESS OF SQUEEZE ROUTINE)

```
10 SYSXXX
15 :
20 REM EXAMPLE PROGRAM
25 :
30 PRINT "EXAMPLE PROGRAM"
35 :
40 FOR I = 1 TO 10
45 :::PRINT I, SQR(I)::REM ROOTS
50 NEXT
55 :
60 IF I < > 0 THEN TO TO 80 ::
65 :
70 I = 1:::B = 1:: REM NONSENSE
75 :
80 END
```

After the SYSXXX squeeze call, the program continues execution with the following BASIC code:

```
10 SYSXXX
30 PRINT"EXAMPLE PROGRAM"
40 FORI=1TO10
45 PRINTI,SQR(I)
50 NEXT
60 IFI < > 0THEN80
70 I=1:B=1
80 END
```

Cautions:

1. Do not use SYS XXX; any blanks between SYS and XXX can confuse the BASIC run-time pointers.
2. Any GOTO, GOSUB, or THEN references to REM-commented lines or : null lines will become erroneous due to the deletion of these lines. (Ed. note: SQUEEZE does not handle these references.)

SQUEEZE can be loaded into the first or second cassette buffer and can then be permanently saved with the BASIC program using the machine-language monitor SAVE command, or it can be made part of the program with DATA statements containing the machine-language code to be transferred to a suitable spot in memory using POKE commands.

Here is the procedure to save a BASIC program with SQUEEZE in the cassette buffer. (Original ROM: use first cassette buffer — $027A - $0339; upgrade ROM: use either cassette buffer — $027A - $0339 or $033A - $03F9; 4.0 ROM: use second cassette buffer — $033A - $03F9.)

1. Load SQUEEZE routine into correct buffer.
2. Type NEW and load BASIC program.
3. Type SYS4, which will display (4.0 ROM)

```
    PC   IRQ  SR AC XR YR SP
.;  0005 E455 30 00 5E 04 F0
```

4. Type .M 002A 002B to display the start-of-BASIC variables pointer, which is usually the same as the end-of-BASIC text pointer. Assume the following display from the above command:

```
.M  002A 002B
.;  002A 4B 04 4B 04 4B 04 00 80
```

5. Now, to save the BASIC program and the SQUEEZE routine together on disk assuming SQUEEZE was loaded in the first cassette buffer, type

```
.S "0:EXAMPLE",08,027A,044B
```

027A = Start address of first cassette buffer.
044B = Contents of end-of-BASIC text pointer as displayed in locations $002B-$002A.

For tape use 01 instead of 08.

General Information

All CBM system labels references are consistent with the labels specified in Appendix F of the *PET/CBM Personal Computer Guide* by A. Osborne.

Hexadecimal dumps of the routine assembled for the three different versions of the PET ROMs are included in this article.

With some minor pointer modifications, the SQUEEZE routine should also operate on most other 6502 systems.

Hans Hoogstraat is a scientific research and systems development software and hardware consultant to the petroleum industry. You may contact him at Box 20, Site 7, SS 1, Calgary, Alberta, Canada T2M 4N3.

Listing 1: **SQUEEZE Assembled for 4.0 ROMs**

```
                ;SYSTEM EQUATES
                ;
                ;BASIC .DI 1 ;ORIGINAL ROM
                ;BASIC .DI 3 ;UPGRADE ROM
                ;BASIC .DI 4 ;BASIC 4.0
                ;
                BASIC       .DI 4
                ;
                ;---------------------------------------------------
                ;------------------- SQUEEZE -----------------------
                ;---------------------------------------------------
                ;
                ;THIS ROUTINE SQUEEZES A BASIC PROGRAM FROM ALL ITS
                ;IMBEDDED BLANKS, LINE SEPARATORS AND COMMENTS.
                ;
                ;IN ADDITION THE FOLLOWING SYNTAX CORRECTIONS ARE MADE:
                ;
                ;1. GO TO ......... = GOTO
                ;2. IF ....... GOTO = IF .. THEN
                ;3. IF .. THEN GOTO = IF .. THEN
                ;
                ;BASIC REFERENCES.
                ;
                            IFE BASIC-1
                BPOINT      .DI $7A
                WORK        .DI $A6
                LNKPRG      .DE $C430
                            ***
                ;
                            IFE BASIC-3
                BPOINT      .DI $28
                WORK        .DI $54
                LNKPRG      .DE $C442
                            ***
                ;
                            IFE BASIC-4
                BPOINT      .DI $28
                WORK        .DI $54
                LNKPRG      .DE $B4B6
                            ***
                ;
                            .BA BPOINT
                ;
0028-           TXTTAB      .DS 2           ;POINTER TO START OF BASIC
002A-           VARTAB      .DS 2           ;POINTER TO START OF VAR.
002C-           ARYTAB      .DS 2           ;PNTR TO START OF ARRAY TA
002E-           STREND      .DS 2           ;POINTER TO END OF VAR.
                ;
                ;PAGE ZERO WORK AREAS.
                ;
                            .BA WORK
                ;
0054-           INPPTR      .DS 2           ;INPUT LINE POINTER.
0056-           NXTLIN      .DS 2           ;NEXT BASIC LINE ADDRESS
                OUTPTR      .DI VARTAB      ;OUTPUT LINE POINTER.
0058-           INPIND      .DS 1           ;INPUT TEXT INDEX.
0059-           OUTIND      .DS 1           ;OUTPUT TEXT INDEX.
005A-           OUTSEG      .DS 1           ;OUTPUT LINE SEGMENT LENGT
005B-           QTFLAG      .DS 1           ;QUOT FOUND FLAG.
005C-           PRVOUT      .DS 1           ;PREVIOUS OUTPUT CHARACTER
005D-           IFFLAG      .DS 1           ;IF TOKEN FOUND FLAG.
                ;
                RAMLOC      .DI $400        ;START BASIC TEXT
                ;
                ;BASIC TOKEN EQUATIONS.
                ;
                GOTOTK      .DI $89         ;GO TO
                IFTK        .DI $8B         ;IF
                REMTK       .DI $8F         ;REM
                TOTK        .DI $A4         ;TO
                THENTK      .DI $A7         ;THEN
                GOTK        .DI $CB         ;GO
                ;
                ;---------------------------------------------------------
                ;
                            .BA $33A
                ;
                ;SET BASIC OUTPUT LINE ADDRESS POINTER.
                ;
033A- A9 01     SQUEEZE     LDA #L,RAMLOC+1
033C- 85 2A                 STA *OUTPTR
033E- A0 04                 LDY #H,RAMLOC+1
0340- 84 2B                 STY *OUTPTR+1
                ;
                ;SET BASIC INPUT LINE ADDRESS POINTER.
                ;
0342- 85 54     NEXTLIN     STA *INPPTR
0344- 84 55                 STY *INPPTR+1
                ;
                ;RESET ALL BASIC SCAN LINE FLAGS.
                ;
0346- A0 00                 LDY #0
0348- A2 00                 LDX #0
                ;
                ;COPY BASIC LINK AND LINE NUMBER FROM INPUT TO OUTPUT.
                ;
034A- B1 54     COPYLNK     LDA (INPPTR),Y
034C- 91 2A                 STA (OUTPTR),Y
034E- 99 56 00              STA NXTLIN,Y
```

(Continued on next page)

Listing 1 *(continued)*

```
0351- 96 5A               STX *OUTSEG,Y
0353- C8                  INY -
0354- C0 04               CPY #4
0356- 90 F2               BCC COPYLNK
                    ;
                    ;--- CARRY SET ---.
                    ;
                    ;SET START BASIC INPUT AND OUTPUT TEXT INDEXES.
                    ;
0358- 84 58               STY *INPIND
035A- 84 59               STY *OUTIND
                    ;
                    ;CHECK FOR END OF BASIC TEXT.
                    ;
035C- A0 01               LDY #1
035E- B1 2A               LDA (OUTPTR),Y
0360- D0 16               BNE SCAN
                    ;
                    ;ADJUST START OF VARIABLE ADDRESS.
                    ;
0362- A2 05               LDX #5
                    ;
0364- A4 2B               LDY *VARTAB+1
0366- A5 2A               LDA *VARTAB
0368- 69 01               ADC #1                 ;WITH CARRY SET = ADC #2.
036A- 90 01               BCC CLR
036C- C8                  INY -
                    ;
                    ;PERFORM BASIC CLR
                    ;
036D- 94 2A         CLR   STY *OUTPTR,X
036F- CA                  DEX -
0370- 95 2A               STA *OUTPTR,X
0372- CA                  DEX -
0373- 10 F8               BPL CLR
                    ;
                    ;FIX BASIC LINKS AND RETURN TO CALLER.
                    ;
0375- 4C B6 B4      LINK  JMP LNKPRG
                    ;
                    ;------------------------------------------------------------
                    ;
                    ;SCAN BASIC INPUT TEXT LINE.
                    ;
0378- A4 58         SCAN  LDY *INPIND            ;GET AN INPUT TEXT CHARA(
037A- B1 54               LDA (INPPTR),Y
037C- E6 58               INC *INPIND            ;BOOST INPUT TEXT INDEX.
                    ;
037E- A6 5B               LDX *QTFLAG            ;BASIC QUOT FOUND FLAG O
0380- D0 45               BNE OUTTEXT            ;YES .. COPY ALL TEXT CH
                    ;
0382- C9 20               CMP #'                 ;TEXT = BLANK ?
0384- F0 F2               BEQ SCAN               ;YES .. IGNORE BLANKS.
                    ;
0386- C9 8F               CMP #REMTK             ;TEXT = REM ?
0388- D0 01               BNE CKSEG              ;NO ... NEXT CHECK.
                    ;
038A- 8A                  TXA -                  ;YES .. FORCE END-OF-LIN
                    ;
038B- C9 3A         CKSEG CMP #':                ;END OF TEXT LINE SEGMEN
038D- D0 0B               BNE CKEOL              ;NO ... NEXT CHECK.
                    ;
                    ;--- CARRY SET ---.
                    ;
038F- 86 5D               STX *IFFLAG            ;YES .. RESET IF FLAG.
                    ;
0391- A4 5A               LDY *OUTSEG            ;ANY SEGM. CHARS. ON OUT
0393- F0 E3               BEQ SCAN               ;NO ... IGNORE SEGM. SEP
                    ;
0395- CA                  DEX -                  ;YES .. TRIGGER ZERO SEG
0396- 86 5A               STX *OUTSEG
                    ;
                    ;--- CARRY STILL SET ---.
                    ;
0398- 90 A8         NEXTLINJ BCC NEXTLIN         ;LONG JUMP ACCOMODATION.
                    ;
039A- AA            CKEOL TAX -                  ;TEXT = END-OF-LINE ?
039B- F0 2A               BEQ OUTTEXT            ;YES .. COPY EOL-TEXT CH
                    ;
039D- E6 5A               INC *OUTSEG            ;INCR. OUTPUT SEGMENT CH
                    ;
039F- A4 5C               LDY *PRVOUT            ;GET PREVIOUS OUTPUT CHA
                    ;
03A1- C9 8B         CKIF  CMP #IFTK              ;TEXT = IF TOKEN ?
03A3- D0 02               BNE CKGO               ;NO ... NEXT CHECK.
                    ;
03A5- 85 5D               STA *IFFLAG            ;FLAG HAPPENING.
                    ;
03A7- C9 CB         CKGO  CMP #GOTK              ;TEXT = GO TOKEN ?
03A9- D0 02               BNE CKTO               ;NO ... NEXT CHECK.
                    ;
03AB- A9 89               LDA #GOTOTK            ;YES .. REPLACE BY GOTO
                    ;
03AD- C9 A4         CKTO  CMP #TOTK              ;TEXT = TO TOKEN ?
03AF- D0 08               BNE CKIFGO             ;NO ... NEXT CHECK.
                    ;
03B1- C0 89               CPY #GOTOTK            ;PRECEEDED BY GOTO TKEN ?
03B3- F0 C3               BEQ SCAN               ;YES .. IGNORE INPUT TO
                    ;
```

Listing 1 *(continued)*

```
03B5- C0 A7                CPY #THENTK        ;PRECEEDED BY THEN TOKEN ?
03B7- F0 BF                BEQ SCAN           ;YES .. IGNORE INPUT TO TO
                ;
03B9- A6 5D     CKIFGO     LDX *IFFLAG        ;IF TOKEN FOUND ?
03BB- F0 0A                BEQ OUTTEXT        ;NO ... COPY TEXT CHARACER
                ;
03BD- C9 89     CKGOTO     CMP #GOTOTK        ;TEXT = GOTO TOKEN ?
03BF- D0 06                BNE OUTTEXT        ;NO ... COPY TEXT CHARACTE
                ;
03C1- C0 A7                CPY #THENTK        ;PRECEEDED BY THEN TOKEN ?
03C3- F0 B3                BEQ SCAN           ;YES .. IGNORE INPUT GOTO
                ;
03C5- A9 A7                LDA #THENTK        ;YES .. REPL. GOTO BY THEN
                ;
03C7- A4 59     OUTTEXT    LDY *OUTIND        ;COPY TEXT CHARACTER TO OU
03C9- 91 2A                STA (OUTPTR),Y
03CB- 85 5C                STA *PRVOUT        ;SAVE AS PREVIOUS OUTPUT C
03CD- E6 59                INC *OUTIND        ;BOOST OUTPUT TEXT INDEX.
                ;
03CF- C9 22                CMP #'"            ;A BASIC QUOT COPIED ?
03D1- D0 04                BNE CKEND          ;NO ... CONTINUE
                ;
03D3- 45 5B                EOR *QTFLAG        ;SET BASIC QUOT FOUND FLAG
03D5- 85 5B                STA *QTFLAG        ;TO EITHER ON OR OFF.
                ;
03D7- A5 5C     CKEND      LDA *PRVOUT        ;END-OF-LINE REACHED ?
03D9- D0 9D                BNE SCAN           ;NO ... CONTINUE SCAN.
                ;
                ;OUTPUT TEXT LINE CLEANUP
                ;
03DB- C0 05     CLEANUP    CPY #5             ;ANY OUTPUT LINE CHARACTER
03DD- 90 11                BCC NEXTIN         ;NO ... DELETE LINE.
                ;
                ;--- CARRY SET ---.
                ;
03DF- A6 5A                LDX *OUTSEG        ;ANY OUTPUT LINE SEGMENT C
03E1- D0 04                BNE NEXTOUT        ;YES .. VALID LINE.
                ;
03E3- 88        DELCHR     DEY -              ;DELETE LAST OUTPUT CHARAC
03E4- 8A                   TXA -
03E5- 91 2A                STA (OUTPTR),Y
                ;
03E7- 98        NEXTOUT    TYA -
03E8- 65 2A                ADC *OUTPTR        ;WITH CARRY SET = (A)+1+OU
03EA- 85 2A                STA *OUTPTR
03EC- 90 02                BCC NEXTIN
03EE- E6 2B                INC *OUTPTR+1
                ;
                ;GET THE NEXT BASIC INPUT LINE POINTER.
                ;
03F0- A5 56     NEXTIN     LDA *NXTLIN
03F2- A4 57                LDY *NXTLIN+1
                ;
03F4- 18                   CLC
03F5- 90 A1                BCC NEXTLINJ       ;AND CONTINUE SQUEEZING.
                ;
                           .EN
```

***Listing 2:* Version for BASIC 1.0 Original ROM**

```
000 A9 01 85 7C A0 04 84 7D
008 85 A6 84 A7 A0 00 A2 00
010 B1 A6 91 7C 99 A8 00 96
018 AC C8 C0 04 90 F2 84 AA
020 84 AB A0 01 B1 7C D0 16
028 A2 05 A4 7D A5 7C 69 01
030 90 01 C8 94 7C CA 95 7C
038 CA 10 F8 4C 30 C4 A4 AA
040 B1 A6 E6 AA A6 AD D0 45
048 C9 20 F0 F2 C9 8F D0 01
050 8A C9 3A D0 0B 86 AF A4
058 AC F0 E3 CA 86 AC 90 A8
060 AA F0 2A E6 AC A4 AE C9
068 8B D0 02 85 AF C9 CB D0
070 02 A9 89 C9 A4 D0 08 C0
078 89 F0 C3 C0 A7 F0 BF A6
080 AF F0 0A C9 89 D0 06 C0
088 A7 F0 B3 A9 A7 A4 AB 91
090 7C 85 AE E6 AB C9 22 D0
098 04 45 AD 85 AD A5 AE D0
0A0 9D C0 05 90 11 A6 AC D0
0A8 04 88 8A 91 7C 98 65 7C
0B0 85 7C 90 02 E6 7D A5 A8
0B8 A4 A9 18 90 A1
```

***Listing 3:* Version for BASIC 3.0 Upgrade ROM**

```
000 A9 01 85 2A A0 04 84 2B
008 85 54 84 55 A0 00 A2 00
010 B1 54 91 2A 99 56 00 96
018 5A C8 C0 04 90 F2 84 58
020 84 59 A0 01 B1 2A D0 16
028 A2 05 A4 2B A5 2A 69 01
030 90 01 C8 94 2A CA 95 2A
038 CA 10 F8 4C 42 C4 A4 58
040 B1 54 E6 58 A6 5B D0 45
048 C9 20 F0 F2 C9 8F D0 01
050 8A C9 3A D0 0B 86 5D A4
058 5A F0 E3 CA 86 5A 90 A8
060 AA F0 2A E6 5A A4 5C C9
068 8B D0 02 85 5D C9 CB D0
070 02 A9 89 C9 A4 D0 08 C0
078 89 F0 C3 C0 A7 F0 BF A6
080 5D F0 0A C9 89 D0 06 C0
088 A7 F0 B3 A9 A7 A4 59 91
090 2A 85 5C E6 59 C9 22 D0
098 04 45 5B 85 5B A5 5C D0
0A0 9D C0 05 90 11 A6 5A D0
0A8 04 88 8A 91 2A 98 65 2A
0B0 85 2A 90 02 E6 2B A5 56
0B8 A4 57 18 90 A1
```

BASIC Line Delete for PET/CBM and VIC

by Thomas Henry

Use this convenient utility during your BASIC program development. It allows you to delete a whole range of lines, rather than just one at a time.

BASIC Line Delete
requires:

Upgrade or 4.0 PET/CBM
or VIC

"BASIC Line Delete," a command you can add to your Commodore computer's resident BASIC, deletes blocks of BASIC lines instantly. For example, suppose you wish to delete line numbers 1000 through 5000 in a BASIC program. Simply type "<1000-5000" and hit [return] and all those lines will be deleted instantly! This BASIC Line Delete function is easy to use since the syntax is the same as that found for the LIST command. In addition, extensive error checking is employed to avoid disasters.

You can consider BASIC Line Delete as an addition to the computer's BASIC language. It is loaded into the computer at the start of a session and can be invoked at any time, in the immediate mode, to perform its task. Because this 177 byte-long machine-language program sits at the top of memory with memory pointers lowered accordingly, it can peacefully coexist with any BASIC program.

The original program was written on a CBM-8032 with 4.0 ROMs. However, it should be easy to convert to any type of Commodore computer since the ROM routines used are common to all models — only the addresses are different. In addition, it is likely that other Microsoft BASIC machines can use this program with a few changes. When we examine the ROM routines you will note that they are routines that any BASIC interpreter must have.

VIC-20 owners shouldn't feel left out either. Even though the program is in machine language, the VIC-20 can still use it simply by employing a BASIC loader that POKEs the required data into memory. I will present a program to do this later in the article.

Even if you don't want or need a BASIC Line Delete, you may want to look over the program description anyway. Several interesting routines are presented that could be put to other uses. In addition, you may want to see how the program implements error checking and apply it to your own work.

Format of the New Command

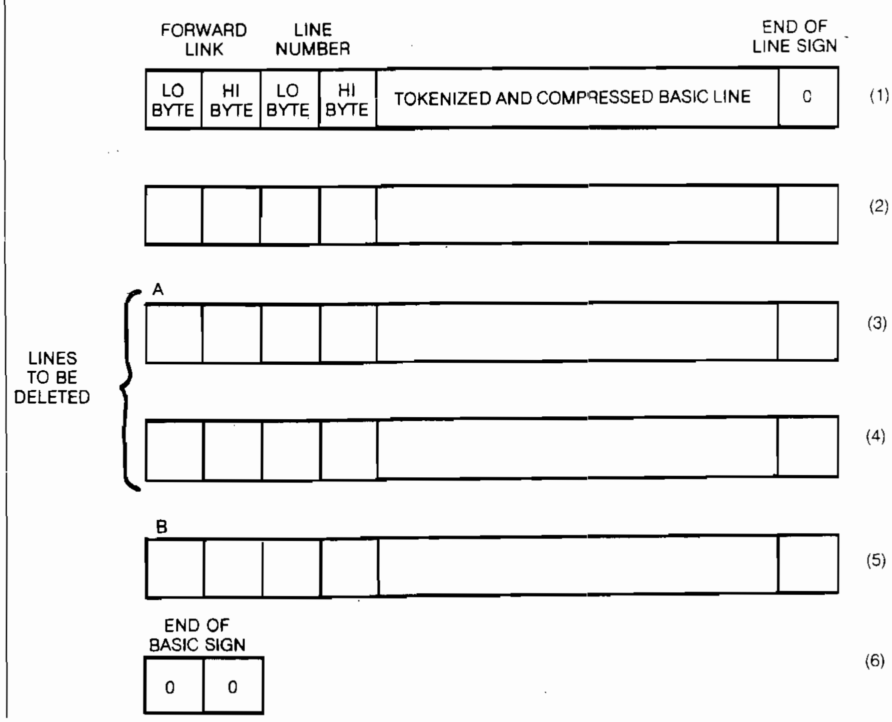
To get a feel for how the program works, let's examine how it should look to the user. The "<" sign indicates the function, although other keys could be used by making one small change in the program. As mentioned before, the format is identical to that used for the LIST command. Let's summarize all proper uses of the BASIC Line Delete:

Proper	Improper
<100-200	<
<100-	<-
<-200	<100
	<--
	etc.

The first statement under proper syntax will delete lines 100 through 200 inclusive. The second one will delete all statements from 100 on. The last one will delete all statements up to line 200 inclusive. And just like the LIST command, there doesn't have to be any line number 100 or 200 for this to work. Suppose the first line number past 90 in your program is actually 122 and the last one before line 210 is 186. Then "<100-200" will still delete all of the lines between this range, meaning that actually lines 122 through 186 are deleted.

The second column shows some of the possible statements with improper

Figure 1 How BASIC is Stored and Principle of DELETE



syntax. If you type any of these, the operation will be aborted and a "?SYNTAX ERROR" message will be returned. It is important to have this feature since a delete function could have potentially catastrophic results if improperly used. So, essentially the statements shown in column one all have proper syntax and will produce meaningful results from the computer, while all other statements will not execute and will produce a syntax error message.

If the range is "backwards" (e.g., <200-100), an error message will again be produced. Finally, I feel so strongly about error checking that I incorporated one more feature. After entering a valid delete command, the computer will respond with "ARE YOU SURE?", giving you one last chance to change your mind! This feature is only available to users with 4.0 operating systems since the "ARE YOU SURE?" routine is part of the normal SCRATCH and HEADER commands.

About the Program

Figure 1 illustrates the principle. As you probably know, a BASIC line is stored in the computer in a specific form. As shown in the illustration, two bytes are devoted to storing the forward link address, which is nothing more than a pointer to the following line in memory. The next two bytes contain the line number. The next area, variable in size, contains the compressed or tokenized BASIC statement. This is polished off with a zero byte to indicate the end of a line. This format is followed throughout memory until the last line is hit. A pair of zeros is included at the end of the last line to indicate the end of the program. (Actually there are three zeros here, if you count the normal end-of-the-line zero). Suppose we wish to delete lines 3 and 4 as indicated in figure 1. What we will do is pick up everything from point B to the end of BASIC and put it back down again at point A. Lines 3 and 4 will be written over in this step. At this point we have just transferred some memory. The link addresses will now be all wrong for the new locations. Fortunately, there is a routine in the ROMs that will rebuild the link addresses for us automatically. After this routine is called the delete has been performed and the BASIC program is all set to go again!

Figure 2 is an assembler listing of the BASIC Line Delete program. As mentioned above, the error checking is the only hard part of the program; the

Figure 2

```
0000                ;***********************************
0000                ;*                                 *
0000                ;*    BASIC LINE DELETE UTILITY    *
0000                ;*                                 *
0000                ;*   ASSEMBLER CODE FOR CBM-8032   *
0000                ;*           THOMAS HENRY          *
0000                ;*                                 *
0000                ;***********************************
0000                ;
0000                ;
0000                VALUE  = $11              ;INTEGER VALUE.
0000                VARBLE = $2A              ;POINTER TO VARIABLES.
0000                MEMTOP = $34              ;TOP OF MEMORY POINTER.
0000                SAVE   = $59              ;SAVE START ADDRESS.
0000                ADDRES = $5C              ;ADDRESS OF FOUND LINE #.
0000                CHRGET = $70              ;BASIC CHRGET ROUTINE.
0000                CHRGOT = $76              ;BASIC CHRGOT ROUTINE.
0000                POINTR = $77              ;CHRGET POINTER.
0000                WEDGE  = $79              ;WEDGE GOES HERE.
0000                RETURN = $7D              ;RETURN TO CHRGET ROUTINE.
0000                FIXUP  = $B4AD            ;ADJUST POINTERS.
0000                CHAIN  = $B4B6            ;REBUILD LINE CHAINING.
0000                SEARCH = $B5A3            ;SEARCH FOR BASIC LINE.
0000                INTEGR = $B8F6            ;FETCH INTEGER INPUT.
0000                ERROR  = $BF00            ;SYNTAX ERROR ROUTINE.
0000                QUERY  = $DB9E            ;'ARE YOU SURE?'
0000                CHROUT = $E202            ;PRINT CHARACTER TO SCREEN.
0000                ;
0000                ;
0000                       * = $7F52
7F52                ;
7F52  A9 4C         SETUP  LDA #$4C           ;OP-CODE FOR 'JMP'.
7F54  85 79                STA WEDGE
7F56  A9 63                LDA #<ENTRY        ;LOW BYTE OF ENTRY.
7F58  85 34                STA MEMTOP         ;LOWER MEMORY TO PROTECT.
7F5A  85 7A                STA WEDGE+1
7F5C  A9 7F                LDA #>ENTRY        ;HIGH BYTE OF ENTRY.
7F5E  85 35                STA MEMTOP+1       ;LOWER MEMORY TO PROTECT.
7F60  85 7B                STA WEDGE+2
7F62  60                   RTS                ;INITIALIZATION COMPLETE.
7F63                ;
7F63                ;
7F63  C9 3C         ENTRY  CMP #'<            ;LOOK FOR DELETE SYMBOL.
7F65  D0 08                BNE COMMON         ;SORRY, NOT HERE.
7F67  48                   PHA                ;YES, IT'S HERE.  SAVE.
7F68  A5 77                LDA POINTR
7F6A  C9 00                CMP #$00           ;CHECK FOR IMMEDIATE MODE.
7F6C  F0 09                BEQ DELETE         ;DO DELETE IF IMMEDIATE.
7F6E  68                   PLA                ;DON'T DO IN PROGRAM MODE.
7F6F  C9 3A         COMMON CMP #$3A           ;COMPLETE CHRGET ROUTINE.
7F71  90 01                BCC FINISH
7F73  60                   RTS
7F74  4C 7D 00      FINISH JMP RETURN
7F77                ;
7F77                ;
7F77  20 70 00      DELETE JSR CHRGET         ;FETCH FIRST CHARACTER.
7F7A  90 0D                BCC FIRST          ;IT'S A NUMBER.
7F7C  F0 1E                BEQ MIDDLE         ;NULL INPUT IS ERROR.
7F7E  C9 2D                CMP #'-            ;IS IT A MINUS SIGN?
7F80  D0 1E                BNE BYPASS         ;NO, ERROR!
7F82  20 70 00             JSR CHRGET         ;FETCH NEXT CHARACTER.
7F85  C9 2D                CMP #'-            ;IS IT ANOTHER MINUS SIGN?
7F87  F0 73                BEQ BAD            ;IF IT IS, THEN ERROR.
7F89  20 F6 B8      FIRST  JSR INTEGR         ;ACCEPT INTEGER INPUT.
7F8C  20 A3 B5             JSR SEARCH         ;FIND THE LINE NUMBER,
7F8F  A6 5C                LDX ADDRES         ;AND SAVE ITS ADDRESS.
7F91  A4 5D                LDY ADDRES+1
7F93  86 59                STX SAVE
7F95  84 5A                STY SAVE+1
7F97  20 76 00             JSR CHRGOT         ;LOOK AGAIN AT CHAR.
7F9A  90 13                BCC LAST           ;GO GET LAST LINE NUMBER.
7F9C  F0 5E         MIDDLE BEQ BAD
7F9E  C9 2D                CMP #'-            ;IS IT A MINUS SIGN?
7FA0  D0 5A         BYPASS BNE BAD            ;NO, ERROR!
7FA2  20 70 00             JSR CHRGET         ;YES, FETCH NEXT CHAR.
7FA5  D0 08                BNE LAST           ;IF PRESENT, GO ON.
7FA7  A2 FF                LDX #$FF           ;OTHERWISE DEFAULT TO
7FA9  86 11                STX VALUE          ;LINE NUMBER $FFFF.
7FAB  86 12                STX VALUE+1
7FAD  D0 03                BNE DEFALT         ;BRANCH ALWAYS.
7FAF  20 F6 B8      LAST   JSR INTEGR         ;GET LAST LINE #.
7FB2  20 A3 B5      DEFALT JSR SEARCH         ;FIND ADDRESS OF LINE #.
7FB5  90 0C                BCC CHECK          ;BRANCH, LINE NOT FOUND.
7FB7  A0 00                LDY #$00
7FB9  B1 5C                LDA (ADDRES),Y     ;GET FORWARD LINK TO
7FBB  AA                   TAX                ;POINT TO NEXT LINE IN
7FBC  C8                   INY                ;MEMORY.
7FBD  B1 5C                LDA (ADDRES),Y
7FBF  86 5C                STX ADDRES
7FC1  85 5D                STA ADDRES+1
7FC3  38            CHECK  SEC                ;CHECK TO SEE THAT THE
7FC4  A5 5C                LDA ADDRES         ;START NUMBER IS LOWER
7FC6  E5 59                SBC SAVE           ;THAN THE STOP NUMBER.
```

Figure 2 *(continued)*

```
7FC8  A5 5D           LDA ADDRES+1
7FCA  E5 5A           SBC SAVE+1
7FCC  90 2E           BCC BAD           ;IT'S NOT, SO ERROR.
7FCE  20 9E DB        JSR QUERY         ;IT IS.  LAST CHANCE
7FD1  B0 21           BCS DONE          ;TO CHANGE YOUR MIND.
7FD3  A0 00    MOVE   LDY #$00
7FD5  B1 5C           LDA (ADDRES),Y    ;SHIFT BYTES BACK,
7FD7  91 59           STA (SAVE),Y      ;ONE BY ONE.
7FD9  E6 59           INC SAVE          ;INCREMENT START ADDRESS.
7FDB  D0 02           BNE NOCAR1
7FDD  E6 5A           INC SAVE+1
7FDF  E6 5C    NOCAR1 INC ADDRES        ;INCREMENT END ADDRESS.
7FE1  D0 02           BNE NOCAR2
7FE3  E6 5D           INC ADDRES+1
7FE5  A5 5C    NOCAR2 LDA ADDRES        ;IS END ADDRESS TOUCHING
7FE7  C5 2A           CMP VARBLE        ;THE START OF VARIABLES YET?
7FE9  D0 E8           BNE MOVE          ;IF IT ISN'T, DO MORE.
7FEB  A5 5D           LDA ADDRES+1
7FED  C5 2B           CMP VARBLE+1
7FEF  D0 E2           BNE MOVE
7FF1  20 B6 B4        JSR CHAIN         ;REBUILD CHAINING OF LINES.
7FF4  A9 0D    DONE   LDA #$0D          ;PRINT CARRIAGE RETURN.
7FF6  20 02 E2        JSR CHROUT
7FF9  4C AD B4        JMP FIXUP         ;CLEAN UP POINTERS, ETC.
7FFC  4C 00 BF BAD    JMP ERROR
7FFF                  .END
```

delete part is quite easy. I will let you examine the assembler listing, but as an aid to understanding, let me describe the key ROM routines used in it. You may want to jot these down in your notebook for future reference, since I'm sure these routines have many more valuable uses.

The routine at $B8F6 will get an integer from the screen. The CHRGET routine (at $70) is called first and this causes locations $77 and $78 to point to the start of the integer (which is in ASCII). After a JSR $B8F6, the ASCII representation is converted to a binary form and the result is deposited in locations $11 and $12 (low byte and high byte, respectively). If $77 and $78 point to the "-" sign (as in the command "<-200"), the subroutine will return with zeros in $11 and $12. You can consider this as a default lower line number.

Given a line number, routine $B5A3 will find where in memory that BASIC line sits. Simply put the desired line number in locations $11 and $12 and call routine $B5A3. The routine will return with the address of the first byte of the desired line in locations $5C and $5D. You will note that the routine described in the preceding paragraph ends with the desired data in locations $11 and $12, whereas this routine begins with data in these locations. This means that we can chain the two routines without saving any intermediate results!

An interesting feature of this line-finding routine is its ability to adapt to non-existent line numbers. For example, suppose you tell it to find line 100 but no such number exists in your program. However, your program does contain a statement with line number 110. When you call the routine it will look for number 100 and won't find it. But it will continue to look for the first line number beyond 100 (in this case 110) and return with its address instead. You can see that this is exactly what the BASIC Line Delete program needs! One other feature is that if the exact line number specified was found, then the carry flag is set. Otherwise, as in our example here, the carry flag will be cleared.

In the program, if no last line number is specified, a default number of $FFFF (65535 decimal) is specified. Notice what happens when this number is acted on by subroutine $B5A3. Suppose the actual last number in your BASIC program is 1000 and you enter the command "<250-". The default number $FFFF is loaded into $11 and $12 and routine $B5A3 is called. The routine will start with 65535 and will whittle away at the numbers until it eventually hits your actual last number (1000 in this case). Once again, this is exactly what the BASIC Line Delete requires.

The routine at $DB9E will query "ARE YOU SURE?" and wait for a reply. If the answer is "Y" or "YES" the carry flag will be cleared. Any other response will set the carry flag. Note that due to a quirk in this routine, you should print a carriage return to the screen following it. This will move the cursor to the proper position on the next line. To print a carriage return, do the following:

Figure 3

```
100 REM ****************************************
110 REM *                                      *
120 REM *         BASIC LINE DELETE:           *
130 REM *           VIC-20 VERSION             *
140 REM *                                      *
150 REM *            THOMAS HENRY              *
160 REM *                                      *
170 REM *       TRANSONIC LABORATORIES         *
180 REM *          249 NORTON STREET           *
190 REM *         MANKATO, MN 56001            *
200 REM *                                      *
210 REM ****************************************
220 REM
230 REM
240 PRINT"WAIT A MOMENT..."
250 X=PEEK(55)+256*PEEK(56)-163
260 FORA=X TO X+162
270 READD:POKEA,D:NEXT
280 Y=X+17:H%=Y/256:L=Y-256*H%
290 POKEX+5,L:POKEX+11,H%
300 SYS(X):NEW
310 DATA169,76,133,124,169,110,133,55,133,125,169,29,133,56,133,126
320 DATA96,201,60,208,8,72,165,122,201,0,240,9,104,201,58,144
330 DATA1,96,76,128,0,32,115,0,144,13,240,116,201,45,208,112
340 DATA32,115,0,201,45,240,105,32,107,201,32,19,198,166,95,164
350 DATA96,134,92,132,93,32,121,0,144,19,240,84,201,45,208,80
360 DATA32,115,0,208,8,162,255,134,20,134,21,208,3,32,107,201
370 DATA32,19,198,144,12,160,0,177,95,170,200,177,95,134,95,133
380 DATA96,56,165,95,229,92,165,96,229,93,144,36,160,0,177,95
390 DATA145,92,230,92,208,2,230,93,230,95,208,2,230,96,165,95
400 DATA197,45,208,232,165,96,197,46,208,226,32,51,197,76,42,197
410 DATA76,8,207
```

```
LDA #$0D
JSR $E202
```

To rebuild the forward link chaining, simply call subroutine $B4B6. No set-up is needed to enter this routine.

The BASIC Line Delete program ends with two alternate ways to get back into BASIC. If JMP $B4AD is used, then a graceful return will be made to BASIC, indicating that all went well. However, if a return is made *via* JMP $BF00, the statement "SYNTAX ERROR" will be printed indicating that the attempted operation was aborted.

To round out your survey of this program note that locations $59 and $5A hold the address of the start line number (where the later memory will be moved to; "A" in figure 1). $5C and $5D hold the address of the end line ("B" in figure 1). $2A and $2B are pointers to the end of BASIC.

How to Load and Use the Program

If you have a computer other than 4.0, you will have to make the required translations to your machine. If you have memory maps handy this shouldn't take too long. I was able to make a VIC-20 version in about fifteen minutes simply by comparing memory maps. Just enter the resident machine-language monitor and list out the required lines with the command:

```
.M 7F52,7FFF
```

Now type over what the computer shows, using the byte values generated in the assembly in figure 2 as a guide. When you are done, save the program with the command:

```
.S "DELETE – 32594",08,7F52,7FFF
```

If you are saving to tape replace the "08" with an "01". The number in the title is the SYS number.

Suppose you are using the program at the start of a session (from a cold start). First LOAD the program in the normal way (just like a BASIC program). There is no need to load it from the monitor; the CBM-8032 knows where to put it. Next type NEW and hit return. This step is important since it resets some pointers previously disarrayed by the LOAD command. Now type SYS32594 and hit return. The BASIC Line Delete is now activated. The top of memory pointers are automatically lowered to protect it. You are now free to call up the function whenever desired.

This program is very relocatable. If you decide to put it somewhere else in memory only locations $7F57 and $7F5D need be changed. These two bytes form the address of the CHRGET

Add-on, starting at $7F63 in this case. Everything else remains the same. This is due to extensive use of relative addressing; there are no internal JSR or JMP commands to be altered. Simply transfer the program, change the two bytes mentioned, and run it using the new SYS address!

VIC-20 owners need a different way to get the program into memory since the VIC has no resident machine-language monitor. Figure 3 shows a loader program that will enter an equivalent BASIC Line Delete into memory. Note that this loader is completely automatic since it not only loads the program but also instantly adjusts to VIC-20s with any amount of add-on memory. In addition, the program automatically does a SYS to the right address. All the user has to do is LOAD the program and RUN it!

Now you have a new command for your Commodore computer. You don't really have to understand how it works to use it, but I recommend you look over the assembly listing again. As mentioned before, the ROM routines called are quite powerful and probably have many other uses. In addition, the program itself could serve as an example of how to incorporate worst-case error checking into your own routines.

Acknowledgements

I owe a big debt of gratitude to Dick Immers of the Central Illinois PET User's Group for explaining some of the quirks of the CBM-8032 machine-language tape-save routine. Thanks also go to Dr. Kenneth Good, Mankato State University, for putting early versions of this program to the acid test. He found several conditions that could have caused users real troubles were they not flagged with "SYNTAX ERROR" statements.

Thomas Henry is a professional writer in the areas of electronic music, circuit design, and Commodore computers. He may be contacted at Transonic Laboratories, 249 Norton Street, Mankato, MN 56001.

MICRO

SOUP: A CBM Machine-Language Compare Program

by Henry Troup and Jim Strasma

SOUP is an efficient compare program for machine-language program files on Commodore disk. It uses BASIC 4.0 disk commands, but is otherwise compatible with other Microsoft BASICs.

SOUP
requires:
PET/CBM
disk drives
printer (optional)

This program, originally adapted by Henry Troup from a similar mini-computer utility, compares two versions of a machine-language program on disk and prints out any lines that differ between the two versions. All you need to use SOUP are disk copies of the two machine-language programs to be compared. The only other restriction is that they must begin loading at the same address.

To use the program, place the disk or disks with the files to compare in your disk unit. Also prepare your printer, if you are using one. At start up, you will be asked the name and drive number of the two files. This is the only time in the program that disk status is checked. If an error is found here, repair the cause and re-enter the file name and drive number.

From here on, operation is automatic. As differences are discovered they are listed either to the screen or printer. You may wish to make some changes in the formatting used here. Lines 700 and 710 set the maximum fields per line for screen and printer respectively. If your screen has over 40 columns, or your printer over 80, you may increase the value given to variable mf. Likewise, if your printer is not device #4, change lines 690 and 710 to allow the device number you need. If your paper is not the 11-inch variety common in the U.S., change line 350 to adjust the lines printed per page to your needs.

To better explain its workings, the program as printed here is heavily commented and uses fewer multiple statement lines than it could. Feel free to omit remark statements and lines containing only a colon; none is referenced by other lines. You may also be able to combine some lines. For example, the subroutine beginning in line 460 could be reduced to four lines. Likewise, the spaces that are not within quotation marks may safely be left out. However, you may find it better to leave the program as listed here and compile it.

In the interest of speeding up the program, often-used constants are replaced by variables, seldom-used lines are moved to the end of the listing, and disk status is left unchecked once the needed files are successfully opened. If you notice that the program seems to have halted with the disk error light on, hit the [stop] key, and check the disk status in immediate mode:

?ds$

Most likely the error will be fatal, and you will have to start over again after correcting the problem.

The program uses only a few special characters. In lines 670, 730, 740, 780, and 790 notice the three equal signs in a row (= = =). These represent three [cursor left] characters. These characters place the flashing input cursor over a likely default answer. They also protect the user from accidentally falling out of the program. Even so, you may omit them.

To use this program with other computers or disk drives, you will need only to substitute your disk commands for Commodore's. The most difficult task for other disk operating systems is likely to be reading in the program files one character at a time. The other essential task is to detect the end of file when it is reached. If you know how to do these tasks on your machine, you can probably make SOUP work for you.

Henry Troup and Jim Strasma may be contacted at 1280 Richland Ave., Lincoln, IL 62656.

Listing 1

```
100 REM SOUP -- AS OF 7 SEPT 82
110 GOSUB 630:REM PUT MOST-USED LINES AT START FOR SPEED
120 REM MAIN ROUTINE
130 NM$="SOUP: FILE A="+CF$+" &  FILE B="+PF$:REM TITLE
140 PRINT#4,NM$:REM START NEW PAGE
150 GET#1,A$:REM READ A CHARACTER FROM FILE A
160 S1=ST:REM REMEMBER I/O STATUS OF A
170 IF A$=NL$ THEN A$=ZE$:REM TRAP NULL DATA BUG
180 GET#1,B$:REM READ A CHARACTER FROM FILE B
190 S2=ST:REM REMEMBER I/O STATUS OF B
200 IF B$=NL$ THEN B$=ZE$:REM FIX NULL DATA BUG
210 IF A$=B$ GOTO 420:REM ONLY REPORT DIFFERENCES
220 A=ASC(A$):B=ASC(B$):REM CONVERT TO DECIMAL CODE
230 N=AD:GOSUB 490:REM CONVERT ADDRESS TO HEXADECIMAL
240 PRINT#4,"@"HX$",A=";:REM PRINT MISMATCH
250 N=A:GOSUB 490:REM CONVERT A'S VALUE TO HEX
260 PRINT#4,HX$"+B=";:REM & PRINT IT
270 N=B:GOSUB 490:REM THEN CONVERT B'S
```

Listing 1 *(continued)*

```
280 PRINT#4,HX$;:REM & PRINT IT
290 FC=FC+1:REM PRINT 4 MISMATCHES PER LINE
300 REM TAB IF HAVE ROOM FOR ANOTHER ON LINE
310 IF FC<MF THEN PRINT#4,"  ";:GOTO 420
320 FC=0:REM ELSE RESET FIELD COUNTER
330 PRINT#4:REM & FINISH LINE
340 LC=LC+1:REM INCREMENT LINE COUNTER
350 IF LC<59 THEN 420:REM 58 MISMATCH LINES PER PAGE
360 LC=0:REM RESET LINE COUNTER
370 FOR I=1 TO 6:REM SKIP LAST 6 LINES
380 : PRINT#4
390 NEXT
400 PRINT#4,NM$:REM TITLE NEXT PAGE
410 REM END ON STATUS CHANGE, (END OF FILE)
420 IF S1 OR S2 THEN DCLOSE:PRINT#4:CLOSE 4:END
430 AD=AD+1:REM ELSE INCREMENT ADDRESS COUNTER
440 GOTO 150:REM & CONTINUE
450 :
460 REM DECIMAL TO HEX CONVERTER SUBROUTINE
470 REM ENTER WITH NUMBER IN N
480 REM RETURNS HEX EQUIVALENT IN HX$
490 IF N=0 THEN HX$="00":GOTO 600:REM HANDLE EXCEPTION
500 HX$="":REM INITIALIZE OUTPUT VARIABLE
510 D=-LOG(N)/LOG(16)
520 D%=D-(D<>INT(D))
530 FOR I=D% TO 0:REM LOOP FOR DIGITS
540 : P=16^(-I)
550 : Q%=N/P
560 : HX$=HX$+CHR$(Q%+48-7*(Q%>9))
570 : N=N-Q%*P
580 NEXT
590 IF LEN(HX$)=1 THEN HX$="0"+HX$:REM FORMAT 1 CHARACTER
600 HX$="$"+HX$
610 RETURN
620 REM SETUP SUBROUTINE
630 PRINT"SOUP BY HENRY TROUP & JIM STRASMA
640 PRINT"COMPARES MACHINE-LANGUAGE PROGRAMS
650 REM PRESET VARIABLES TO GAIN SPEED
660 NL$="":ZE$=CHR$(0)
670 INPUT"OUTPUT DEVICE: 3=SCREEN, 4=PRINTER  3===";OT$
680 DV=VAL(OT$):REM CONVERT TO NUMBER
690 IF DV<3 OR DV>4 GOTO 670:REM VALIDATE
700 MF=2:REM 2 FIELDS PER LINE ON SCREEN
710 IF DV<>3 THEN MF=4:REM 4 FOR PRINTER
720 CLOSE 4:OPEN 4,DV:REM HELLO DEVICE
730 INPUT"FILE A'S NAME  +===";CF$
740 INPUT"ON DRIVE  0===";R1
750 IF R1<>0 AND R1<>1 THEN 740:REM VALIDATE
760 DOPEN#1,(CF$),D(R1):REM HELLO FILE A
770 IF DS THEN PRINT DS$:GOTO 730:REM ON ERROR
780 INPUT"FILE B'S NAME  +===";PF$
790 INPUT"ON DRIVE  0===";R2
800 IF R2<>0 AND R2<>1 THEN 790:REM VALIDATE
810 DOPEN#2,(PF$),D(R2):REM HELLO FILE B
820 IF DS THEN PRINT DS$:GOTO 780:REM ON ERROR
830 GET#1,A1$:GET#1,A2$:REM READ A'S LOAD ADDRESS
840 GET#2,B1$:GET#2,B2$:REM & B'S
850 REM TRAP ZERO DATA BUG
860 IF A1$=NL$ THEN A1$=ZE$
870 IF A2$=NL$ THEN A2$=ZE$
880 IF B1$=NL$ THEN B1$=ZE$
890 IF B2$=NL$ THEN B2$=ZE$
900 REM CALCULATE LOAD ADDRESSES
910 AD=ASC(A1$)+ASC(A2$)*256
920 A2=ASC(B1$)+ASC(B2$)*256
930 IF AD=A2 THEN RETURN:REM IF MATCH, BEGIN
940 PRINT"START ADDRESSES DON'T MATCH
950 DCLOSE:REM ELSE CLOSE DISK FILES
960 END:REM & ABORT
```

SOUP Sample Run

```
SOUP: FILE A=SOUP &  FILE B=SOUP 7SE82
@$401,A=$1B+B=$04  @$402,A=$64+B=$00  @$403,A=$8F+B=$20  @$405,A=$20+B=$43
@$406,A=$45+B=$20  @$407,A=$28+B=$41  @$408,A=$44+B=$2C  @$409,A=$4E+B=$2C
@$40A,A=$44+B=$2C  @$40B,A=$50+B=$2C  @$40C,A=$41+B=$32  @$40D,A=$29+B=$00
@$40E,A=$43+B=$04  @$40F,A=$6E+B=$00  @$410,A=$8F+B=$20  @$411,A=$50+B=$52
@$412,A=$49+B=$4F  @$413,A=$52+B=$20  @$414,A=$4C+B=$49  @$415,A=$4E+B=$45
@$416,A=$20+B=$4E  @$418,A=$44+B=$45  @$419,A=$44+B=$20  @$41A,A=$42+B=$59
@$41B,A=$20+B=$44  @$41C,A=$54+B=$4C  @$41D,A=$20+B=$43  @$41E,A=$4F+B=$4D
@$41F,A=$50+B=$49  @$420,A=$4C+B=$45  @$421,A=$52+B=$00  @$422,A=$76+B=$04
```

MICRO

It Pays to Write for MICRO

Get paid for your ideas: write for MICRO! Thousands of people read MICRO every month. MICRO is sold in computer stores and on newsstands worldwide. Send for a copy of our Writer's Guide now. Our author payment rate is competitive with the leading magazines in the industry.

We welcome articles on any aspect of 6502/6809/68000 hardware and software for the Apple, Atari, CBM/PET, TRS-80 Color Computer, VIC, OSI, 6809, or 68000.

1983 Features: March — Printers
April — Communications
May — Wave of New Computers
June — Operating Systems
July — Hardware
August — Word Processing
September — Education
October — Programming Techniques
November — Games
December — New Microprocessors

MICRObits

Deadline for MICRObits: 20th of second month before publication; i.e. December 20th for February issue. Send typewritten copy (40 word limit) with $25.00 per insertion. (Subscribers: first ad at $10.00.)

6800/6809 Software

Includes compatible single user, multi-user and network operating systems, compilers, accounting and word processing packages. Free catalog.

Software Dynamics
2111 W. Crescent, Ste. G
Anaheim, CA 92801

Lessons in Algebra

An easy and fun way to learn the basic elements of high school algebra. Apple computer diskette $29.95. 30 day money-back guarantee if not satisfied.

George Earl
1302 So. General McMullen Dr.
San Antonio, TX 78237

(Continued on page 86)

MICRO™

PET Vet

By Loren Wright

Graphics on the Commodore 64

The Commodore 64 offers a lot of computing power in its small package. There are 64K of RAM, CP/M capability, and sophisticated sound features. But the most outstanding feature is the graphics. To sum it up, the 64 offers considerably more graphics capabilities than the Apple in this area and rivals the Atari 800, at a price that beats them both.

What, exactly, does the 64 do in the way of graphics? I've been studying a preliminary draft of the *Commodore 64 Programmer's Reference Guide* and have begun to learn about all the graphics on my own 64.

The 64 has the following modes, some of which can be mixed on the same screen:

1. Standard character mode
   a. ROM characters
   b. Programmable RAM characters
2. Multicolor character mode (both ROM and RAM)
3. Extended background color mode (both ROM and RAM)
4. Standard bit-map mode (320 × 200 resolution)
5. Multicolor bit-map mode (160 × 200 resolution)
6. Sprites (both standard and multicolor modes)

Various blocks of memory and control registers are involved in pulling off all these different modes. Screen memory consists of 1000 bytes, normally located at $400, and these usually determine what characters will appear on the screen. There is a character ROM, which contains two complete character sets, as on the PET and VIC. Pointers may be altered so that custom characters can be set up in RAM. Color memory, which can't be moved, is 1000 4-bit locations at $D800, each corresponding to a location in screen memory. Four bits is enough to code for sixteen different colors.

The VIC II uses the different bits of two control registers to select nearly all of the graphics modes. Other registers are used to control positions and colors of sprites, to read light pens, and to select background colors. This month's data sheet (p. 109) lists the control registers for the 64. I will refer to them here only by name.

Character Modes

The 64's characters are normally read from the character ROM and the color is determined by the contents of the corresponding location in color memory. The pointer to the character ROM can be altered to point to RAM, where you can design custom characters. There's plenty of memory to play with, so this is a lot more practical than on an unexpanded VIC!

Multicolor character mode has a lot of possibilities. Standard characters consist of eight rows of eight pixels, while multicolor characters consist of eight rows of four double-width pixels. (A pixel is the smallest dot of light on the TV screen in the current graphics mode.) The bits of each byte in character memory are considered in pairs rather than individually. Each of the four possible bit combinations for a bit pair determines where to get the color for the double-wide pixel on the screen. Combinations 00, 01, and 10 get the color from background registers 0, 1, and 2, respectively, and 11 gets the color from the appropriate location in color memory. Since any background color can be changed with a single POKE, parts of all the characters on the screen can be changed at once! This mode is probably best used with custom characters, since this way of interpreting the character data would make most standard characters nearly unrecognizable. The VIC uses a similar scheme in its multicolor mode.

Extended background mode allows the background for each screen location to be any of four different colors. The sacrifice is that only the first 64 characters in character memory can be used. Bits 6 and 7, which would normally select the other 192 characters, determine the background color instead. The background color is read from background color register 0, 1, 2, or 3.

Figure 1. **Multicolor Character Mode a) Bits in character memory are considered in pairs. b) Each bit combination indicates a diferent source for the color. c) The final character displayed with double-width pixels.**

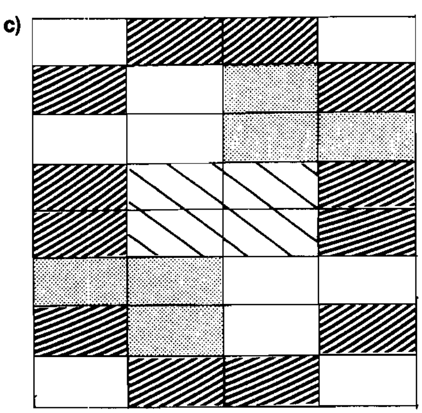
a)

0 0	0 1	0 1	0 0
0 1	0 0	1 0	0 1
0 0	0 0	1 0	1 0
0 1	1 1	1 1	0 1
0 1	1 1	1 1	0 1
1 0	1 0	0 0	0 0
0 1	1 0	0 0	0 1
0 0	0 1	0 1	0 0

b)

00 Background 0
01 Background 1
10 Background 2
11 4-bit color location

c)



Bit-mapped Modes

Standard bit-map (or high-resolution) mode allows control of each individual pixel on the screen, with a resolution of 320 by 200. 8K of RAM, normally taken from the top of BASIC RAM, is used for high-resolution graphics. The bytes are arranged in the same way the pixels of characters are coded. That is, the first byte in hi-res memory codes for the first eight pixels in the first row of pixels on the screen, and the second codes for the first eight pixels in the second row. The ninth byte codes for the ninth through sixteenth pixels of the first row. What this means is that you have to go through a little arithmetic to find the correct bit to change in hi-res memory, given X (in the range of 0 to 319) and Y (in the range of 0 to 199).

Screen memory is used to determine the color of the pixels in the area normally occupied by a character. The high nibble determines the color of all the bits set to 1, and the low nibble determines the color for the 0's.

Multicolor bit-map mode reduces the resolution to 160 by 200. As with multicolor character mode, the bits in hi-res memory are considered in pairs to determine the color of the corresponding double-width pixel on the screen. Combination 00 selects the screen color (background 0), 01 gets the color from the high nibble of the appropriate byte in screen memory, 10 gets the color from the low nibble in screen memory, and 11 gets the color from the 4-bit color memory location.

Commodore plans a VSP Cartridge, which will include convenient commands for high-resolution graphics.

Fine Scrolling

The VIC II chip allows the whole screen to be scrolled up, down, left, or right by only one pixel. To make this work smoothly, there are provisions to reduce the width of the screen to 38 columns and to reduce the height to 24 columns. That allows two columns (and/or one row) to be hidden, while characters are lined up before fine scrolling into the visible area of the screen. The programming for this smooth scrolling is best accomplished with some simple machine-language routines.

Sprites

What is a sprite? The name doesn't really mean much, but the concept is similar to "Player/Missile Graphics" on Atari computers. Each sprite is a high-resolution entity, 24 by 21 pixels, maintained by the VIC II chip. To program one all you need to do is define its bit pattern, select its color, select its X-Y position, and turn it on. By changing the X and Y values you can move the sprite to any position on (or off) the screen.

Now, for the details... Eight sprites may be displayed on the screen at one time. Each sprite has a one-byte pointer at the top of the screen RAM block. The pointer indicates a 64-byte block within the 16K bank currently selected for the VIC II. The last byte of the 64 is a control byte; the others contain the pixel data for the screen representation of the sprite. Each three bytes represent a 24-pixel row in the sprite. In the standard mode, a bit set to 1 displays a pixel of the selected color and a bit set to 0 displays what's under it (usually the background, but it could be part of a sprite of lower priority!).

Associated with each sprite are several other memory locations in the VIC II chip. The sprite display enable register has a bit for each sprite, as do the sprite multicolor enable, sprite expand 2X horizonal, sprite expand 2X vertical, sprite-to-background priority, sprite-to-sprite collision detect, and sprite-to-background registers. Also, there is a byte for each sprite's vertical position, and a byte for each sprite's horizontal position. Since there are more than 256 possible horizontal positions, there is also a byte containing a ninth X-position bit for each sprite. It sounds — and is — complicated. However, this complexity is required to maintain such a powerful graphics mode. Read on for details of the different capabilities of sprite graphics.

Standard sprites can be displayed in any one of the sixteen colors in a resolution equivalent to the standard bit-map mode. Multicolor mode allows up to four colors in each sprite, and the colors are determined by considering bit pairs in the sprite definition. 00 selects screen color, 01 the color in sprite multicolor register #0, 10 the color in the appropriate sprite's color register, and 11 the color in sprite multicolor register #1. As with the other multicolor modes, the horizontal resolution is decreased and the sprites are displayed using double-width pixels.

Each sprite can be expanded to double its horizontal or vertical dimension or both.

To handle smoothly the entry and exit of sprites on the screen, the possible X and Y positions actually extend beyond the visible portion of the screen. That way it is possible to have a corner or an edge appear first, followed smoothly by the rest of the sprite.

I mentioned priorities earlier. The sprites themselves have fixed priorities with respect to each other: sprite 0 is higher priority than sprite 1, 1 higher than 2, and so on. However, each sprite may be selected to be higher or lower in priority with respect to the background data. Objects of higher priority will overwrite objects of lower priority.

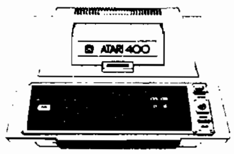
Collisions are detected by the VIC II and appropriate bits are set in two registers. If the corresponding sprite is involved in a collision, then its bit will be set in the register. The bits in the register will remain set until the register's contents are read by your program. Then the whole register is cleared. There is one register for sprite-to-sprite collisions and another for sprite-to-background collisions.

Some of the limitations can be circumvented with more sophisticated programming. For instance, it is possible to display more than eight sprites at once using raster interrupt techniques. Also, because there is so much memory, you can have lots of sprite definitions stored and only alter the pointers. If the fixed sprite priorities are a problem, just swap the pointers and the appropriate bits and registers.

The *Programmer's Reference Manual* gives all the details of the various graphic modes, along with sample programs. Even the little quirks of the system (and ways to get around them) are mentioned. It is good to see Commodore finally paying attention to quality documentation with the VIC-20 and Commodore 64 *Programmer's Reference Guides*. The *Guide* for the 64 should be available in early December.

MICRO

computer mail order west



ATARI

400

16K...... $269
32K...... $349
48K...... $429

800 — 48K

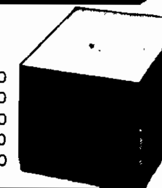
~~$869~~ call

410 Recorder	$ 76.00
810 Disk Drive	$449.00
822 Printer	$269.00
825 Printer	$589.00
830 Modem	$159.00
820 Printer	$259.00
850 Interface	$169.00
CX40 Joysticks (Pair)	$ 18.00
CX853 Atari 16K Ram	$ 77.95

Microtek 16K Ram	$ 74.95
Axlon Ramdisk (128K)	$429.95
Intec 48K Board	$159.00
Intek 32K Board	$ 74.00
One Year Extended Warranty	$ 70.00
CX481 Entertainer Package	$ 69.00
CX482 Educator Package	$130.00
CX 483 Programmer Package	$54.00
CX 484 Communicator Package	$344.00
Atari 800 Dust Cover	$6.99
Atari 400 Dust Cover	$6.99
Atari 810 Dust Cover	$6.99

PERCOM

Disk Drives For Atari Computers



S1 Single Drive	$589.00
A1 Add-on Drive	$339.00
S2 Dual Drive	$879.00
Single Side Dual Head	$679.00
Dual Drive Dual Head	$1046.00

μ-SCI

MICRO-SCI Disk Drives For Franklin & Apple

A2	$319.00
A40	$369.00
A70	$499.00
C2 Controller	$79.00
C47 Controller	$89.00

ATARI

Pac Man	$35.00
Centipede	$35.00
Caverns of Mars	$32.00
Asteroids	$29.00
Missile Command	$29.00
Star Raiders	$35.00

DATASOFT

Pacific Coast Highway	$25.00
Canyon Climber	$25.00
Tumble Bugs	$25.00
Shooting Arcade	$25.00
Clowns and Balloons	$25.00
Graphic Master	$30.00
Graphic Generator	$13.00
Micro Painter	$25.00
Text Wizard	$89.00
Spell Wizard	$64.00
Bishop's Square	$25.00

ON-LINE

Jawbreaker	$27.00
Softporn	$27.00
Wizard and the Princess	$29.00
The Next Step	$34.00
Mission Asteroid	$22.00
Mouskattack	$31.00

SYNAPSE

File Manager 800	$79.00
Chicken	$26.00
Dodge Racer	$26.00
Synassembler	$30.00
Page 6	$19.00
Shamus	$26.00
Protector	$26.00
Nautilus	$26.00
Slime	$26.00
Disk Manager	$24.00

K-BYTE

Krazy Shoot Out	$32.00
K-razy Kritters	$32.00
K-razy Antics	$32.00
K-star Patrol	$32.00

STICK STAND $6.99

VISICORP

For Apple, IBM, Franklin

Visidex	$189.00
Visifile	$189.00
Visiplot	$159.00
Visiterm	$189.00
Visitrend/Plot	$229.00
VisiSchedule	$229.00
Desktop Plan	$189.00
VISICALC	$179.00

for Apple II plus, Atari, CBM & IBM

Continental

The Home Accountant (Apple/Franklin)	$59.00
The Home Accountant (IBM)	$119.00
1st Class Mail	$59.00

FLOPPY DISKS

Maxell

MD I (Box of 10)	$36.00
MD II (Box of 10)	$46.00
MFD I (8")	$44.00
MFD II (8" Double Density)	$54.00

Verbatum

5 1/4" SS DD	$26.00
5 1/4" DS DD	$36.00

Elephant

5 1/4" SS DD	$19.99

TIMEX

TIMEX SINCLAIR 1000

LOWEST PRICE EVER! $89.00

NORTH STAR ALTOS

Call for price and availability on all models.

MODEMS

Hayes

Smart	$239.00
Smart 1200 (1200 baud)	$549.00
Chronograph	$199.00
Microdem II	$279.00
Microdem 100	$309.00

Novation

Cat	$144.00
D-Cat	$159.00
Auto Cat	$219.00
212 Auto Cat	$589.00
Apple Cat II	$339.00
212 Apple Cat II	$609.00

Anchor

Mark I (RS-232)	$79.00
Mark II (Atari)	$79.00
Mark III (TI-99)	$109.00
Mark IV (CBM/PET)	$125.00
Mark V (OSBORNE)	$95.00
Mark VI (IBM-PC)	$179.00
Mark VII (Auto Answer/Dial)	$119.00
9 Volt Power Supply	$9.00

MONITORS

AMDEK

300G	$169.00
Color I	$339.00
Color II	$699.00
Color III	$429.00

BMC

12" Green	$ 85.00
13" Color 1400	$279.00
13" Color 1401 (Mid Res)	$369.00

ZENITH

ZVM 121	$ 99.00

SHARP

Sharp 13" Color TV	$275.00

PANASONIC

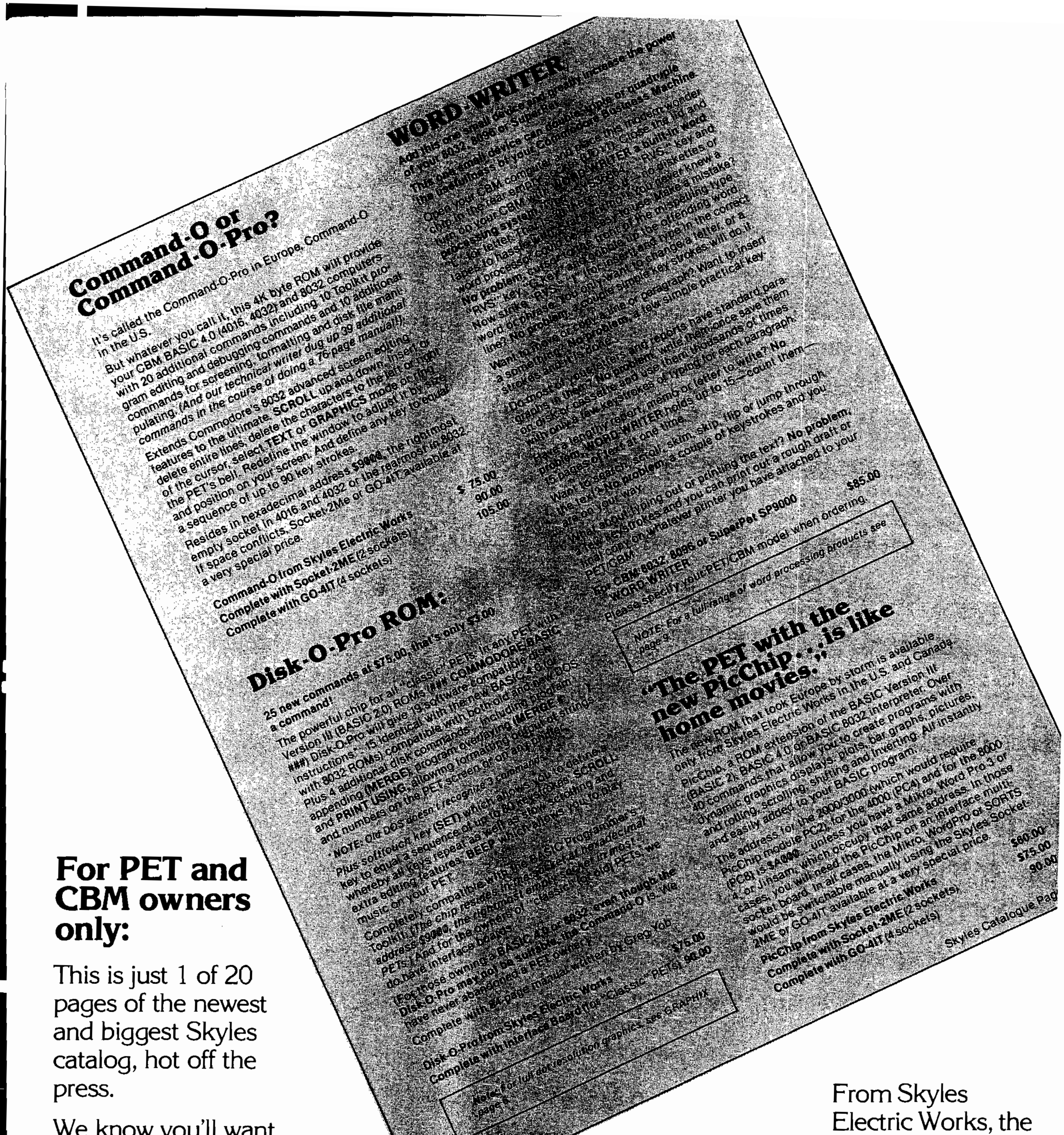
TR-120 MIP (High Res. Green)	$159.00
CT-160 Dual Mode Color	$299.00

west **800-648-3311** west

IN NV. CALL (702) 588-5654

P.O.BOX 6689 STATELINE, NV. 89449

In-stock items shipped same day you call. No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the continental United States with no waiting period for certified checks or money orders. Add 3% (minimum $3.00) shipping and handling on all C.O.D. and Credit Card orders. NV and PA residents add sales tax. All items subject to availability and price change. **NOTE:** We stock manufacturer's and third party software for most all computers on the market! Call today for our new catalogue.



Command-O or Command-O-Pro?

It's called the Command-O-Pro in Europe, Command-O in the U.S.

But whatever you call it, this 4K byte ROM will provide your CBM BASIC 4.0 (4016, 4032) and 8032 computers with 20 additional commands including 10 Toolkit program editing and debugging commands and 10 additional commands for screening, formatting and disk file manipulating. *(And our technical writer dug up 39 additional commands in the course of doing a 76-page manual!)*

Extends Commodore's 8032 advanced screen editing features to the ultimate. **SCROLL** up and down, insert or delete entire lines, delete the characters to the left or right of the cursor, select **TEXT** or **GRAPHICS** mode or ring the PET's bell. Redefine the window to adjust it by size and position on your screen. And define any key to equal a sequence of up to 90 key strokes.

Resides in hexadecimal address $9000, the rightmost empty socket in 4016 and 4032 or the rearmost in 8032. If space conflicts, Socket-2Me or GO-4IT available at a very special price.

Command-O from Skyles Electric Works $75.00
Complete with Socket-2ME (2 sockets) 90.00
Complete with GO-4IT (4 sockets) 105.00

WORD-WRITER

For CBM 8032, 8096 or SuperPet SP9000 $85.00

Please specify your PET/CBM model when ordering.

Disk-O-Pro ROM:

25 new commands at $75.00, that's only $3.00 a command!

Disk-O-Pro from Skyles Electric Works $75.00
Complete with Interface Board (for "classic" PETs) 90.00

"The PET with the new PicChip... is like home movies."

PicChip from Skyles Electric Works $60.00
Complete with Socket-2ME (2 sockets) 75.00
Complete with GO-4IT (4 sockets) 90.00

Skyles Catalogue Page

For PET and CBM owners only:

This is just 1 of 20 pages of the newest and biggest Skyles catalog, hot off the press.

We know you'll want this page, in its full 8½ x 10 splendor, and another 19 pages of peripherals, software and books that will make your PET or CBM computer even nicer to live with. So, if we missed sending you your very own copy within the last few weeks, call us at **(800) 227-9998** **(unless you live in California, in which case call (415) 965-1735.**

From Skyles Electric Works, the oldest and largest professional specialists in the business.

Skyles Electric Works
231-E S. Whisman Road
Mountain View, California 94041

Microcomputers in a College Teaching Laboratory, Part 2

by Richard Heist, Thor Olsen, and Howard Saltsburg

Many laboratory situations involve measuring continuous ranges of light, heat, and sound. An inexpensive device to help the digital computer deal with these analog quantities is the analog transducer. Specific applications to temperature and light intensity measurement are discussed.

Part 1 of this series (MICRO 53:53)[1] gave an overview of the microcomputer laboratory program at the University of Rochester, Department of Chemical Engineering. In this article the problems of measuring physical, chemical, and mechanical properties will be addressed, since such problems are common to most engineering and scientific laboratories. Temperature, pressure, flow, and light intensity are typical quantities of interest, and in many cases the required information is provided by a transducer in the form of an analog signal, usually electrical in nature. Difficulties in the measurement and conversion to the desired physical or chemical quantity of these signals may tend to obscure the purpose of the measurement. The microcomputer often offers a simpler alternative to more conventional laboratory instrumentation, thus making it easier for the user to maintain a focus on the purpose of the measurements. Furthermore, it combines this decrease in complexity with low cost, high speed, reliability, and precision.

In what follows, the use of simple interfacing devices will be discussed. These devices were selected for their flexible operating characteristics, which give them quite general utility. Examples will illustrate their application to the measurement of temperature and light intensity. The emphasis will be on specific applications, not on design or construction of the devices, which are very simple.

Analog Signals and A/D Converters

When the transducer of interest produces an electrical signal, the problem of property measurement is reduced to one of measuring that signal (usually voltage, current, or resistance) to the desired degree of accuracy and at an appropriate rate. Many laboratory measurements require only slow (< 50 Hz) data acquisition rates or low (8-bit) precision. The actual requirements should be evaluated carefully and realistically since they have an important bearing on the technique and instrumentation used to measure the electrical quantities.

When high-speed data acquisition and high resolution are not needed, it is remarkably easy to interface many laboratory experiments and measuring devices to the computer. As will be demonstrated, an appropriate A/D converter, selected for its flexibility, combined with a microcomputer and a high-resolution dot matrix printer, becomes a versatile data acquisition system (the universal instrument referred to in the first article in this series (MICRO 53:53). This combination can be used effectively and inexpensively to solve many laboratory measurement problems.

The two types of A/D converters, which have been widely used in the Rochester program, both employ a pulse-width technique for data conversion, even though one is used to measure voltage and the other resistance. Each device, upon command from the computer (a trigger pulse) begins a timing cycle, the length of which is proportional to the magnitude of the applied analog signal. At the end of the cycle, the converter signals the computer that conversion is complete (end of conversion, EOC).

The computer is programmed to measure the length of the timing cycle by repeatedly incrementing the microprocessor index registers until the EOC signal is received. The microprocessor requires a fixed number of machine cycles to run through the program loop in which it tests for EOC and increments the index registers. Since these cycles are accurately timed by the internal crystal oscillator, the count accumulated in the index registers is proportional to the elapsed time. By suitable calibration, this count can be converted to the desired data format, and the measurement is complete.

Typical resolution can range from eight to 12 bits; the corresponding conversion times are approximately three to 200 milliseconds. The ability to trade off conversion time for resolution gives these simple devices a flexibility not shared by other kinds of A/D converters and makes them feasible for many laboratory applications.

The device used for voltage measurements is a QM-100 A/D converter (Analog Systems, P.O. Box 35879, Tucson AZ). This device has three independent A/D channels, each with a 0 to 10 VDC input range. In operation, a voltage ramp generator is triggered by the computer, and its output is compared to the transducer voltage. A comparator signals the computer when the ramp just exceeds the transducer voltage (EOC).

For resistance measurements, a simple A/D method outlined in an article in MICRO[2] was chosen. It uses a 555 timer IC in the configuration shown in figure 1. The conversion method involves charging the timing capacitor, C1, to a fixed voltage through the transducer resistance, R, and measuring the charging time with

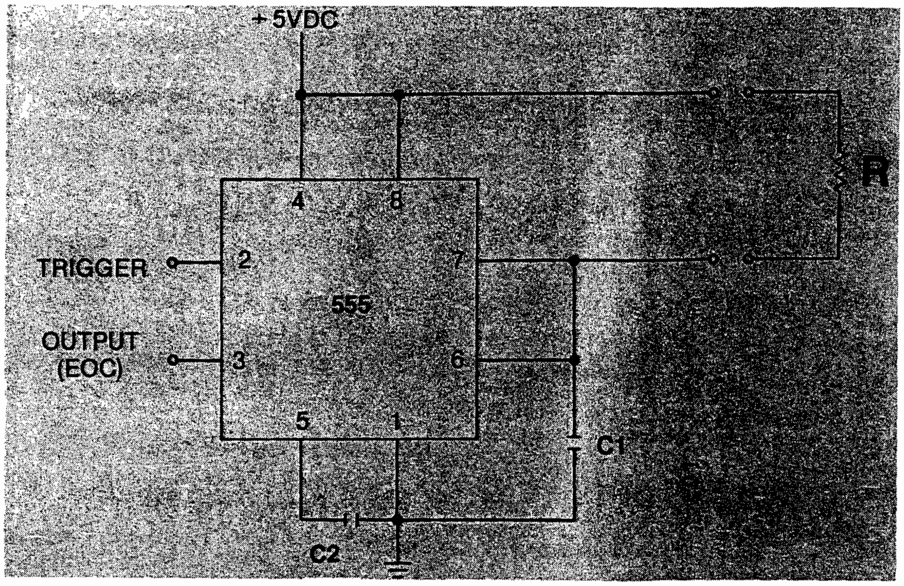


Figure 1: **A 555 timer integrated circuit wired as a monostable multivibrator. A typical value for C2 is .01 μF. The value chosen for C1 depends upon R. For instance, if R = 150 KΩ and 10-bit conversion is desired (1024 counts, see text), then C1 should be about 0.1 μF (see reference 3).**

the computer. The computer triggers the charging process and is then signaled by the 555 timer when conversion is complete. By choosing the appropriate combination of transducer and timing capacitor for a specific application[3], you have a simple and inexpensive data acquisition system.

While the examples described here are specific to temperature and light-intensity measurements, the concepts are general. These interfacing methods can be extended to virtually any kind of voltage or resistance measurement. Moreover, it is clear that the use of a resistance transducer, when appropriate, can result in a significant simplification of hardware, compared to other techniques, and it will often pay to change to sensors of this type.

One additional point that should be made in connection with the pulse-width A/D converters is the ease with which these devices can be multiplexed. Many times it is necessary to measure a number of inputs simultaneously. Since most microcomputers will support only a limited number of I/O lines, it is useful to be able to switch-select devices automatically (multiplex). Examples of this include the simultaneous monitoring of the temperature of each tray of a multistage distillation column and multiple concentration profile measurements along a tubular reactor. The circuit shown in figure 2 has been used to multiplex the sensors in several experiments. It is based on the 74150 IC, a 16-channel multiplexer. A similar circuit, based on the 75151 IC, can be used to construct an 8-channel device. Both multiplexer ICs and their operation are described in detail in the literature listed in reference 4.

Construction details have not been discussed at length since they are adequately described in the microcomputer and electronics literature[5], but good construction techniques must not be underemphasized, particularly for applications requiring higher precision. The important construction practices are documented in the literature and are well known to experienced personnel. Do not hesitate to ask for advice.

Some care should be exercised in the use of the converters. For instance, the characteristics of all electronic components are, to some extent, temperature-dependent. Therefore, large fluctuations in ambient temperature should be avoided during data collection or between calibration and actual use. Another point concerns the use of the 555-based converter in the triggered mode described above. When the EOC is reached, the 555 IC starts discharging the timing capacitor and the system will remain in discharge mode until it is triggered again. If the time between EOC and the next trigger pulse varies, the circuit may operate with varying levels of residual charge on the timing capacitor. The result will be timing er-

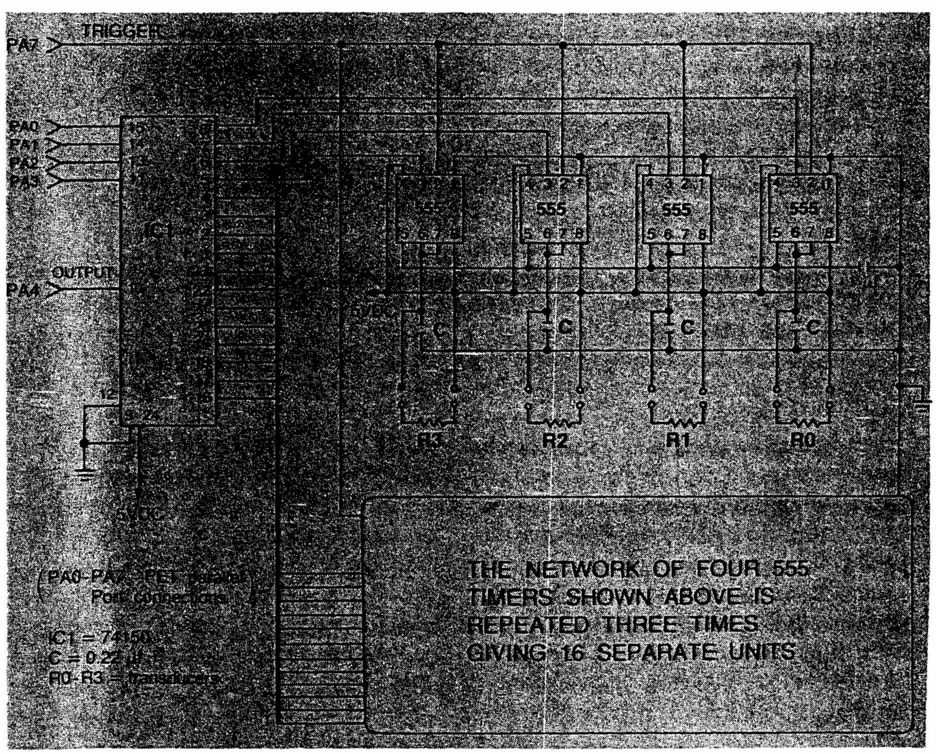


Figure 2: **A 16-channel multiplexer circuit based on a 74150 TTL integrated circuit. The end-of-conversion signal, pin 3, of any of the 555 timers can be accessed by placing the appropriate binary number (0-15) on the input pins (15, 14, 13, and 11, respectively) of the 74150. In the diagram, PA0 - PA4 and PA7 represent PET parallel port connections. The output from the 74150 is available at pin 10. The resistance value of the transducers, R0 - R15, will determine the value of the charging capacitor, C (see figure 1). A typical value is 0.22 μF (see reference 3).**

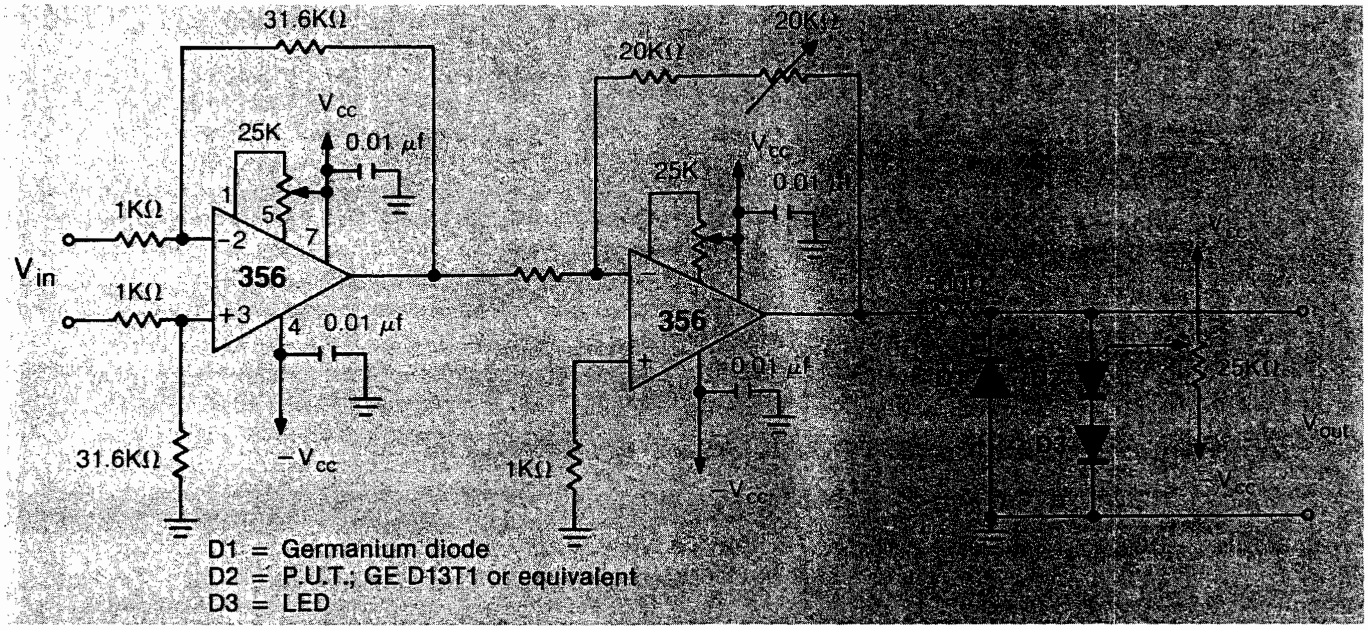


Figure 3: **A two-stage voltage amplifier. The overall gain ranges from 630 to 1260, depending upon the setting of the 20 KΩ variable resistor in the feedback loop of the second stage. The optional diode network ensures that the output voltage will be positive (D1) and will not exceed 10VDC (D2). This is a requirement for proper operation of the QM-100 A/D converter. D3 is used to indicate over-ranging.**

rors, leading to poor reproduction of the data. The problem can be circumvented by introducing a sufficient delay between measurements to assure total discharge, or by operating the system with reproducible discharge time.

Temperature Measurement

Two analog electrical signals commonly associated with temperature are thermocouple voltage and thermistor resistance. The problem is to provide a convenient method for measuring these analog signals, then convert the results to temperature.

Consider, for example, a temperature measurement in which a precision of one degree Celsius is desired at a temperature of 100 degrees. If the sensor is a thermocouple, the transducer output will be in the low millivolt range and a difference of one degree in temperature would produce a voltage difference of, at most, a few tens of microvolts — beyond the direct resolution of most analog meters. As the precision requirement of an experiment increases, conventional thermocouple instrumentation becomes costly.

With digital instrumentation, this precision is not difficult to achieve. Provided the input signal at 100 degrees is within the upper half of the converter's input range, all that is required is an eight-bit A/D converter. An obvious problem, then, in interfacing thermocouples (and many other laboratory devices as well) is the low level of the output voltages. The millivolt-level signals generally available must be amplified to the 0.5 to 10 VDC range before A/D conversion can be performed satisfactorily. Fortunately, the frequency response requirements are minimal for most applications, so large-gain amplifiers (100X - 2000X) are relatively simple to build[6]. See figure 3 for a typical example. When adjustable gain is included, the combination amplifier and QM-100 converter becomes an A/D system that is inexpensive, versatile, and reliable.

Thermistors, in contrast to thermocouples, can be manufactured to provide large resistance changes for small temperature differences. Unfortunately, the response is highly non-linear, and the response characteristics tend to be non-uniform, even among thermistors of the same kind. These properties make it difficult and expensive to reduce thermistor output to temperature with analog hardware. Using a microcomputer with the 555 timer A/D, on the other hand, you can easily handle these complex relationships with appropriate software modifications.

Light-Intensity Measurement

Another property commonly measured in laboratories is light intensity. In chemical laboratories, this measurement is usually made with commercially available instrumentation equipped with photocells or photomultiplier tubes (e.g., colorimeters and spectrophotometers). It has proven to be easy to use either the QM-100 or the 555 converter to interface the microcomputer to such optical instruments. In fact, inexpensive colorimeters based on a 555 timer/photoresistor circuit can be built to almost any geometry required by an intended application.

For photomultiplier-equipped spectrophotometers where the output signal is a current, a simple circuit can be used to convert the transducer output to a voltage[6]. A typical example of a current-to-voltage converter circuit is shown in figure 4. Once a voltage is available, the procedure for using the QM-100 is the same as described above.

A major use of this type of optical instrumentation is in measuring the concentration of light-absorbing chemicals in liquids and gases. Normally, the response of such instruments is proportional to the inverse exponential function of the concentration. Thus, should a linear response be required when using a chart recorder for data acquisition, an expensive linearizing module must be added.

In some cases, not only is a linear response required, but the quantity of interest is the total amount of a chemical that has passed through the detector. This type of measurement requires the capability to integrate a response over time — another module to add to the recorder.

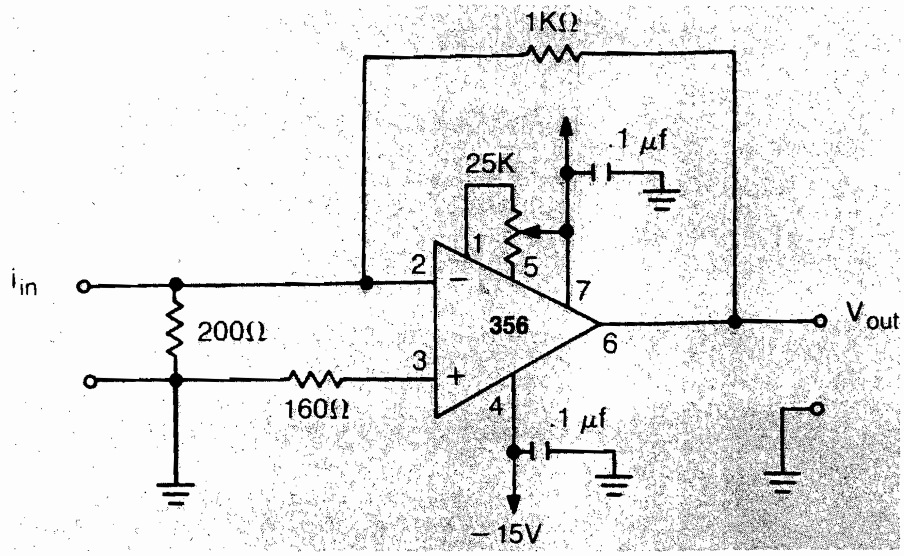


Figure 4: **A current to voltage converter. The circuit shown here will typically produce millivolt-level output for microampere-level input with good frequency response.**

When the microcomputer is used to monitor such instruments, these conversions require only a few lines of additional code in the applications program. Within the limits of the microcomputer's capabilities, any relationship between sensor output and the quantity of interest can be accommodated without additional cost as long as the relationship can be adequately described by mathematical expressions. Also, since the computer can store spectral data between scans, it is possible through computer interfacing to convert a single-beam spectrophotometer into a pseudo dual-beam device.

The simplicity of microcomputer-based systems can best be illustrated by the measurement of optical density of fluids. An extremely simple colorimeter, useful for many chemical concentration measurements, can be constructed from a suitable light source, such as a light-emitting diode, and a photoresistor, placed on opposite sides of a translucent vessel containing the fluid to be studied. The photoresistor is interfaced *via* the 555 A/D converter. Since the components (light source and photoresistor) can be very small, e.g. three mm diameter, and the units are so simple, a variety of geometries can be accommodated. Thus, a chemical reaction involving a color change can be followed *in situ* in a small test tube. There is no need to disturb the process by withdrawing samples for analysis.

Another example is the study of the dispersion of a dye in a liquid flowing in a long tube. It is a simple matter to place these LED-photoresistor colorimeters in collars clamped around the tube, at intervals, and observe the dispersion effect without disturbing the flow.

Note that when a LED is used in

this mode it is important that it is supplied a constant current. A simple circuit that will accomplish this[7] is shown in figure 5.

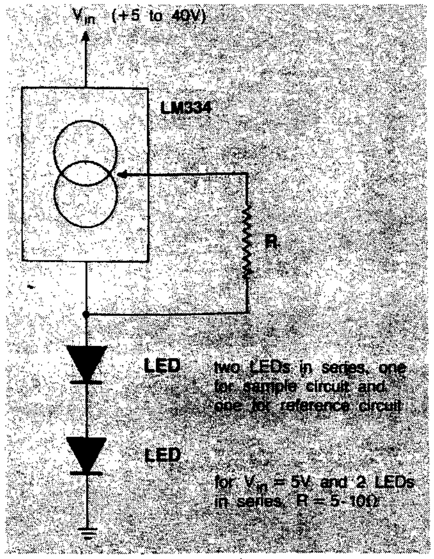


Figure 5: **A current regulator. The LM334 is an adjustable current source with good current regulation. A typical value for R with two LEDs in series is 5 to 10 ohms. The two LEDs in series are used to provide a sample signal and a reference signal for the colorimeter applications discussed in the text.**

Concluding Comments

The general utility of the A/D converter (computer) printer combination deserves reiteration. In going from one application to another, only portions of the applications program need to be changed; the data acquisition routines remain unaltered. The A/D devices previously described can be adapted to a variety of resistance, voltage, and current measurements with little or no modification. The flexibility of these A/D converters, the computational capability of the microcomputer in the reduction of data, and the high-resolution hard copy capability of the dot-matrix printer are combined to make the system an inexpensive but powerful universal data acquisition instrument.

Once it is realized that resistance and voltage can be measured so easily with the microcomputer, you may wish to redesign existing experiments to match the output to the interface, rather than the other way around. In particular, it may be advantageous to generate resistance, rather than current or low-level voltage; e.g., use thermistors instead of thermocouples.

At moderate expense, the system can be expanded further to provide the capability to feed back information and change the operating conditions of the device it monitors. Digital to analog conversion and control will be discussed in a subsequent paper.

The role of the computer in the laboratoy is that of a tool. Certainly it is a remarkable tool in terms of power and capability; but nevertheless, it is a means to an end and not the end in itself. This point is sometimes too easily forgotten.

References

1. H. Saltsburg, R.H. Heist, and T. Olsen, *MICRO, The 6502/6809 Journal*, (53:53).
2. J. Sherburne, *MICRO, The 6502/6809 Journal*, (26:31).
3. See, for example, H. Berlin, "The 555 Timer Applications Sourcebook with Experiments," (Howard W. Sams and Co. Inc., Indianapolis, 1979).
4. See, for example, "Signetics Logic — TTL Data Manual," (Uniplan, San Francisco, 1978); "The TTL Data Book," (Texas Instrument, Inc., 1976), 2nd ed.; D. Lancaster, "The TTL Cookbook," (Howard W. Sams and Co. Inc., Indianapolis, 1979).
5. See, for example, P. Horowitz and W. Hill, "The Art of Electronics," (Cambridge University Press, Cambridge, 1980); F.M. Mims, "Engineer's Notebook II. Integrated Circuit Applications," (Tandy Corporation, 1981); Z.H. Meiksin and P.C. Thackary, "Electronic Design with Off-the-Shelf Integrated Circuits," (Parker Publishing Co. Inc., West Nyack, NY, 1980); S.A. Hoenig, "How to Build and Use Electronic Devices without Frustration, Panic, Mountains of Money or an Engineering Degree," (Little, Brown and Co., Boston, 1980) 2nd ed.
6. See, for example, W. Jung, "IC Op-amp Cookbook," (Howard W. Sams and Co. Inc., Indianapolis, 1979); "Operational Amplifiers: Design and Application," (McGraw-Hill Book Company, New York, 1971), edited by J.G. Graeme, G.E., Tobey, and L.P. Huelsman.
7 F.M. Mims, "Engineer's Notebook II. Integrated Circuit Applications," (Tandy Corporation, 1982) p. 116.

You may contact the authors at the Department of Chemical Engineering, University of Rochester, Rochester, NY 14627.

AICRO

Apple Slices

By Tim Osborn

One of the fastest techniques that lets you search for a specific occurrence of an item within a sorted set is the binary search. This month's column presents a subroutine (BINARY-SEARCH) that you may call from your BASIC programs to perform a binary search on a sorted (ascending) string array. The advantages of a binary search over a serial search increase as the number of items in the array grows. For example, an array of 4096 items can be searched in less than 11 tries.

The Method

A binary search tests the middle element in the remaining part of the array. If the element is higher than the search argument (the value being searched for), the part of the array from this element upward is left out of the search by resetting the upper limit to the index of the element. If the element is lower than the search argument, the part of the array from this element downward is left out by resetting the lower limit to the index of this element. The program then finds the average of the upper limit and the lower limit and searches the element at this location. The procedure continues until the element is found or until it discovers that the upper and lower limits have converged without finding the element.

The Subroutine

The syntax for the binary search is:

& GET (XX$,YY$)

where 1. XX$ represents any legal string array name, and 2. YY$ represents any legal string variable name. This subroutine will return in SS% the index number of the element in XX$ that has a value equal to YY$ if the item is found. If the item is not found the subroutine will return a −1

```
               ;*********************************
               ;*          APPLE SLICES         *
               ;*  B I N A R Y — S E A R C H    *
               ;*           T. S. O.            *
               ;*********************************
               ;ZERO PAGE EQUATES
               LOWTR    EPZ $9B           ;WORK POINTER
               VARNAM   EPZ $81           ;CONTAINS LAST USED VARIABLE NAME
               VARAD    EPZ $83           ;ADDRESS OF PASSED STRING
               CHRGET   EPZ $B1           ;APPLESOFT'S ROUTINE TO GET A BYTE
               ;
               ;EQUATES
               ;
               AMPERV   EQU $3F5          ;AMPERSAND VECTOR LOCATED HERE
               FIND     EQU $E053         ;ROUTINE TO LOCATE VARIABLE DESCRIPTOR
               CHKOPN   EQU $DEBB         ;CHECK FOR OPEN PAREN
               GETARYPT EQU $F7D9         ;ROUTINE TO FIND ARRAY DESCRIPTOR
               CHKCOM   EQU $DEBE         ;CHECK FOR COMMA
               SYNERR   EQU $DEC9         ;DISPLAY SYNTAX ERROR
               DATA     EQU $D995         ;ADVANCE TXTPTR TO END OF STATEMENT
               ;
9400                    ORG $9400
9400                    OBJ $800          ;FOR LISA
9400           ;
9400 A9 4C     SETVEC   LDA #$4C          ;JUMP ABSOLUTE INSTRUCTION
9402 8D F5 03           STA AMPERV
9405 A9 10              LDA #ENTRY        ;LSB OF ENTRY ADDRESS
9407 8D F6 03           STA AMPERV+1
940A A9 94              LDA /ENTRY        ;MSB OF ENTRY ADDRESS
940C 8D F7 03           STA AMPERV+2
940F 60                 RTS
9410           ;
9410 20 B1 00  ENTRY    JSR CHRGET        ;GET CHARACTER
9413 20 BB DE           JSR CHKOPN        ;SHOULD BE OPEN PAREN
9416 20 D9 F7           JSR GETARYPT      ;GET ARRAY DESCRIPTOR
9419 A0 04              LDY #4
941B B1 9B              LDA (LOWTR),Y     ;SHOULD BE A ONE DIMENSION ARRAY
941D C9 01              CMP #1
941F F0 03              BEQ ENTRY1
9421 4C C9 DE           JMP SYNERR        ;ELSE DISPLAY ERROR MESSAGE
9424 A5 9B     ENTRY1   LDA LOWTR         ;SAVE ARRAY DESCRIPTOR ADRS.
9426 8D 7B 95           STA SAVARRAY      ;LSB
9429 A5 9C              LDA LOWTR+1
942B 8D 7C 95           STA SAVARRAY+1    ;MSB
942E 20 BE DE           JSR CHKCOM        ;CHK FOR COMMA + LOAD A W/NEXT BYTE
9431 85 81              STA VARNAM
9433 20 B1 00           JSR CHRGET        ;GET NEXT BYTE
9436 D0 03              BNE ENTRY2        ;SHOULD NOT BE END OF STATEMENT
9438 4C C9 DE  ERROR    JMP SYNERR        ;DISPLAY SYNTAX ERROR MESSAGE
943B C9 24     ENTRY2   CMP #'$'          ;DOLLAR SIGN
943D D0 02              BNE NAMLNG        ;NO, MUST BE TWO CHARACTER NAME
943F A9 00              LDA #$00
9441 09 80     NAMLNG   ORA #$80          ;NEGATIVE ASCII
9443 85 82              STA VARNAM+1
9445 20 53 E0           JSR FIND          ;FIND DESCRIPTOR
9448 A0 02              LDY #2
944A B1 9B              LDA (LOWTR),Y     ;GET + SAVE THE
944C 8D 74 95           STA VARLN         ;LENGTH OF PASSED STRING
944F C8                 INY
9450 B1 9B              LDA (LOWTR),Y     ;GET + SAVE THE
9452 85 83              STA VARAD         ;ADDRESS OF PASSED STRING
9454 C8                 INY
9455 B1 9B              LDA (LOWTR),Y
9457 85 84              STA VARAD+1
9459 AD 7B 95           LDA SAVARRAY      ;REESTABLISH LOWTR TO
945C 85 9B              STA LOWTR         ;ADDRESS OF ARRAY DESCRIPTOR
945E AD 7C 95           LDA SAVARRAY+1
9461 85 9C              STA LOWTR+1
9463 A0 05              LDY #5            ;GET UPPER LIM. OF DIM (LOW BYTE)
9465 B1 9B              LDA (LOWTR),Y
9467 8D 78 95           STA UPLIM+1       ;MAKE LOW-HIGH
946A C8                 INY
946B B1 9B              LDA (LOWTR),Y
946D 8D 77 95           STA UPLIM
9470 A9 00              LDA #$00          ;INITIALIZE LOWER LIMIT
9472 8D 79 95           STA LOWLIM
9475 8D 7A 95           STA LOWLIM+1
9478 20 31 95  SEARCHLP JSR COMPIDX       ;INDEX=(UPLIM+LOWLIM)/2
947B 20 56 95           JSR BY3           ;MULTIPLY INDEX BY 3 (LENGTH OF PTR. ENTRIES)
947E 18                 CLC
947F A5 9B              LDA LOWTR         ;ADD BASE TO INDEX
9481 6D 7B 95           ADC SAVARRAY
9484 85 9B              STA LOWTR         ;TO OBTAIN POINTER TO ELEMENT
9486 A5 9C              LDA LOWTR+1
9488 6D 7C 95           ADC SAVARRAY+1
948B 85 9C              STA LOWTR+1
948D A0 07              LDY #7            ;OFFSET TO LENGTH OF ELEMENT
948F B1 9B              LDA (LOWTR),Y
9491 8D 7D 95           STA ARRAYLN
9494 CD 74 95           CMP VARLN         ;FIND SHORTEST ARGUMENT
```

```
9497 30 06       91            BMI ARRAYST        ;ELEMENT SHORTEST
9499 AE 74 95    92            LDX VARLN          ;STRING SHORTEST
949C 4C A0 94    93            JMP CONT1
949F AA          94  ARRAYST   TAX                ;PUT ELEMENT LENGTH IN X
94A0 C8          95  CONT1     INY                ;OFFSET TO ADDRESS
94A1 B1 9B       96            LDA (LOWTR),Y      ;GET LOW BYTE OF ADDRESS
94A3 8D 7F 95    97            STA ARRAYAD
94A6 C8          98            INY
94A7 B1 9B       99            LDA (LOWTR),Y      ;GET HIGH BYTE
94A9 8D 80 95   100            STA ARRAYAD+1
94AC A0 00      101            LDY #$00           ;INITIALIZE Y
94AE AD 7F 95   102            LDA ARRAYAD        ;SET UP LOWTR AS
94B1 85 9B      103            STA LOWTR          ;ZERO PAGE PTR. FOR ARRAYAD
94B3 AD 80 95   104            LDA ARRAYAD+1
94B6 85 9C      105            STA LOWTR+1
94B8 B1 9B      106  COMPLP    LDA (LOWTR),Y      ;COMPARE ARRAY TO
94BA D1 83      107            CMP (VARAD),Y      ;STRING
94BC 30 2F      108            BMI STRNGHI        ;STRING IS GREATER
94BE F0 03      109            BEQ COMP1
94C0 4C 0F 95   110            JMP STRNGLO        ;STRING IS LOWER
94C3 C8         111  COMP1     INY
94C4 CA         112            DEX
94C5 D0 F1      113            BNE COMPLP         ;CONTINUE COMPARE
94C7 AD 7D 95   114            LDA ARRAYLN
94CA CD 74 95   115            CMP VARLN          ;COMPARE STRING + ELEMENT LENGTH
94CD 30 1E      116            BMI STRNGHI        ;IF STRING IS LONGER
94CF F0 03      117            BEQ EXIT           ;FOUND THE ELEMENT
94D1 4C 0F 95   118            JMP STRNGLO        ;STRING IS SHORTER
94D4 A9 D3      119  EXIT      LDA #$D3           ;FIND OR CREATE A DESCRIPTOR
94D6 85 81      120            STA VARNAM         ;FOR SS% INTEGER
94D8 85 82      121            STA VARNAM+1
94DA 20 53 E0   122            JSR FIND           ;BY JSR TO FIND
94DD A0 02      123            LDY #2
94DF AD 76 95   124            LDA INDEX+1        ;STORE HIGH BYTE OF INDEX
94E2 91 9B      125            STA (LOWTR),Y      ;FIRST
94E4 C8         126            INY
94E5 AD 75 95   127            LDA INDEX          ;THEN LOW BYTE
94E8 91 9B      128            STA (LOWTR),Y
94EA 4C 95 D9   129            JMP DATA           ;RESET TXTPTR + RETURN TO BASIC
94ED AD 79 95   130  STRNGHI   LDA LOWLIM         ;IF LOWLIM = INDEX
94F0 CD 75 95   131            CMP INDEX          ;THAN ELEMENT CAN'T BE FOUND
94F3 D0 0B      132            BNE HI2
94F5 AD 7A 95   133            LDA LOWLIM+1
94F8 CD 76 95   134            CMP INDEX+1
94FB D0 03      135            BNE HI2
94FD 4C 4B 95   136            JMP NOTFOUND       ;SO BRANCH TO NOTFOUND RTN.
9500 AD 75 95   137  HI2       LDA INDEX          ;RESET LOWER LIMIT
9503 8D 79 95   138            STA LOWLIM
9506 AD 76 95   139            LDA INDEX+1
9509 8D 7A 95   140            STA LOWLIM+1
950C 4C 78 94   141            JMP SEARCHLP       ;CONTINUE SEARCH
950F AD 77 95   142  STRNGLO   LDA UPLIM          ;IF UPLIM=INDEX
9512 CD 75 95   143            CMP INDEX          ;THEN ELEMENT CAN'T BE FOUND
9515 D0 0B      144            BNE LO2
9517 AD 78 95   145            LDA UPLIM+1
951A CD 76 95   146            CMP INDEX+1
951D D0 03      147            BNE LO2
951F 4C 4B 95   148            JMP NOTFOUND       ;SO BRANCH TO NOTFOUND ROUTINE
9522 AD 75 95   149  LO2       LDA INDEX          ;RESET UPPER LIMIT
9525 8D 77 95   150            STA UPLIM
9528 AD 76 95   151            LDA INDEX+1
952B 8D 78 95   152            STA UPLIM+1
952E 4C 78 94   153            JMP SEARCHLP       ;CONTINUE SEARCH
9531            154  ;
9531            155  ;COMPUTE NEW INDEX
9531 18         156  COMPIDX   CLC                ;INDEX=(UPLIM+LOWLIM)/2
9532 AD 77 95   157            LDA UPLIM          ;ADD UPLIM TO LOWLIM
9535 6D 79 95   158            ADC LOWLIM
9538 8D 75 95   159            STA INDEX          ;AND STOR IN INDEX
953B AD 78 95   160            LDA UPLIM+1
953E 6D 7A 95   161            ADC LOWLIM+1
9541 8D 76 95   162            STA INDEX+1
9544 4E 76 95   163            LSR INDEX+1        ;DIVIDE BY TWO
9547 6E 75 95   164            ROR INDEX
954A 60         165            RTS
954B A9 FF      166  NOTFOUND  LDA #$FF           ;-1 MEANS NOTFOUND
954D 8D 75 95   167            STA INDEX
9550 8D 76 95   168            STA INDEX+1
9553 4C D4 94   169            JMP EXIT
9556            170  ;
9556 AD 75 95   171  BY3       LDA INDEX          ;LOWTR=(INDEX*3)
9559 85 9B      172            STA LOWTR
955B 06 9B      173            ASL LOWTR          ;(LOWTR*2)
955D AD 76 95   174            LDA INDEX+1
9560 85 9C      175            STA LOWTR+1
9562 26 9C      176            ROL LOWTR+1
9564 18         177            CLC
9565 AD 75 95   178            LDA INDEX      =tP  STA LOWTR
956C AD 76 95   181            LDA INDEX+1
956F 65 9C      182            ADC LOWTR+1
9571 85 9C      183            STA LOWTR+1
9573 60         184            RTS
9574            185  ;
9574            186  ;INTERNAL STORAGE AREAS
9574            187  ;
9574            188  VARLN     DFS $1             ;VARIABLES LENGTH
9575            189  INDEX     DFS $2             ;SEARCH INDEX
9577            190  UPLIM     DFS $2             ;HIGHEST POSSIBLE POSITION FOR SEARCH
9579            191  LOWLIM    DFS $2             ;LOWEST POSSIBLE POSITION FOR SEARCH
957B            192  SAVARRAY  DFS $2             ;WORK AREA
957D            193  ARRAYLN   DFS $2             ;LENGTH OF CURRENT ARRAY ELEMENT
957F            194  ARRAYAD   DFS $2             ;ADDRESS OF CURRENT ARRAY ELEMENT
9581            195  ;
9581            196            END
```

in SS%. To use the & feature you must BRUN the object program. The other choice is to BLOAD the program and use CALL −27632 in place of the ampersand. This will allow you to use this subroutine in conjunction with another ampersand routine.

Upon entering the subroutine at ENTRY the TXTPTR (see July Apple Slices for an explanation of TXTPTR, FIND, CHRGET, DATA, and VARNAM) is advanced to point at the first character past the GET token. Next, a JSR to CHKOPN (an Applesoft built-in routine) is performed, which checks for an open parenthesis. The JSR to GETARYPT (Applesoft built-in routine) returns with the address of the descriptor for XX$ in LOWTR (9B$ - 9C$). If the array cannot be found an "OUT OF DATA IN LINE nnn" error message is produced.

Lines 36-40 check the number of dimensions to be sure that this is a one-dimensional array. If it is not, a syntax error message is produced (line 40). The array descriptor address is then saved for future use in SAVARRAY (lines 41 through 44). A JSR to CHKCOM ensures that a comma separates the two parameters and loads the accumulator with the first byte following the comma. This byte is stored at VARNAM. Lines 47 through 54 load VARNAM + 1 with either the negative ASCII of the second byte of the two-byte or longer variable name, or $80 if the variable name is only one byte long.

A JSR to FIND loads LOWTR with the address of the descriptor of the passed variable. Lines 56 through 64 load and save the length and address of the passed variable in VARLN and VARAD respectively. Lines 65 through 74 re-establish LOWTR to the address of the array's descriptor (SAVARRAY) and initialize the upper limit (UPLIM) to the size of the array. The lower limit (LOWLIM) is then initialized to zero, and the main search loop (SEARCHLP) is entered. First there is a JSR to COMPIDX, which is an internal routine that takes the average of the upper and lower limits and stores the result at INDEX. INDEX will be used as the current position in the array of the binary search.

Now SEARCHLP takes the current value of the INDEX field and multiplies it by three (JSR BY3), placing the result in LOWTR. This is done because each string element in the array has a three-byte entry in the array descriptor, a length byte followed by a two-byte address. To find the displacement of the individual element's entry from the base address of the array's descriptor, it is necessary to multiply INDEX by three.

LOWTR is then added to the base address of the array's descriptor (SAVARRAY); the result is stored back in LOWTR. The length of the searched element is then found and saved in ARRAYLN (lines 88 through 89). The seven-byte Y-index value is needed because the individual string array entries start seven bytes from the beginning of the array descriptor in any one-dimensional array. The X-register will be used as the number of bytes left in the array element and string variable to compare. It is initialized to the lower of the VARLN and ARRAYLN internal parameters (lines 90 through 94).

Next, the address of the array element is found and placed in LOWTR (lines 95 through 104). The compare loop (COMPLP) then compares the array element to the string variable, byte for byte, up to the length of the shortest of the two elements (using the X-register as a counter). If the string is lower in value than the array element a JMP to STRNGLO is performed (line 110). If the string is higher in value, then a JMP to STRNGHI is performed (line 108). If the two items are equal (line 109) the lengths are compared. If the string is shorter it is considered to be lower in value and a JMP to STRNGLO is performed (line 116). If the two items are of equal length then a branch to EXIT is performed, which sets up an integer variable SS% and loads it with the current value of INDEX. This value is the location of the search argument in the array. The last thing EXIT does is JMP to DATA, which is Applesoft's routine to advance the TXTPTR to the end of the current statement (lines 119 through 129).

STRNGHI first compares the lower limit of the search (LOWLIM) to the INDEX. If they are equal then the upper limit and the lower limit have converged, which means the element could not be found. Under this condition a JMP to the internal routine NOTFOUND is performed (lines 130-136). NOTFOUND loads INDEX with a −1 and JMPs to EXIT where INDEX is passed to the SS% parameter as described above.

If the upper and lower limits have not converged, STRNGHI then resets the lower limit by moving INDEX (lines 137 through 140). STRNGHI then returns to the main search loop (SEARCHLP) to continue the search.

STRNGLO works essentially like STRNGHI except it tests for convergence by checking to see if INDEX is equal to the upper limit. If it is not, STRNGLO resets the upper limit to INDEX instead of the lower limit.

Subroutine Hints

Before using BINARY-SEARCH you should set HIMEM to 37888 or lower (if you decide to load the routine at $9400). I could have set HIMEM for you in SETVEC, but I believe that leaving this task to you allows more flexibility; you can BLOAD and CALL the routine instead of using the & feature. You can also BRUN the subroutine from anywhere in your BASIC program, instead of just from the first line.

MICRO

Adding Voice to a Computer

by Michael E. Valdez

A low-cost procedure for sampling and reproducing voice with a computer including the required hardware and software.

Voice
requires:

A computer with a 4-bit port available and a Motorola 3417 speech/digital converter

Several methods are available today to add voice to a computer. The method developed by Texas Instruments uses a model of the mouth and generates the necessary parameters by linear predictive coding. This method gives excellent results producing isolated words with very high quality, but is expensive. Another problem is that it is necessary to have a read-only memory with the parameters of the words to be used; this read-only memory can be produced only by Texas Instruments. It has several ready-made, read-only memories with standard vocabularies at a very reasonable price. Using this method requires minimal knowledge of acoustics and linguistics. The user has to write some simple programs to control the unit, the worst requirement being to prevent the words from running together.

The signal compression and delta modulation method developed by National Semiconductors, although very different technically, is similar from the user's point of view to the one developed by Texas Instruments. With this method it is also necessary to use a read-only memory produced by the manufacturer, and the cost is also in the same range (around two hundred dollars). But, the results are somewhat robotic.

A continuously variable slope delta modulation developed by Motorola uses the same integrated circuit for storing and reproducing speech. This is the only method available today that permits the user to sample his own speech. The unit to be described in this article is inexpensive (fifteen dollars for parts), and the knowledge requirements of acoustics and linguistics are minimal. The user should know how to use a tape recorder and write some simple programs. The hardest requirement is the timing of the loops. The quality of reproduction is quite good and depends heavily on the quality of the tape recording equipment. The digital data can be stored in read-write or read-only memory, or it can be saved on magnetic tape or disk.

The phoneme concatenation method uses the SC01 phoneme synthesizer developed by Votrax. The results of this procedure are mechanical but it is important to recognize that this is the only real synthesis procedure for the production of speech by a computer; that is, it is not necessary to sample speech to obtain data to be reproduced by the computer as in the other methods. The voice is generated by entering numbers into the computer and the SC01, or any other device. Naturally, since this method does not reproduce speech, the generated voice does not resemble the voice of the operator, or anybody else. In its most elementary use, the voice can be described as robot-like because of the lack of intonation and inflections. With additional work and knowledge, it is possible to obtain better results. The cost of a simple unit is under one hundred dollars. The use of this method re-

***Listing 1:* Program for Adding Voice to a Computer**

```
1000:                      ORG  $1000
1000:           * MODIFY TO SUIT INSTALLATION
1000:           ******************************
1000:           *                            *
1000:           *                            *
1000:           * PROGRAM TO ADD VOICE TO ANY *
1000:           *                            *
1000:           *                            *
1000:           *          COMPUTER          *
1000:           *                            *
1000:           *                            *
1000:           ******************************
1000:                      MSB  OFF
1000:           *
1000:           * STORAGE LOCATION MUST BE MODIFIED
1000:           * TO SUIT SYSTEM
1000:           *
0010:           PNT        EQU  $10
0012:           END        EQU  $12
0014:           BITS       EQU  $14
1000:           *
1000:           * SYSTEM SUBROUTINES
1000:           *
F8B2:           KKK        EQU  $F8B2      KEYBOARD INPUT IN ASCII
FA5F:           OUT        EQU  $FA5F      OUTPUT IN ASCII
1000:           *
1000:           * LOCATIONS OF I/O PORT
1000:           *
EF80:           DELR       EQU  $EF80      6522 PORT
EF82:           DELDR      EQU  $EF82      6522 DATA DIRECTION REGISTER
1000:           *
1000:           *
1000:           * PROGRAM START
1000:           *
1000:           *
1000:A2 00      DELTA      LDX  #0         BEGINNING OF BUFFER
1002:BD 52 11   DEL1       LDA  DLM,X
1005:C9 1F                 CMP  #$1F
1007:F0 06                 BEQ  DEL4
```

(continued)

quires some knowledge of linguistics and phonetics if good results are desired, but the manufacturer provides substantial support.

Intel has developed what they call an analog microprocessor — a single-chip device to work with analog signals. This unit, the 2920, can be used for speech synthesis or reproduction, but its use is limited to those persons with a substantial knowledge of acoustics, linguistics, physics, mathematics, and a high level of programming proficiency. This unit is for the serious user. There are several other units in this category, manufactured by TRW, Harris, and others.

The Motorola 3417

The Motorola 3417 is a linear bipolar chip housed in a 16-pin dual in line package, which is compatible with both TTL and CMOS technologies. The 16-pin package makes it easy to mount since sockets are available everywhere. The chip has the circuitry for the encoder (speech to digital) and decoder (digital to speech) conversions.

Pins 1 and 7 are the speech input and output while pins 13 and 9 are the digital input and output, respectively. Data then travels in the chip from pin 1 to pin 9 or from pin 13 to pin 7 depending on the input to pin 15, encode/decode. A high in pin 15 makes the chip encode the speech input to pin 1 giving a digital output through pin 9. A low in pin 15 converts digital input through pin 13 to a speech output in pin 7.

The chip provides for positive and negative excursion of the speech signal with a regulated voltage at half of the supply voltage that is used as zero for the speech input or output. The chip also provides pin 12 to set the threshold between digital zero and one, to adjust the chip to different technologies. The feedback point of the output amplifier is accessible in pin 6 to include a filter if desired. Pins 3, 4, and 11 provide access to the integrator to permit the addition of a syllabic filter. The Motorola 3417 works with a single supply voltage and requires a 16 Khz clock input at pin 14.

The data sheet provides a full explanation of the theory of continuously variable delta modulation as well as a variety of circuit information.

Hardware

For reasons of simplicity and low cost, the unit described in this article

Listing 1 *(continued)*

```
1009:20 5F FA           JSR  OUT
100C:E8                 INX
100D:D0 F3              BNE  DEL1
100F:A9 0E       DEL4   LDA  #$E        INITIALIZE PORT
1011:8D 82 EF           STA  DELDR
1014:20 02 11           JSR  ADRS
1017:D0 01              BNE  DEL2
1019:            * PROGRAM ENDS WHEN THE INITIAL ADDRESS IS ZERO
1019:60                 RTS
101A:C9 FF       DEL2   CMP  #$FF       STANDARD FILE
101C:F0 1A              BEQ  DEL3
101E:A5 13              LDA  END+1      MOVE TO POINTER
1020:85 11              STA  PNT+1
1022:A5 12              LDA  END
1024:85 10              STA  PNT
1026:A2 00              LDX  #0         END OF BUFFER
1028:BD FA 11    DEL5   LDA  DLM3,X
102B:C9 1F              CMP  #$1F
102D:F0 06              BEQ  DEL6
102F:20 5F FA           JSR  OUT
1032:E8                 INX
1033:D0 F3              BNE  DEL5
1035:20 02 11    DEL6   JSR  ADRS
1038:A2 00       DEL3   LDX  #0         INPUT OR OUTPUT?
103A:BD E1 11    DEL7   LDA  DLM1,X
103D:C9 1F              CMP  #$1F
103F:F0 06              BEQ  DEL8
1041:20 5F FA           JSR  OUT
1044:E8                 INX
1045:D0 F3              BNE  DEL7
1047:20 B2 F8    DEL8   JSR  KKK
104A:C9 4F              CMP  #$4F       ASCII O
104C:F0 5B              BEQ  OUTPUT
104E:C9 49              CMP  #$49       ASCII I
1050:D0 E6              BNE  DEL3
1052:            *
1052:            *
1052:            * INPUT ROUTINE
1052:            *
1052:            *
1052:A2 00              LDX  #0         SIGNAL WHEN READY
1054:BD 16 12    INP0   LDA  DLM2,X
1057:C9 1F              CMP  #$1F
1059:F0 06              BEQ  INP4
105B:20 5F FA           JSR  OUT
105E:E8                 INX
105F:D0 F3              BNE  INP0
1061:20 B2 F8    INP4   JSR  KKK
1064:A9 0C              LDA  #$C        START CLOCK
1066:8D 80 EF           STA  DELR
1069:A0 00              LDY  #0
106B:A2 08       INPUT  LDX  #8         EIGTH BITS
106D:A9 04       INP1   LDA  #4         CLOCK LOW
106F:8D 80 EF           STA  DELR
1072:EA                 NOP             DUMMY
1073:EA                 NOP             DUMMY
1074:AD 80 EF           LDA  DELR       GET NEXT BIT
1077:4A                 LSR  A          MOVE TO CARRY FLAG
1078:26 14              ROL  BITS       ASSEMBLE WORD
107A:A9 0C              LDA  #$C        CLOCK HIGH
107C:8D 80 EF           STA  DELR
107F:CA                 DEX             COUNT BITS
1080:D0 18              BNE  INP3       CYCLE EIGHT TIMES
1082:A5 14              LDA  BITS       RECOVER WORD
1084:91 10              STA  (PNT),Y    SAVE IN BUFFER
1086:E6 10              INC  PNT        INCREMENT POINTER
1088:D0 02              BNE  INP2
108A:E6 11              INC  PNT+1
108C:38          INP2   SEC             TEST FOR BUFFER FULL
108D:A5 12              LDA  END
108F:E5 10              SBC  PNT
1091:A5 13              LDA  END+1
1093:E5 11              SBC  PNT+1
1095:B0 D4              BCS  INPUT      GO BACK FOR MORE
1097:4C 00 10           JMP  DELTA      END
109A:A1 14       INP3   LDA  (BITS,X)   DUMMY
109C:A1 14              LDA  (BITS,X)   DUMMY
109E:A1 14              LDA  (BITS,X)   DUMMY
10A0:A1 14              LDA  (BITS,X)   DUMMY
10A2:B5 14              LDA  BITS,X     DUMMY
10A4:B5 14              LDA  BITS,X     DUMMY
10A6:4C 6D 10           JMP  INP1       CONTINUE
10A9:            *
10A9:            *
10A9:            * OUTPUT ROUTINE
10A9:            *
10A9:            *
10A9:A9 00       OUTPUT LDA  #0         CLOCK LOW
10AB:8D 80 EF           STA  DELR
```

Listing 1 *(continued)*

```
10AE:A2 00            LDX  #0         SIGNAL WHEN READY
10B0:BD 16 12   OUT4  LDA  DLM2,X
10B3:C9 1F            CMP  #$1F
10B5:F0 06            BEQ  OUT5
10B7:20 5F FA         JSR  OUT
10BA:E8               INX
10BB:D0 F3            BNE  OUT4
10BD:20 B2 F8   OUT5  JSR  KKK
10C0:A0 00            LDY  #0
10C2:B1 10      OUT0  LDA  (PNT),Y    GET NEXT WORD
10C4:85 14            STA  BITS       SAVE IT IN BITS
10C6:E6 10            INC  PNT        INCREMENT POINTER
10C8:D0 02            BNE  OUT1
10CA:E6 11            INC  PNT+1
10CC:A2 08      OUT1  LDX  #8         SEND EIGHT BITS
10CE:A9 08      OUT2  LDA  #8         CLOCK HIGH
10D0:8D 80 EF         STA  DELR
10D3:A9 02            LDA  #2         PREPARE ACCUMULATOR
10D5:06 14            ASL  BITS       GET BIT
10D7:2A               ROL  A          INTO ACCUMULATOR
10D8:2A               ROL  A          SHIFT ONE MORE
10D9:8D 80 EF         STA  DELR       SEND TO 3417
10DC:29 02            AND  #2         CLEAR CLOCK
10DE:8D 80 EF         STA  DELR       CLOCK LOW
10E1:CA               DEX             EIGHT BITS?
10E2:D0 0E            BNE  OUT3       GO FOR MORE
10E4:38               SEC             TEST FOR BUFFER FULL
10E5:A5 12            LDA  END
10E7:E5 10            SBC  PNT
10E9:A5 13            LDA  END+1
10EB:E5 11            SBC  PNT+1
10ED:B0 D3            BCS  OUT0       GO FOR MORE
10EF:4C 00 10         JMP  DELTA
10F2:A1 14      OUT3  LDA  (BITS,X)   DUMMY
10F4:A1 14            LDA  (BITS,X)   DUMMY
10F6:A1 14            LDA  (BITS,X)   DUMMY
10F8:B1 14            LDA  (BITS),Y   DUMMY
10FA:B5 14            LDA  BITS,X     DUMMY
10FC:B5 14            LDA  BITS,X     DUMMY
10FE:EA               NOP             DUMMY
10FF:4C CE 10         JMP  OUT2       CONTINUE
1102:                 *
1102:                 *
1102:                 * GET ADDRESS SUBROUTINE
1102:                 *
1102:                 *
1102:A9 00      ADRS  LDA  #0
1104:85 12            STA  END
1106:85 13            STA  END+1
1108:20 B2 F8   ADR0  JSR  KKK        GET CHARACTER
110B:20 5F FA         JSR  OUT        DISPLAY IT
110E:C9 53            CMP  #$53       CHECK IF S
1110:D0 11            BNE  ADR1
1112:A9 00            LDA  #0         STANDARD BUFFER
1114:85 10            STA  PNT
1116:84 12            STY  END        CHANGE VALUES
1118:A9 04            LDA  #4
111A:85 11            STA  PNT+1      PER INSTALLATION
111C:A9 40            LDA  #$40
111E:85 13            STA  END+1
1120:A9 FF            LDA  #$FF
1122:60               RTS
1123:C9 0D      ADR1  CMP  #$D        CHECK FOR CAR RET
1125:F0 26            BEQ  ADR3
1127:C9 30            CMP  #$30       TEST IF NUMBER
1129:90 DD            BCC  ADR0       IGNORE IF NOT
112B:C9 3A            CMP  #$3A
112D:90 0C            BCC  PKA
112F:C9 41            CMP  #$41       TEST IF HEXA LETTER
1131:90 D5            BCC  ADR0       IGNORE IF NOT
1133:29 5F            AND  #$5F       CONVERT TO UPPER CASE
1135:C9 47            CMP  #$47
1137:B0 CF            BCS  ADR0
1139:69 09            ADC  #9
113B:29 0F      PKA   AND  #$F
113D:0A               ASL  A          ROL INTO END, END+1
113E:0A               ASL  A
113F:0A               ASL  A
1140:0A               ASL  A
1141:A2 04            LDX  #4
1143:0A         ADR2  ASL  A
1144:26 12            ROL  END
1146:26 13            ROL  END+1
1148:CA               DEX
1149:D0 F8            BNE  ADR2
114B:F0 BB            BEQ  ADR0
114D:A5 12      ADR3  LDA  END        GET IF ZERO
114F:05 13            ORA  END+1
1151:60               RTS
```

(continued)

uses the Motorola MC3417 continuously variable delta modulator/demodulator. The Harris HC55516 could also be used but the circuit must be redesigned to account for the fact that the 55516 is a CMOS chip. If the computer to be used has an available port with four free bits, very few additional components are needed. Furthermore, none of the components shown on the circuit is critical and the values can vary before the quality of the results is degraded. Normally, the noise and the quality of the tape recording equipment will be the limiting factors for the quality of the reproduction. The circuit shows part of a 6522 Versatile Interface Adapter controlling the 3417, but the job can be done with any other programmable parallel port, or with three flip-flops and one tri-state unit. If the program presented with this article is to be used, the location of each signal in the word must be respected. Bit zero is the digital output from the chip, bit one is the digital input to the chip, bit two is the encode/decode control, and bit three is the clock. Bit zero must be programmed as input and the other three as outputs.

One interesting point to mention in this circuit is the lack of a clock. The 3417 requires a 16 Khz clock; in this circuit the clock is produced in software thereby avoiding the problems of synchronization. If an independent clock is used, it is necessary to sample it to send and recover the bits at the proper time.

The audio amplifier shown on the circuit is very simple and includes an elementary filter to reduce the digitizing noise. Notice the capacitor in parallel with the speaker for the same reason. Some experimentation with the values used in a particular circuit might improve the quality of reproduction. The circuit can be built in the existing board of the computer, if there is room, or wire wrapped in a small board and connected as convenient. Only five volts are required to power the unit.

Software

The software presented with this article is self explanatory. The user must adjust the memory locations to match his system. The subroutine KKK reads the keyboard and returns with the ASCII character in the accumulator; the subroutine OUT displays the accumulator.

The only part of the program that

should be treated carefully is the generation of the clock. It is important to maintain the sampling and reproduction clocks as close as possible. Large variations produce unpleasant results.

The program presented here has been written for the 6502. Converting the code to any other microprocessor requires only limited programming ability.

The Use of the Unit

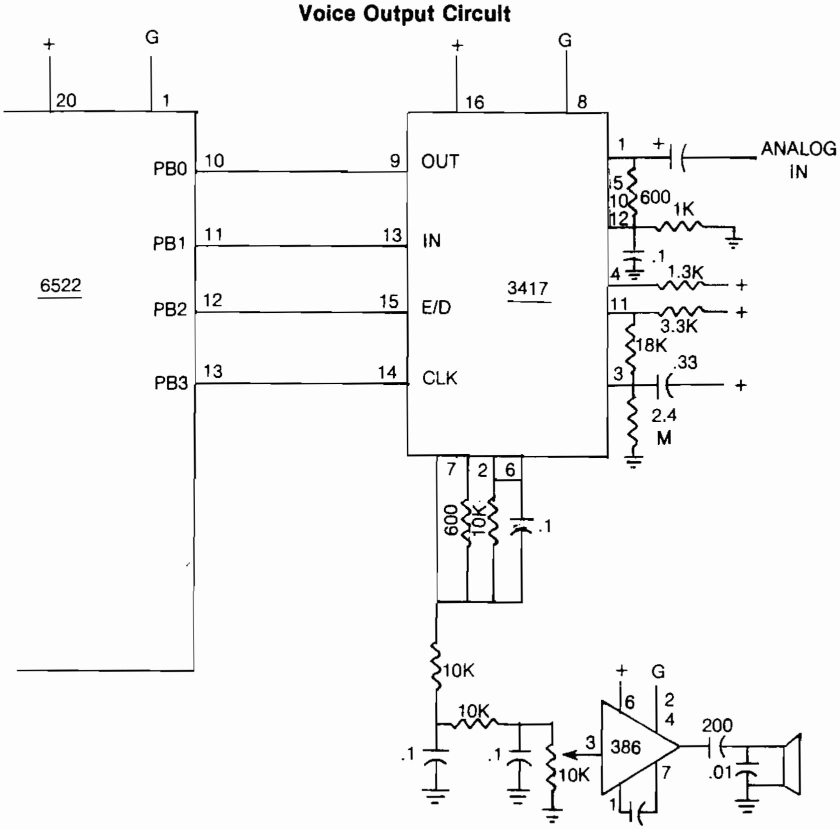
The unit is very simple to use. A cassette or any tape recorder records the words of messages to be stored for later reproduction. It is good to leave pauses before and after each part to aid in recognition. When an acceptable record has been obtained, especially without too much background noise, the output of the tape recorder is connected to the input of the unit, and the program is run.

Some practice is required to start the tape recorder and to signal the computer such that the whole record is sampled; this is especially true when the record is long and the buffer is small. Recall that 2K of memory is needed for each second of speech. The program permits finding the initial and final location of memory used by the

Listing 1 *(continued)*

```
1152:             *
1152:53 50 45     DLM      ASC   "SPEECH   ANALYSIS AND SYNTHESIS USING"
1155:45 43 48
1158:20 41 4E
115B:41 4C 59
115E:53 49 53
1161:20 41 4E
1164:44 20 53
1167:59 4E 54
116A:48 45 53
116D:49 53 20
1170:55 53 49
1173:4E 47
1175:0D                    DFB   13
1176:43 4F 4E              ASC   "CONTINUOUSLY VARIABLE SLOPE DELTA
                                  MODULATION"
1179:54 49 4E
117C:55 4F 55
117F:53 4C 59
1182:20 56 41
1185:52 49 41
1188:42 4C 45
118B:20 53 4C
118E:4F 50 45
1191:20 44 45
1194:4C 54 41
1197:20 4D 4F
119A:44 55 4C
119D:41 54 49
11A0:4F 4E
11A2:0D                    DFB   13
11A3:57 49 54              ASC   "WITH      THE MOTOROLA MC3417 IC."
11A6:48 20 54
11A9:48 45 20
11AC:4D 4F 54
11AF:4F 52 4F
11B2:4C 41 20
11B5:4D 43 33
11B8:34 31 37
11BB:20 49 43
11BE:2E
11BF:0D 0D                 DFB   13,13
11C1:50 4C 45     DLM0     ASC   "PLEASE,  ENTER BEGINING ADDRESS"
11C4:41 53 45
11C7:2C 20 45
11CA:4E 54 45
11CD:52 20 42
11D0:45 47 49
11D3:4E 49 4E
11D6:47 20 41
11D9:44 44 52
11DC:45 53 53
11DF:0D 1F                 DFB   13,$1F
11E1:             *
11E1:0D           DLM1     DFB   13
11E2:49 53 20              ASC   "IS       IT INPUT OR OUTPUT?"
11E5:49 54 20
11E8:49 4E 50
11EB:55 54 20
11EE:4F 52 20
11F1:4F 55 54
11F4:50 55 54
11F7:3F
11F8:0D 1F                 DFB   13,$1F
11FA:             *
11FA:50 4C 45     DLM3     ASC   "PLEASE,  ENTER LAST ADDRESS"
11FD:41 53 45
1200:2C 20 45
1203:4E 54 45
1206:52 20 4C
1209:41 53 54
120C:20 41 44
120F:44 52 45
1212:53 53
1214:0D 1F                 DFB   13,$1F
1216:             *
1216:50 4C 45     DLM2     ASC   "PLEASE,  SIGNAL WHEN READY"
1219:41 53 45
121C:2C 20 53
121F:49 47 4E
1222:41 4C 20
1225:57 48 45
1228:4E 20 52
122B:45 41 44
122E:59
122F:0D 1F                 DFB   13,$1F
1231:             *
*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Voice Output Circuit



sample, by changing the initial and final locations of the part to be reproduced.

If the message has pauses, it is possible to save memory by converting the reproduction program into a subroutine, making a call for each one of the parts, with appropriate waiting loops separating them. If it is better to leave the pauses in, clear the tape noise by storing hexadecimal 55 in all the locations of the pause. Now it is possible to see how little noise the process itself introduces!

When the message is to be stored in permanent memory and used many times, it is advisable to use a good high-speed tape recorder and a person with a pleasant voice to produce the originals. With several messages stored on disk it is possible to write a routine that calls the proper message into a standard area of memory and reproduces it. In this way, the same routine can handle many messages in an economical way.

You may contact Mr. Valdez at 1001 Flotilla, Indian Harbour Beach, FL 32937.

MICRO

Enhanced Video for OSI CIP

by David Cantrell and Terry Terrance

Add a screen blanker, inverse upper case, and dim character set to your Challenger.

Enhanced Video
requires:
OSI C1P
hardware modification

By adding five chips and cutting only two traces, you can add several features to your C1P video section. There will be a trade-off for these features, however. To keep the hardware and software as simple as possible, you lose lower-case alphanumerics when these features are implemented. But, no software support is necessary; no cumbersome POKEing and no software drivers to scroll a background screen (because there isn't any). You simply release your SHIFT-LOCK key whenever you want to enter modified video. Your machine's video will interpret lower-case characters as modified video whenever this modification is enabled. Since the rest of your machine simply "sees" lower-case alphanumerics, they can be put into strings and then simply PRINTed to the screen. The video modification can be disabled with either a hardware or software switch.

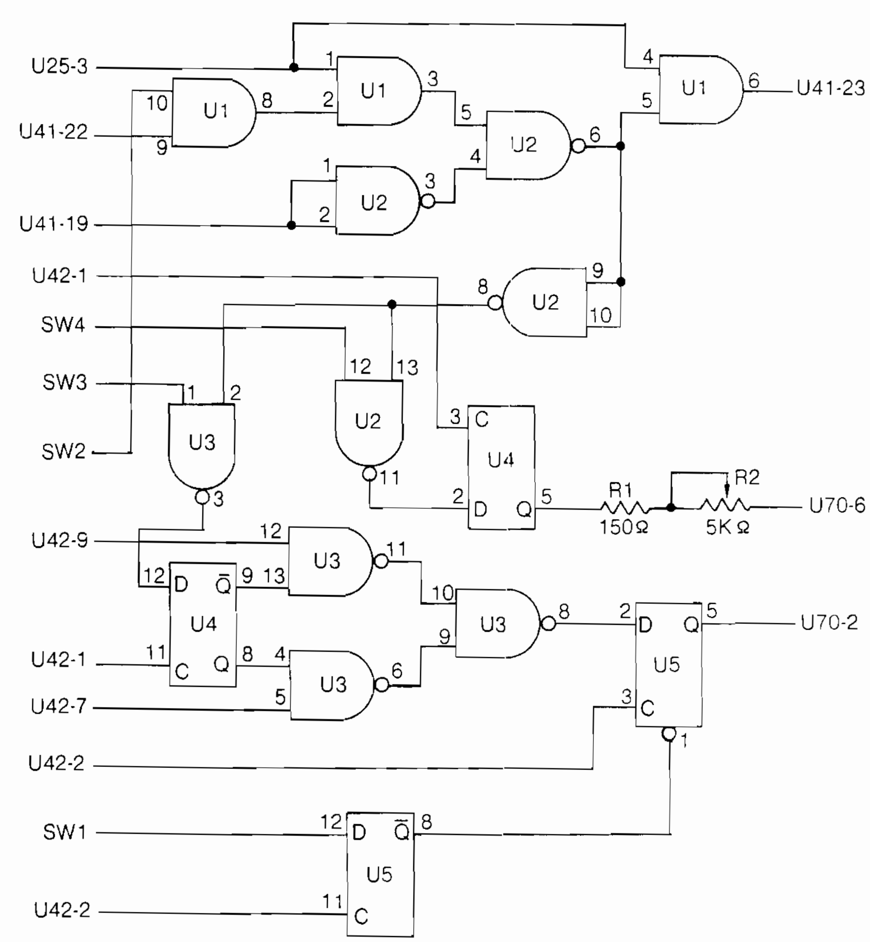
The circuit keys on Video Data Bit 5 (VD5) and Video Data Bit 6 (VD6). Whenever these bits are high and the modification is enabled, VD5 and VD6 will be masked, turning lower case into upper case, and an upper-case character in the selected "mode" (i.e., inverse, dim, etc.) will be displayed instead of the lower-case character. Since characters above 128 also have VD5 and/or VD6 set, gating is used to restore VD5 and VD6 and disable the modification whenever VD7 is set, retaining your graphics characters.

Before we get into soldering, let's discuss OSI's video as implemented on the C1P. Even though we've spent the past couple of years squinting at our C1P's screen almost daily, some of its subtleties have escaped us. When the screen is filled with CHR$(161) (OSI's solid white block character) and is viewed from about two feet away, all but the poorest TV or video monitor will show faint dark vertical lines on character cell boundaries. You may have attributed these lines to a one-dot-wide intercell space.

Closer inspection reveals that the whole screen is filled with evenly spaced dots — no blank spaces appear between cells. As the rows of dots of each character are clocked out of the shift register U42, the first dot in each row is held only one-third as long as the others in that row. Since this happens for the first dot of each row and for each character, the end result is faint dark bars when viewed from a distance.

This is the subtle video defect alluded to before. It's so subtle that most OSIers do not notice it, or pass it off as intercell spacing. If C4 users are wondering why this effect can't be seen, the effect is reversed on the C4. The first

***Figure 1:* Schematic for Enhanced Video**



dot is accentuated giving rise to bright vertical lines. This minor problem wouldn't be worth mentioning except the timing defect that causes it must be fixed if we are to add our modified video.

Before you begin construction, here are a few warnings. Keep all wires as short and as direct as possible. You'll be dealing with your video signal at RF frequencies. You'll want to avoid reradiating your game of invaders all over your house and quite possibly to the neighbors' too. Do not substitute 74LSXX series components for 74XX series components or *vice versa*. This circuit is carefully balanced regarding timing and current drive capabilities; tampering will probably overheat all of the components in the circuit.

The parts list is short; you will need

U1	74LS08	Quad 2-Input And Gates
U2, U3	74LS00	Quad 2-Input Nand Gates
U4, U5	7474	Dual D Flip-Flop
R1		150 Ohm resistor
R2		5K Ohm potentiometer
SW1-SW4		SPST switch

Since there are five chips in the circuit, it cannot be assembled in the proto area of your C1P. You can assemble the circuit on perfboard or solderless breadboard using wire-wrap (or any technique you prefer). The circuit assembles in a straightforward manner. In figure 1 the chips numbered U1-U5 refer to the components of our modification; all other "U" numbers refer to chips on your C1P.

The schematic does not show how to wire in SW1-SW4. SW1-SW4 are the mode slection switches; each one should connect its associated line to ground. We have not found it necessary, but good circuit design would dictate that the lines SW1-SW4 should be pulled up to +5 by 3.3K pull-up resistors. Figure 1 does not show supplying +5V and ground to all of the chips in the circuit. All the chips used have the standard DIP power and ground pins. For 14-pin packages, all pins 7 should be wired to ground and all pins 14 should be supplied with +5V.

Once the circuit is assembled, you must splice it onto your C1P. Cut the trace running from U41 pin 23 to U40 pin 13, and the trace running from U42 pin 9 to U70 pin 2. Connect U25 pin 3 to U1 pin 1. Connect U41 pin 22 to U1 pin 9 and U41 pin 19 to U2 pin 2. Connect U1 pin 6 to U41 pin 23.

We'll stop for a moment and explain what this part of the circuit does. U25 pin 3 is VD5 and U41 pin 22 is VD6, the data bits that the circuit keys on to know whether to output modified video. U41 pin 19 is VD7. Three gates of U1 and two gates of U2 perform logic to accomplish the following functions. If VD5 and VD6 are high and SW2 is high and VD7 is low, U1 pin 6 is low causing lower-case characters to be read as upper case and activating the rest of the circuit *via* U2 pins 9 and 10. If either VD6 or VD5 is low or SW2 is low, U1 pin 6 will be high and the screen will behave normally.

Continuing with conections, U42 pin 9 is brought into U3 pin 12. U42 pin 1 is brought into U4 pin 11; U42 pin 7 is brought into U3 pin 5. Connect U42 pin 2 to U5 pin 3 and connect U42 pin 2 to U5 pin 8. Signals coming out of the circuit on U5 pin 5 must be connected to U70 pin 2. The output of the potentiometer R2 should be brought to U70 pin 6.

This is where our circuit starts modifying video. If the first part of the circuit has recognized a modified video situation (i.e., VD5 VD6 VD7 SW2), then U2 pin 8 goes high. The signal is now fed to parts of U2 and U3 where, combined with the states of switches SW3 and SW4, the inverse and dim options are selected. If dim is selected, either alone or in combination with inverse, the signal on U2 pin 11 is used to enable the flip-flop U4, which is clocked at the shift-load rate (i.e., CLK/8) and through the R1-R2 network modulates the video for a dimming effect. R2 controls the level of brightness from almost fully bright to almost dark. SW3 controls the inverse option. If it is low, the normal video signal is passed from U42 pin 9 out to U5 pin 5 without inversion (but with latching as we will see in a moment). When SW3 is high, the shift-load clock (from U42 pin 1) and the inverse shift register output are combined by sections of U4 and U3 to produce inverse video. The section of U5 that immediately follows fixes the video defect we mentioned earlier. Instead of the dots being cut off by the video chain clock, it is now latched for the whole period of the system clock and, therefore, maintains full brightness. This part of the circuit operates regardless of whether any modified video options are selected.

We haven't forgotten SW1 and the other half of U5. They combine, along with your system's clock, to produce the blank screen option mentioned earlier. When SW1 is high, your screen will not show any display. Video memory will still be updated, however, so that whenever SW1 is brought low the whole screen will be restored. This could be handy to do screen set-ups, hide your game moves in a two-player game, etc.

Table 1 offers a recap on the operation of switches SW1-SW4.

Table 1

SWITCH # 1 2 3 4	MODE
H X X X	BLANK SCREEN
L L X X	NORMAL SCREEN
L H L L	UPPER CASE ONLY
L H H L	INVERSE UPPER CASE
L H L H	DIM UPPER CASE
L H H H	DIM INVERSE UPPER CASE

H = High, L = Low, X = Don't care

To test the modification, be sure all of the mode selection switches (SW1-SW4) are in the low state; this will ensure that you will have a normal screen to look at while you're setting up. We'll write a little program to fill the screen with mixed upper- and lower-case characters like the one below:

```
10 FORX=1TO12
20 PRINT"AaBbCcDdEeFfGgHhIiJj"
30 NEXT
```

This should fill your screen with alternating upper- and lower-case letters.

Using the mode selection switches, select inverse upper case; according to table 1 this should be L H H L. With the switches thus set, all lower-case letters should now be displayed as inverse upper case. Step through all the other modes to ascertain that they are working properly. If not, carefully check your wiring of both the circuit board and its interconnections to your C1P.

You may contact the authors at Orion Software Assocs., 147 Main St., P.O. Box 310, Ossining, NY 10562.

Home Control Interface for C1P

by John Krout

A circuit is presented that uses the C1P's ACIA to control an ultrasonic transducer. The transducer generates signals that control the receiver modules.

BSR X-10 DRIVER
requires:
OSI C1P
BSR X-10
hardware modifications

Perhaps the greatest untapped potential of personal computers is control of common household devices such as lamps, air conditioners, and TV sets. A computer that turns an air conditioner off after you leave for work and on before you return will rapidly pay for itself in energy savings; and one that handles lights and entertainment equipment on a schedule will discourage burglars who prefer to enter unoccupied homes. You can probably think of more uses.

BSR markets the X-10 Control System through the mail and in Sears and Radio Shack stores. This remarkable system consists of a central command console about the size of a 3" × 5" file box, and up to 16 control modules, each the size of a pack of cigarettes. An appliance is plugged into a control module, which in turn is plugged into a power outlet. A control dial on each control module allows the user to set a unique unit code, ranging from 1 to 16, for that module. The user may control the module remotely *via* the console by pushing a button to specify the unit code. Another button turns the selected control module on or off.

A second form of control module includes a dimming control for lamps, and a third form replaces a wall switch. Each control module is a radio receiver, which accepts transmitted commands only after receiving its own unit code. The command console is the transmitter, utilizing home power lines as an antenna.

Ohio Scientific was probably the first computer manufacturer to recognize the value of interfacing the X-10 command console to a personal computer. OSI now offers a hardware interface and a disk operating system to support the X-10. However, OSI charges a premium price for these items, and offers nothing to those using BASIC-in-ROM.

An optional feature of the command console provides the key to a simple and inexpensive interface to a computer. BSR also developed an ultrasonic hand-held command unit and combined the console with an ultrasonic receiver. This allows wireless control at a distance (like the ultrasonic hand-held TV controller). If you know the ultrasonic code used by BSR, a few hardware modifications in your C1P will allow computer generation of the same codes, through an ultrasonic transducer, to transmit to the command console.

Figure 1 shows the various components of a single word of BSR code. The code is binary, with each bit represented by an 8-ms pattern of sound. A bit with value 1 is sent as 4 ms of tone followed by 4 ms of silence. A bit with value 0 is sent as 1.2 ms of tone followed by 6.8 ms of silence. The data word begins with a 1 bit, followed by five bits of data, followed by five inverted bits of the same data, and completed with 16 ms of tone and 24 ms of silence. The tone itself is 40 KHz. The five-bit code for each control module and function is shown in table 1.

A single latched output bit in the computer is all you need to transmit the code. The C1P uses latched output bits to scan the keyboard and joysticks as well as drive a digital-to-analog converter (D/A) circuit. However, BASIC

Figure 1

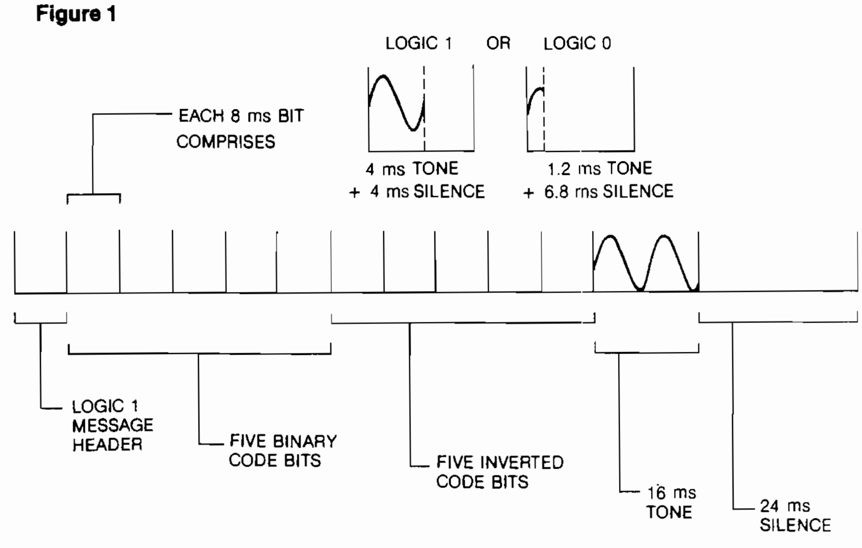


Figure 2

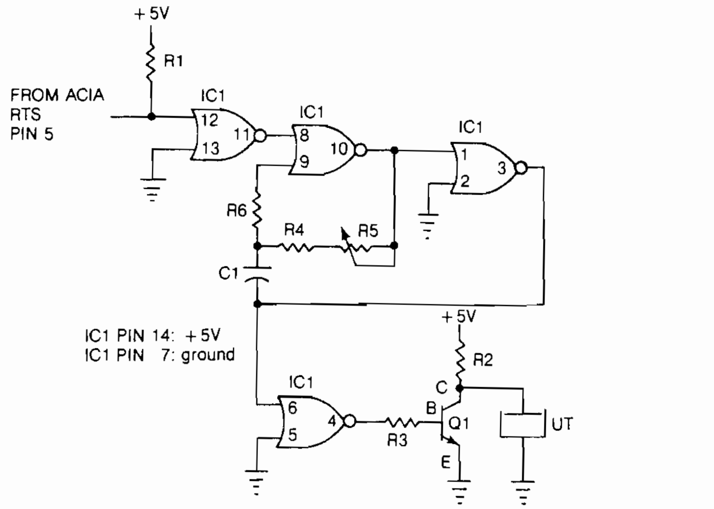


Table 2

Item	Value
IC1	4001 CMOS quad NOR gate 14-pin DIP
R1	2.2K resistor
R2	2.2K resistor
R3	2.2K resistor
R4	12K resistor
R5	50K trim potentiometer
R6	330K resistor
C1	330 pF capacitor
Q1	Sylvania ECG123A transistor or equivalent
UT	40 KHz ultrasonic transducer

Table 1

Unit Code	Binary Code				
1	0	1	1	0	0
2	1	1	1	0	0
3	0	0	1	0	0
4	1	0	1	0	0
5	0	0	0	1	0
6	1	0	0	1	0
7	0	1	0	1	0
8	1	1	0	1	0
9	0	1	1	1	0
10	1	1	1	1	0
11	0	0	1	1	0
12	1	0	1	1	0
13	0	0	0	0	0
14	1	0	0	0	0
15	0	1	0	0	0
16	1	1	0	0	0

Function Code	Binary Code				
17/All Units Off	0	0	0	0	1
18/All Lights On	0	0	0	1	1
19/On	0	0	1	0	1
20/Off	0	0	1	1	1
21/Dim	0	1	0	0	1
22/Bright	0	1	0	1	1

continually scans the keyboard (unless the Control-C break is disabled by an appropriate POKE) so some sort of tone is almost always being produced on the D/A output while BASIC, or any other keyboard-oriented program, is being used. This makes using the D/A unpleasant for music composition and playback.

A less well-known bit of latched output exists in the C1P. This is the RTS (Request-To-Send) line associated with the 6850 Asynchronous Serial Communications Interface chip (ACIA) used in the C1P to exchange data with a cassette machine, modem, or printer. This particular line is not used by the C1P, although the ACIA designers provide it so that a computer can indicate whether or not it is ready to receive data.

The control register of the ACIA chip controls the status of the RTS line, among other ACIA activities. In BASIC, whenever the Break key is depressed, the control register is reset to a value of 17 and RTS goes low. If you POKE a value of 64 to the register, then RTS will go high and stay there until another value is stored in the register. One advantage of this bit in the BSR interface is that it will automatically turn off when Break is depressed. The ACIA control register is located in the C1P at address 61440 ($F000).

The RTS line can be toggled at a 40-KHz rate to produce the BSR code. Since the C1P uses a standard clock rate of 1 MHz, the wavelength of a 40-KHz tone is precisely 25 clock cycles. However, I found by timing my C1P with an oscilloscope that its clock is running about 4% slow. Thus, I could produce the tone using a 24-clock cycle wavelength. Instead, I chose to build a free-running 40-KHz oscillator and use the RTS line to switch the oscillator output to an ultrasonic transducer.

The oscillator circuit is shown in figure 2, and the parts are listed in table 2. The only part not universally available is the ultrasonic transducer, a capacitive loudspeaker that creates the actual tone. Since these devices are

Listing 1

```
 10 ; ASSEMBLY LISTING OF BSR X-10 DRIVER ROUTINE
 20 ;
 30 ; BY JOHN KROUT
 40 ;
 50          *=$0222
 60          DELAY=$FC91
 70 ;
 80 START    JSR $AE05 ; puts arguement in $AE,AF
 90          LDX $AF
100          LDA TABLE-1,X
110          STA $AF     ; lookup & store code word
120          LDA #5
130          STA $15
140 MASTER   JSR WORD
150          DEC $15     ; counts data words sent
160          BNE MASTER
170          RTS         ; return to Basic
180 ;
190 ;
200 WORD     JSR LOGIC1  ; send message header bit
210          LDA $AF     ; command code into accumulator
220          JSR SEND    ; send top 5 accumulator bits
230          LDA $AF     ; reload accumulator
```

(continued)

Listing 1 *(continued)*

```
240          EOR #255    ; invert accumulator bits
250          JSR SEND    ; send 5 inverted bits
260          LDA #64
270          STA $F000   ; begin 16 ms tone
280          LDX #4
290          STX $16
300 LOOP1    JSR MS4
310          DEC $16
320          BNE LOOP1
330          LDA #17
340          STA $F000   ; begin 24 ms silence
350          LDX #5
360          STX $16
370 LOOP2    JSR MS4
380          DEC $16
390          BNE LOOP2
400          JMP MS4
410 ;
420 SEND     STA $13
430          LDA #5
440          STA $14     ; counter for bits sent
450 ROLL     ROL $13     ; place bit in Carry
460          BCC ZERO    ; branch if Carry=0
470          JSR LOGIC1  ; send logic 1
480          JMP COUNT
490 ZERO     JSR LOGIC0  ; send logic 0
500 COUNT    DEC $14
510          BNE ROLL    ; branch until 5 bits sent
520          RTS
530 ;
540 LOGIC1   LDA #64
550          STA $F000   ; begin 4 ms tone
560          JSR MS4
570          LDA #17
580          STA $F000   ; begin 4 ms silence
590          JMP MS4
600 ;
610 LOGIC0   LDA #64
620          STA $F000   ; begin 1.2 ms tone
630          JSR MS1.2
640          LDA #17
650          STA $F000   ; begin 6.8 ms silence
660          JMP MS6.8
670 ;
680 MS4      LDX #15
690 LOOP3    DEX
700          BNE LOOP3
710          LDX #3
720          JMP DELAY
730 ;
740 MS1.2    LDX #228
750 LOOP4    DEX
760          BNE LOOP4
770          RTS
780 ;
790 MS6.8    LDX #52
800 LOOP5    DEX
810          BNE LOOP5
820          LDX #5
830          JMP DELAY
840 ;
850 TABLE    .BYTE 96,224,32,160,16,144,80,208
860          .BYTE 112,240,48,176,0,128,64,192
870          .BYTE 8,24,40,56,72,88
```

Listing 2

```
FC91 A0F8    LDY #$F8
FC93 88      DEY
FC94 D0FD    BNE $FC93
FC96 55FF    EOR $FF,X
FC98 CA      DEX
FC99 D0F6    BNE $FC91
FC9B 60      RTS
```

Listing 3

```
100              *=$0222
110 START        LDX #64
120              STX $F000
130              NOP
140              LDX #198
150 X1           DEX
160              BNE X1
170              STX $F000
180              LDX $3
190              LDX #198
200 X2           DEX
210              BNE X2
220              JMP START
```

pretuned to a specific frequency, be sure the one you buy is set to 40 KHz. One transducer that costs less than $10 is #J4-815 in the Calectro catalog.

The circuit can be installed on any of the unconnected prototype sockets adjacent to the ACIA, with a pair of output lines running out of the computer case to the transducer. Or the circuit can be placed externally on perfboard, with connection lines for power, ground, and RTS. Because my C1P board is crowded with add-ons, I chose the latter method. I recommend that you do not mount the transducer to the C1P case because it has to be in a fairly direct line with the receiver microphone grid on the front face of the command console for transmission to be reliable. To preserve aiming flexibility, put the transducer on a lengthy flexible signal cable. You can secure it to the command console grid, if you wish.

A USR software-driver routine for the interface appears in listing 1. This routine begins by calling the ROM BASIC subroutine at address $AE05, which deciphers the argument value within the parentheses following the USR call in BASIC text, and puts that value in locations $AE and $AF in the form of a 15-bit integer with a sign bit. Any argument value outside the range of −32768 to +32767 will cause a function call error if the $AE05 routine is called.

The USR routine assumes that the argument is a number between 1 and 22, corresponding to a BSR unit or command number. Lines 90 through 110 look up the appropriate five-bit command code in a data table and replace the original argument value with the code. Lines 120 through 160 produce five repetitions of code transmission, a factor which was found reliable when used in a BASIC program that turned house lights on and off over a two-hour period. This means that each USR call takes about 640 ms.

The main subroutine WORD begins at line 200 with transmission of the single-bit prefix, a logic 1. Then the command code is loaded and transmitted once, reloaded, inverted in line 240, and transmitted again. The codeword suffix is sent by the remainder of WORD.

Subroutine SEND analyzes each bit of the five-bit command code and transmits the appropriate tone sequence. In line 450, ROL $13 places each command bit into the Carry bit of the 6502 status register and, in line 460, BCC branches if the Carry bit is zero.

Subroutine LOGIC1 turns on the RTS line, waits 4 ms, turns off the RTS line, and waits another 4 ms. LOGIC0 waits 1.2 ms after turning on RTS and then waits 6.8 ms after turning off RTS.

The three timing subroutines MS4, MS1.2, and MS6.8 handle the precise waiting periods required by the other subroutines. Each includes a DEX/BNE loop that takes five clock cycles per iteration, except that only four are used when BNE does not branch. The prior LDX immediate in each case takes two cycles, as does the following LDX immediate in MS4 and MS6.8. These two routines then use three cycles to JMP to a routine called DELAY in the monitor ROM at $FC91.

Delay is a time-delay loop that, perhaps, was included in ROM to aid in disk I/O. It appears in listing 2 and uses 1250 cycles per iteration, with the number of repetitions controlled by the 6502 X register. The RTS at the end takes an extra six cycles. The difficulty with DELAY is that it wipes out not only the X and Y registers but also the

Listing 4

```
10 PRINT"Enter your C1P clock"
15 PRINT"rate as a decimal frac-"
20 PRINT"tion of the standard 1"
25 PRINT"megahertz clock rate"
30 PRINT"(example: 6% fast is"
35 PRINT"entered as 1.06)";
40 INPUT Q
45 M4=INT(4000*Q)-12
50 M1=INT(1200*Q)-7
55 M6=INT(6800*Q)-12
60 D=1250
65 D4=INT(M4/D):R4=INT((M4-D4*D)/5)
70 R1=INT(M1/5)
75 D6=INT(M6/D):R6=INT((M6-D6*D)/5)
80 POKE675,R4:POKE680,D4
85 POKE685,D1
90 POKE691,R6:POKE696,D6
```

Listing 5

```
5 X=546:Z=60000
7 SAVE
9 PRINT:PRINT
10 FORI=0TO175
20 IFI=INT(I/15)*15THENPRINT:
   PRINTZ;"DATA";:Z=Z+5:GOTO30
25 PRINT",";
30 A$=STR$(PEEK(I+X)):PRINTRIGHT
   $(A$,LEN(A$)-1);
40 NEXT
50 PRINT
60 PRINT"20 POKE11,34:POKE12,2"
70 PRINT"30 FORI=0TO175:READA:
   POKEI+546,A:NEXT"
80 PRINT"40 NEW"
90 PRINT"POKE515,0:RUN"
95 POKE517,0
```

Listing 6

```
60000 DATA32,5,174,166,175,189,187,2,133,175,169,5,133,21,32
60005 DATA56,2,198,21,208,249,96,32,130,2,165,175,32,106,2
60010 DATA165,175,73,255,32,106,2,169,64,141,0,240,162,4,134
60015 DATA22,32,162,2,198,22,208,249,169,17,141,0,240,162,5
60020 DATA134,22,32,162,2,198,22,208,249,76,162,2,133,19,169
60025 DATA5,133,20,38,19,144,6,32,130,2,76,125,2,32,146
60030 DATA2,198,20,208,239,96,169,64,141,0,240,32,162,2,169
60035 DATA17,141,0,240,76,162,2,169,64,141,0,240,32,172,2
60040 DATA169,17,141,0,240,76,178,2,162,15,202,208,253,162,3
60045 DATA76,145,252,162,228,202,208,253,96,162,52,202,208,253,162
60050 DATA5,76,145,252,96,224,32,160,16,144,80,208,112,240,48
60055 DATA176,0,128,64,192,8,24,40,56,72,88
20 POKE11,34:POKE12,2
30 FORI=0TO175:READA:POKEI+546,A:NEXT
40 NEW
POKE515,0:RUN
```

Listing 7

```
5 GOTO2000
6 :
10 REM ... LITESHOW CONTROL PROGRAM ...
12 REM ... FOR BSR X-10 INTERFACE ...
14 REM ... BY JOHN KROUT
99 :
100 REM SPOTS: 1 ON, 1 OFF
101 :
110 FORA=1TO3:B=A+1:IFA=3THENB=1
120 Y=USR(B):IFPEEK(Q)=EGOTO1000
130 Y=USR(A):IFPEEK(Q)=EGOTO1000
140 NEXT:GOTO110
199 :
200 REM SPOTS: 2 ON, 1 OFF
201 :
210 FORA=1TO3
220 Y=USR(18):IFPEEK(Q)=EGOTO1000: REM ALL SPOTS ON
230 Y=USR(A):Y=USR(20):IFPEEK(Q)=EGOTO1000: REM 1 OFF
235 FORI=1TO1000:NEXT:REM TIME DELAY
240 NEXT:GOTO210
299 :
300 REM KEYBOARD CONTROL
301 :
302 GOSUB4000:PRINT" SPOTS":PRINT:PRINT"STROBES":PRINT:PRINT"PROJECTOR
304 POKEG,89:POKEG+2,66:POKEG+4,82
310 POKE530,1:POKE57088,127:P=PEEK(57088):POKE530,0
315 IFPEEK(Q)=EGOTO1000
320 FORA=1TO7:IFS(A,1)=PGOTO335
325 NEXT:GOTO310
335 Y=USR(A):IFS(A,0)=0THENY=USR(19):S(A,0)=1:POKES(A,2),43:GOTO310
340 Y=USR(20):S(A,0)=0:POKES(A,2),32:GOTO310
399 :
400 REM STROBES: 1 ON, 1 OFF
401 :
410 FORA=4TO6:B=A+1:IFA=6THENB=4
420 Y=USR(B):Y=USR(19):IFPEEK(Q)=EGOTO1000
430 Y=USR(A):Y=USR(20):IFPEEK(Q)=EGOTO1000
440 NEXT:GOTO410
499 :
1000 REM MAIN MENU
1020 FORI=1TO7:S(I,0)=0:NEXT:REM STATUS RESET
1025 GOSUB4000
1030 PRINT"MAIN MENU:":PRINT
1040 PRINT"1. SPOTS: 1 ON, 1 OFF":PRINT:PRINT
1042 PRINT"2. SPOTS: 2 ON, 1 OFF":PRINT:PRINT
1044 PRINT"3. KEYBOARD CONTROL":PRINT:PRINT
1046 PRINT"4. STROBES: 1 ON, 1 OFF":PRINT:PRINT
1100 INPUT"function number";F:PRINT
1110 IFF<1ORF>10ORF>INT(F)GOTO1100
1115 Y=USR(17): REM SHUTDOWN
1120 ONFGOTO100,200,300,400
1200 END
2000 REM INIT
2010 DIMS(7,2)
2020 S(1,1)=127
2030 S(2,1)=191
2040 S(3,1)=223
2050 S(4,1)=239
2060 S(5,1)=247
2070 S(6,1)=251
2080 S(7,1)=253
2100 Q=57100:E=222
2110 G=53901
2120 S(1,2)=G+64
2130 S(2,2)=G+66
2140 S(3,2)=G+68
2150 S(4,2)=G+128
2160 S(5,2)=G+130
2170 S(6,2)=G+132
2180 S(7,2)=G+194
2999 GOTO1000
4000 REM SCREEN CLR SUB
4010 FORI=1TO28:PRINT:NEXT:RETURN
```

accumulator. The latter could have been avoided by using a few NOPs instead of the EOR. In the USR routine, whenever a delay routine is called, this problem forces storage in memory of the command word, the number of words sent, and the number of bits sent. Since BASIC does not use the input buffer beginning at $13 for anything other than input, USR can access that space with compact and speedy page zero addressing for data storage on a non-permanent basis. Alternatives include stack storage and replacing DELAY with your own non-destructive time delay.

Because my C1P runs about 4% slow, the time delays in MS4, MS6.8, MS1.2, and the message suffix portion of WORD have been shortened about 4% to compensate. If you can obtain an oscilloscope, listing 3 will load and execute a useful infinite loop USR routine. This routine turns on RTS for precisely 999 cycles, and then turns off RTS for 1001 cycles, giving an overall wavelength of exactly 2 ms for a machine running at exactly 1 MHz. If your machine is running a few percent slow or fast, listing 4 will compute and POKE the necessary loop constant alterations to the BSR X-10 driver routine.

As with many USR routines, it is convenient to place the driver in unused memory below BASIC text, starting at $0222. Because the OSI Assembler occupies this space and cannot directly assemble the routine there, a loader in BASIC is useful. Listing 6 uses the familiar method of POKEing numbers from DATA statements to memory, and is itself a product of listing 5, a BASIC program generator. Listing 5 includes the very advantageous features of placing two immediate-mode commands at the end of listing 6: a POKE to terminate LOAD, and RUN. Since the DATA statements are so long in this case, the NEW statement in line 40 of listing 6 erases listing 6 after its work is done, leaving behind the driver routine and the data in locations 11 and 12 that tell BASIC where the USR routine begins.

Listing 7 is a BASIC light show control program, which is loaded after listing 6 has finished. The program presumes that X-10 lamp modules 1, 2, and 3 control colored spotlights, that appliance modules 4, 5, and 6 control colored strobe lights, and that appliance module 7 controls the lamp of a slide projector. Projector lamps usually exceed 300 watts. You should keep the projector fan running even when the lamp is off to cool the lamp and avoid a blowout.

Would you like some automation in your life? Perhaps you need a timer for your toaster, or a security system for your office copier. Computer intelligence plus BSR X-10 versatility can do it for you.

The author may be contacted at 5108 N. 23rd Rd., Arlington, VA 22207.

MICRO

ATARI Meets the BSR X-10

by David A. Hayes

A circuit is presented to interface the ultrasonic version of the BSR X-10 home control system to Atari computers. Programming information and a sample program are included.

Demo Program
requires:

Atari 400/800
BSR X-10

To use the BSR X-10 home control device, many computers require a hardware modification. David Staehlin presented a circuit, in the January 1982 issue of *BYTE* magazine, which will couple a non-ultrasonic BSR X-10 to an RS-232 port. I have interfaced the Atari's controller jack port to the more common ultrasonic version of the BSR X-10. Figure 1 shows the complete interface circuit required for this purpose. Modification of the BSR X-10 is not trivial and should be performed by competent technicians only.

The program in listing 1 loads a machine-language program into page 6 of memory. Line 100 sets up controller jack 1, pin 1, as output. Table 1 lists the code that the BSR X-10 understands. The machine-language program sends this code out controller jack 1, pin 1, whenever it is called by the USR routine.

For example, if you have made the appropriate hardware modifications, have typed in the program in listing 1, and now want to turn all lights on, line 110 of your program should look like this:

```
110 X = USR(1536,0,0,0,128,128,
        128,128,128,0,0)
```

Now turn on channel five.

```
120 X = USR(1536,0,0,0,128,0,128,
        128,128,0,128):REM SELECT
        CHANNEL 5
130 X = USR(1536,0,0,128,0,128,128,
        128,0,128,0):REM TURN ON
```

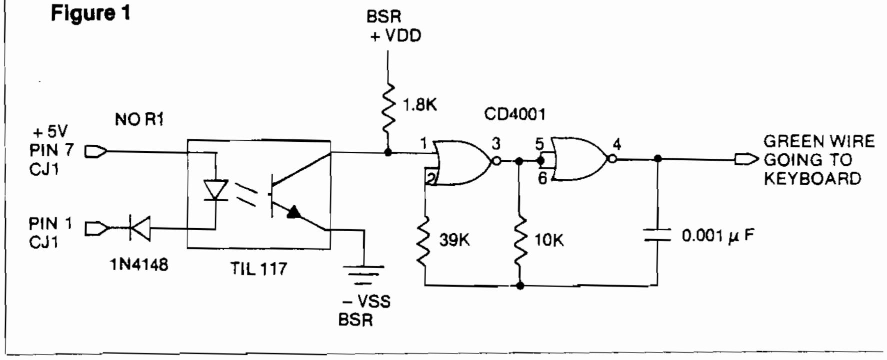
The author may be contacted at 2004 Woody Drive, Kingston, TN 37763.

(Continued on next page)

Table 1

FUNCTION	X = USR(1536,A,B,C,D,E,F,G,H,I,J)
ALL LIGHTS ON	0,0,0,128,128,128,128,128,0,0
ALL OFF	0,0,0,0,128,128,128,128,128,0
ON	0,0,128,0,128,128,128,0,128,0
OFF	0,0,128,128,128,128,128,0,0,0
BRIGHTEN	0,128,0,128,128,128,0,128,0,0
DIM	0,128,0,0,128,128,0,128,128,0
CHANNEL	
1	0,128,128,0,0,128,0,0,128,128
2	128,128,128,0,0,0,0,0,128,128
3	0,0,128,0,0,128,128,0,128,128
4	128,0,128,0,0,0,128,0,128,128
5	0,0,0,128,0,128,128,128,0,128
6	128,0,0,128,0,0,128,128,0,128
7	0,128,0,128,0,128,0,128,0,128
8	128,128,0,128,0,0,0,128,0,128
9	0,128,128,128,0,128,0,0,0,128
10	128,128,128,128,0,0,0,0,0,128
11	0,0,128,128,0,128,128,0,0,128
12	128,0,128,128,0,0,128,0,0,128
13	0,0,0,0,0,128,128,128,128,128
14	128,0,0,0,0,0,128,128,128,128
15	0,128,0,0,0,128,0,128,128,128
16	128,128,0,0,0,0,0,128,128,128

Figure 1



Listing 1

```
10 FOR ADD=1536 TO 1756: READ INST: POKE ADD,INST: NEXT ADD
20 DATA 104,32,138,6,104,104,48,6,32,169,6,76,17,6,32,138,6,
        104,104,48,6,32,169
25 DATA 6,76,30,6,32,138,6,104,104
30 DATA 48,6,32,169,6,76,43,6,32,138,6,104,104,48,6,32,169,
        6,76,56,6,32,138,6
35 DATA 104,104,48,6,32,169,6,76,69
40 DATA 6,32,138,6,104,104,48,6,32,169,6,76,82,6,32,138,6,
        104,104,48,6,32,169
45 DATA 6,76,95,6,32,138,6,104,104
50 DATA 48,6,32,169,6,76,108,6,32,138,6,104,104,48,6,32,
        169,6,76,121,6,32,138
55 DATA 6,104,104,48,6,32,169,6,76
60 DATA 134,6,32,138,6,32,200,6,96,169,254,141,0,211,162,
        120,160,10,136,208
65 DATA 253,202,208,248,169,255,141,0,211,162
70 DATA 120,160,10,136,208,253,202,208,248,96,169,254,141,
        0,211,162,40,160,10
75 DATA 136,208,253,202,208,248,169,255,141
80 DATA 0,211,162,31,160,70,136,208,253,202,208,248,96,169,
        254,141,0,211,162
85 DATA 54,160,70,136,208,253,202,208,248
90 DATA 169,255,141,0,211,96
100 POKE 54018,56: POKE 54016,1: POKE 54018,60: POKE 54016,1
```

MICRO™

68000 Logic Instructions

by Joe Hootman

This is the third in a series of articles on programming the 68000. Professor Hootman is presenting the instruction set of the 68000 microprocessor and will then consider the addressing modes and how they apply to the various instructions. This month's topic is the logical instructions.

The logic instructions implemented in the 68000 are given in table 1. These instructions are the AND, the OR, the NOT, and the EOR. The implementation of the logical operations is straightforward. The logic operations affect the CCR depending on the results of the operation. It should be noted that the logical operations do not operate on the address registers directly.

The logic operations on the status register are privileged. Logical operations on the user condition code register are not privileged.

Joe Hootman can be contacted at the University of North Dakota, Department of Electrical Engineering, University Station, Grand Forks, North Dakota 58202.

***Table 1:* Logic Instructions**

<table>
<tr><th>Mnemonic</th><th>Data Size/CCR</th><th>Name</th><th>Comments</th></tr>
<tr><td>AND</td><td>8, 16, 32
CCR
X N Z V C
- * * 0 0</td><td>Logical AND</td><td>The source and destination are logically ANDed and the result stored in the destination.
Opword Format
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11 10 9</td><td>8 7 6</td><td>5 4 3</td><td>2 1 0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>Register</td><td>Op Mode</td><td>Effective Address
Mode</td><td>
Register</td></tr>
</table>
Register — Any of the eight data registers.
Op Mode field
<table>
<tr><td></td><td>Byte</td><td>Word</td><td>Long word</td><td></td></tr>
<tr><td>A)</td><td>000</td><td>001</td><td>010</td><td>Data register ANDed with the EA and result left in the data register.</td></tr>
<tr><td>B)</td><td>100</td><td>101</td><td>110</td><td>EA ANDed with the data register and result left in the EA.</td></tr>
</table>
For case A of the Op Modes the following effective addressing modes cannot be used: 2, 13, 14.* For case B of the Op Modes the following effective addressing modes cannot be used: 1, 2, 10, 11, 12, 13, 14.*</td></tr>
<tr><td>ANDI</td><td>8, 16, 32
CCR
X N Z V C
- * * 0 0</td><td>AND Immediate</td><td>The immediate data and the destination are logically ANDed and the result stored in the destination.
Opword Format
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7 6</td><td>5 4 3</td><td>2 1 0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>Size</td><td>Effective Address
Mode</td><td>
Register</td></tr>
<tr><td colspan="8">Word data (16 bits including the first 8 bits)</td><td colspan="3">Byte data (8 bits)</td></tr>
<tr><td colspan="11">Long data (32 bits including the previous bits)</td></tr>
</table>
Size field
00 - Byte
01 - Word
10 - Long word
The following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*</td></tr>
<tr><td>ANDI to CCR</td><td>8
CCR
X N Z V C
* * * * *</td><td>AND Immediate to Condition Code Register</td><td>The immediate data is ANDed with the CCR and the results stored in the CCR. The state of the CCR after the operation depends on the previous data in the CCR and the immediate data in the operation.
Opword Format
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="8">Byte Data</td></tr>
</table></td></tr>
</table>

(continued)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Name	Comments
EOR	8, 16, 32 CCR X N Z V C - * * 0 0	Exclusive OR Logical	The source and the destination are exclusively ORed together and the result stored in the destination. (Data registers only for source data.)

Opword Format

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register	Op Mode	Effective Address Mode	Effective Address Register

Register field — Any one of the eight data registers can be specified.

Op Mode field
100 - Byte
101 - Word
110 - Long word

The effective address specifies the destination of the result of the operation and the following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
EORI	8, 16, 32 CCR X N Z V C - * * 0 0	Exclusive OR Immediate	The immediate data and the destination data is exclusively ORed together and the result stored in the destination.

Opword Format

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	1	0	1	0	Size	Effective Address Mode	Effective Address Register
Word data (16 bits)								Byte data (8 bits)		
Long data (32 bits)										

Size field
00 - Byte: The data is in the lower order byte of the immediate word.
01 - Word: The data is the entire immediate word.
10 - Long word: The data is contained in the next two immediate words.

The effective address specifies the destination of the result of the operation and the following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
EORI to CCR	8 CCR X N Z V C * * * * *	Exclusive OR Immediate to Condition Code Register	The immediate data is exclusively ORed with the CCR and the result stored in the CCR.

Opword Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data							

Mnemonic	Data Size/CCR	Name	Comments
NOT	8, 16, 32 CCR X N Z V C - * * 0 0	Logical Complement	The ones complement of the destination is taken and the results stored in the destination.

Opword Format

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	0	1	1	0	Size	Effective Address Mode	Effective Address Register

Size field
00 - Byte
01 - Word
10 - Long word

The effective address specifies the destination and the following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

(continued)

MICRObits *(continued)*

VisiCalc To Apple Plot

Interface translates from VisiCalc to Apple Plot, prevents erroneous graphs, fits curves to data, and supplements VisiCalc with rank ordering and alphabetizing. Send SASE for details or $30.00 for the copyable program.

Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

VisiCalc To Apple Writer

Veecee-Writer translated VisiCalc (/PF) files for Apple Writer 1. Send $15.00 for the copyable program.

Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

TRS-80 Color Computer

Expand your 4K system to 16K for $29.95. Expand 4K or 16K to a 32K system for only $99. Obtain better color graphics. Full instruction/documentation provided in each kit. Two- to three-week delivery time. $3 postage/handling charge.

Dick Williams
Computer Shed
Lane 2-1
Derry, NH 03038
(603) 432-3634

Unique VIC-20 User Group

Borrow any program from our extensive loan library for only 10% of cost and get free newsletter and special purchase prices on all VIC-20 hardware and software from our huge catalog. Membership only $25 by check, VISA, MasterCard.

Software To Go
Rt. 3, Box 309 A 52
Clinton, TN 37716
(615) 457-5068
(615) 584-0022

68000 Software

For Apple-compatible boards (DTACK). *The Moose:* professional and only available 68000 chess program — $67. *MUXA68:* UCSD 68000 Crossassembler — $70. *68TICID:* Debugger — $47. *PCON68:* UCSD-Interface for DTACK board — $30. $10 shipping and handling.

Moose Systems
Steenbargkoppel 21
D-2000 Hamburg 65
Germany

(Continued)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Function	Comments
OR	8, 16, 32 CCR X N Z V C - * * 0 0	Inclusive OR Logical	The inclusive OR operation performs the OR operation on the source data and the destination data. The result is left in the destination.

Opword Format

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	0	Register	Op Mode	Effective Address Mode	Effective Address Register

Register field specifies any of the 8 data registers.

Op Mode field
- 000 - Byte
- 001 - Word
- 010 - Long word

The result is stored in the specified data register. The effective address specifies the source and the following addressing modes cannot be used: 2, 13, 14.*

Op Mode field
- 100 - Byte
- 101 - Word
- 110 - Long word

The result is stored in the effective address and the following addressing modes cannot be used: 1, 2, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
ORI	8, 16, 32 CCR X N Z V C - * * 0 0	Inclusive OR Immediate	The immediate data is inclusive ORed with the data in the destination and the result is left in the destination.

Opword Format

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	0	0	0	0	Size	Effective Address Mode	Effective Address Register
Word data (16 bits)								Byte data (8 bits)		
Long data (32 bits)										

Size field
- 00 - Byte — The data is the lower byte of the data word.
- 01 - Word — The data is the entire 16 bits of the data word.
- 10 - Long word — The data is the two immediate words.

The effective address is the destination and the following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
ORI to CCR	8 CCR X N Z V C * * * * *	Inclusive OR Immediate data to Condition Code Register	The immediate data is inclusive ORed with the CCR and the result left in the CCR.

Opword Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 bits)							

*The addressing modes will be covered in future issues.

MICRO

MICRObits *(continued)*

OSI Peter Packer

Pack widgits into boxes and ship them out in the elevators before your defective robot assistant *unpacks* them or packs *you*! An original arcade game of cunning and skill that increases in difficulty each round. 8K tape $14.95.

Watts Ware
153 Madrona Drive
Anacortes, WA 98221

OSI - Affordable DBMgr

8" single/dual floppy under OS65D V3.3 video. User-defined files with formatted screen viewing and inputting. Features: find, update, delete, paging, coding, and 'screen,' 'quick,' and 'format' dump. $55.00. Label print option — $25.00. Report Generator (January 1983), manual only — $10.00.

Bunin & Ward Computer Services
P.O. Box 895 Church Street Sta.
New York, NY 10008
(212) 434-5760

Low-Cost Software

Unique programs and hardware kits to adapt small computers to the real world. Control machines, make music, build test equipment and security systems, etc. For information, write to us describing your system and interests. Include stamped self-addressed envelope.

S.W. Associates
45 Furman Drive
Wayne, NJ 07470

OSI Super Defender

Play this great arcade game at home. All machine code includes: scanner, smart bombs, laser fire, moving mountains, and more. Save your humanoids from the alien landers. Very smooth (half-character moves) graphics. $14.95 for C1, 2, 4 tape or 5¼" disk.

DMP Systems
319 Hampton Blvd.
Rochester, N.Y. 14612

Dynamite PET/CBM Accessories!

Write-protect switches/indicators for 2040/4040 disk drives. Real world software at low cost. 2114 RAM adapter (replaces obsolete 6550's) and 4K memory expansion for "old" 8K PETs. Hundreds of satisfied customers. Write for *free* catalog!

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

MICRO

Programmable Character Generator for OSI

by Colin Macauley

Design your own character set and save the characters in a form suitable for incorporation into an EPROM.

Character Generator
requires:
OSI Superboard

While developing software for a minimum chip homebrew 6502 system, it was necessary to produce a character generator. I wrote the program for an 8K OSI Superboard II to draw characters on the OSI video and save these characters in RAM. The characters could then be incorporated in an EPROM, or transferred to the homebrew system. The program was made fairly general, as the homebrew computer included the capability of a variable character depth, whereas the OSI is restricted to 8 × 8 characters. Although the program was intended for a specific purpose, it is equally useful in developing alternate character generators for an OSI. Thus, if games are a major attraction you may wish to define new characters (e.g., Space Invader aliens) for unused characters in your OSI character set. Accordingly, the new character set may then be loaded into a 2K EPROM (2716) and replace the original OSI character-generator ROM.

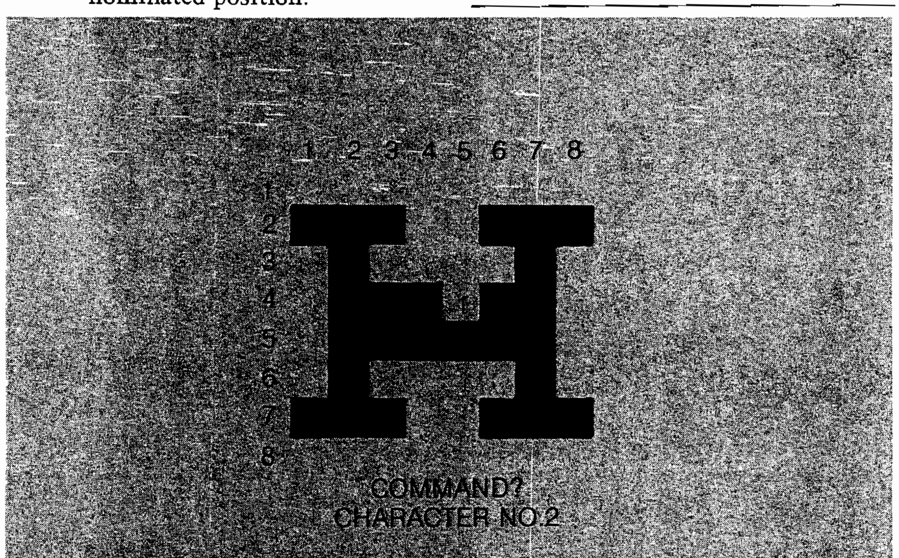
The MEMORY SIZE? cold start prompt should be restricted to 6000. This will prevent overwriting the character-generator RAM that commences at $1800 (6144 decimal), allowing the number of characters to be 256 with a character depth of 8. The required character number is input and a display will appear on the screen to assist in the graphing of the intended character. A cursor in the top left-hand corner indicates the bit currently being altered.

The key commands available for manipulating the cursor are as follows:

- "1" The indicated bit is set and the cursor is shifted. A block character will be inserted at the former cursor position.
- "0" The indicated bit is cleared and the cursor is shifted. A blank character will be inserted at the former cursor position.
- "H" The cursor will move from its present position to its home position (i.e., top left-hand corner of display).
- "D" The cursor will move down a row of the display.
- "F" The cursor will be shifted to the next bit without modifying the status of the previous bit.
- "ESC" Return to BASIC.
- "CR" Enter displayed character into "character-generator" RAM at nominated position.
- "R" A prompt for the number of a predefined character will be requested. This character will then be displayed and may be modified to form the basis of a new character.

Set bits will be indicated by a block and cleared bits will be blanked to allow for an enlarged graphical representation of the character being created. The cursor will be either a "1" or a "0" to enable the condition of that bit to be readily identified. The 2K character generator may be saved on cassette, using well-known machine code save programs, or used directly by an EPROM programmer.

Colin Macauley is a member of the firm of Callinan and Associates, Patent Attorneys and a physicist. He uses a modified OSI Superboard II and is interested in utility-type programming. He may be contacted at 39 Shoalhaven St., Werribee, Victoria 3030, Australia.



The figure illustrates use of the program in constructing a "car" character. The "1" cursor indicates that this bit has been set.

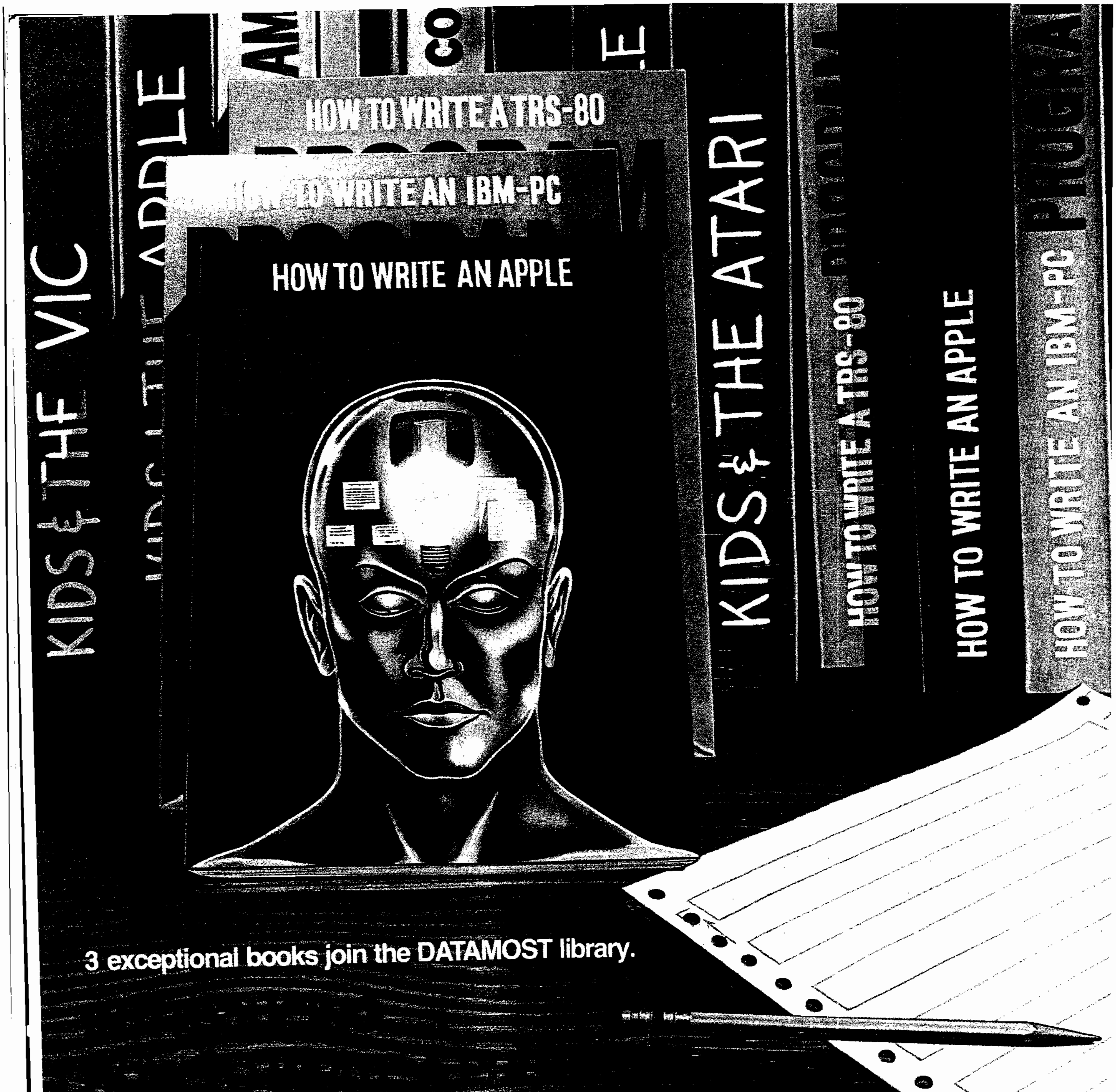
Listing 1: **Programmable Character Generator**

```
4 REM LOAD USR ROUTINE
5 GOSUB350
10 FORX=1TO32:PRINT:NEXTX
20 PRINT"PROGRAMMABLE CHARACTER GENERATOR":PRINT
30 PRINT"COPYRIGHT 1981 COLIN MACAULEY":PRINT
40 INPUT"NO. OF CHARACTERS, IN GROUPS OF 16":A
50 IF(A/16)-INT(A/16)<>0ORA>256THEN40
55 POKE11,162:POKE12,2
60 PRINT:INPUT"CHARACTER DEPTH. 1 TO 16";B
70 IFB>16THEN60
80 PRINT:INPUT"NEW CHARACTER SET (Y/N)":A$
90 IFMID$(A$,1,1)<>"Y"THEN110
95 REM BLANK CHAR. GEN. RAM
100 FORX=6144TO8191:POKEX,32:NEXTX
110 C=6143
120 PRINT:INPUT"CHARACTER NO.":D
130 IFD>ATHEN120
135 REM SET UP SCREEN
140 GOSUB600
210 REM USR ROUTINE SAVES REGISTERS & GETS CHAR. FROM KEYBD
220 Z=USR(Z):M=0
230 W=PEEK(216)
235 REM CHECK WHICH KEY PRESSED
236 REM "0" KEY?
240 IFW<>48THEN260
245 Q=32:GOSUB400:GOTO220
250 REM "1" KEY?
260 IFW<>49THEN270
265 Q=161:GOSUB400:GOTO220
268 REM "H" KEY?
270 IFW<>72THEN280
274 POKEV,UC:Y=53448:UC=PEEK(Y):L=1:V=Y:E=48
275 IFUC=161THENE=49
276 POKEY,E:Y=53415:GOTO220
278 REM "D" KEY?
280 IFW<>68THEN290
285 GOSUB500:GOTO220
288 REM "F" KEY?
290 IFW<>70THEN300
295 Q=UC:GOSUB400:GOTO220
298 REM "ESC" KEY?
300 IFW=27THENEND
305 REM "CR" KEY?
310 IFW<>13THEN320
315 GOSUB700:GOTO130
318 REM "R" KEY?
320 IFW=82THENGOSUB900
330 GOTO220
340 REM LOAD USR SUBR.
350 X=674:FORY=0TO15:READA:POKEX+Y,A:NEXTY
360 DATA72,138,72,152,72,32,186,255,133,216,104,168,104,
    170,104,96
370 RETURN
390 REM SUBR. FOR KEYS "0,1 OR F"
395 REM SHIFTS CURSOR & SETS OR RESETS INDICATED BITS
400 X=Y+(L*32)+8:P=V+1:IFP>XTHENM=L+1
410 POKEV,Q:IFM>BTHEN480
420 IFM>0ANDM<>LTHEN440
430 V=P:GOTO450
440 V=Y+1+(M*32):L=M
450 UC=PEEK(V):E=48
460 IFUC=161THENE=49
470 GOTO490
480 UC=PEEK(V):E=48:IFUC=161ORUC=49THENE=49
485 IFUC=48THENUC=32
490 POKEV,E:RETURN
495 REM SUBR. FOR "D" KEY-SHIFTS CURSOR DOWN A LINE
500 L=L+1:IFL>BTHENL=L-1:GOTO540
510 POKEV,UC:V=V+32:UC=PEEK(V):E=48
520 IFUC=161THENE=49
530 POKEV,E
540 RETURN
590 SUBR. FOR DRAWING WORKSHEET FOR CHAR.
600 FORX=1TO32:PRINT:NEXTX
610 X=53415:F=48
620 FORZ=1TO8:POKEX+Z,F+Z:NEXTZ
640 FORZ=1TOB:W=Z:IFW>9THENW=W-10
645 POKEX+(32*Z),48+W:NEXTZ
650 Y=53448:UC=PEEK(Y):L=1:V=Y:E=48
660 IFUC=161THENE=49
```

Listing 1 *(continued)*

```
670 POKEY,E:Y=Y-33
680 A$="COMMAND?"
685 PRINTCHR$(13)"  CHARACTER NO.":D;
690 FORX=1TO8:POKE54053+X,ASC(MID$(A$,X,1)):NEXTX:RETURN
695 REM SUBR. FOR "CR" KEY
698 REM SAVES CHAR. IN "CHAR. GEN." RAM AT CORRECT POSITION
700 POKEV,UC
710 Z=Y
720 FORX=1TOB
730 F=Z+(32*X):G=0
740 FORH=1TO8
750 I=PEEK(F+H):J=0:IFI=161THENJ=1
760 G=G+J:IFH=8THEN780
770 G=2*G
780 NEXTH
790 POKEC+((X-1)*A)+D,G
800 NEXTX
805 PRINT
810 INPUT"NEXT CHARACTER NO.";D
820 RETURN
880 REM SUBR. FOR "R" KEY-DRAWS REQUIRED CHAR. ON SCREEN
900 PRINT:INPUT"NO. OF CHARACTER TO BE REVIEWED";K
910 IFK>ATHEN900
920 GOSUB600:Z=Y
930 FORX=1TOB
940 F=C+((X-1)*A)+K:I=PEEK(F)
950 FORH=1TO8:R=INT(2↑(H-1)+.5):N=128/R
960 J=INT(I/N)
970 IFJ=1THENPOKE(Z+(X*32)+H),161:I=I-N
980 NEXTH:NEXTX
990 UC=PEEK(Y+33):L=1:V=Y+33
1000 E=48:IFUC=161THENE=49
1010 POKEV,E
1015 IFUC=48THENUC=32
1020 RETURN
```

MICRO


HOW TO WRITE A TRS-80
HOW TO WRITE AN IBM-PC
HOW TO WRITE AN APPLE
KIDS & THE VIC
KIDS & THE ATARI
HOW TO WRITE A TRS-80
HOW TO WRITE AN APPLE
HOW TO WRITE AN IBM-PC
3 exceptional books join the DATAMOST library.

MICRO™

Updates and Microbes

Updates

John Beckett of Collegedale, TN, sent in this revision to "A Homespun 32K Color Computer" (53:91).

Solder the chips together rather than expecting hand-bent pins to make good contact. It is best to put a ferrite bead around the wire connected to the 6883 chip, just before it reaches the 6883. Failing this, use a 33-ohm resistor. This is done in Tandy's 32K version and is recommended by Motorola in their 6883 data sheet. Later models of the PC board have a place on the PC board where you may connect the lead from the extra bunk of chips, that avoids soldering directly to the 6883.

Myron Pulier, M.D., from Teaneck, NJ, sent in this update:

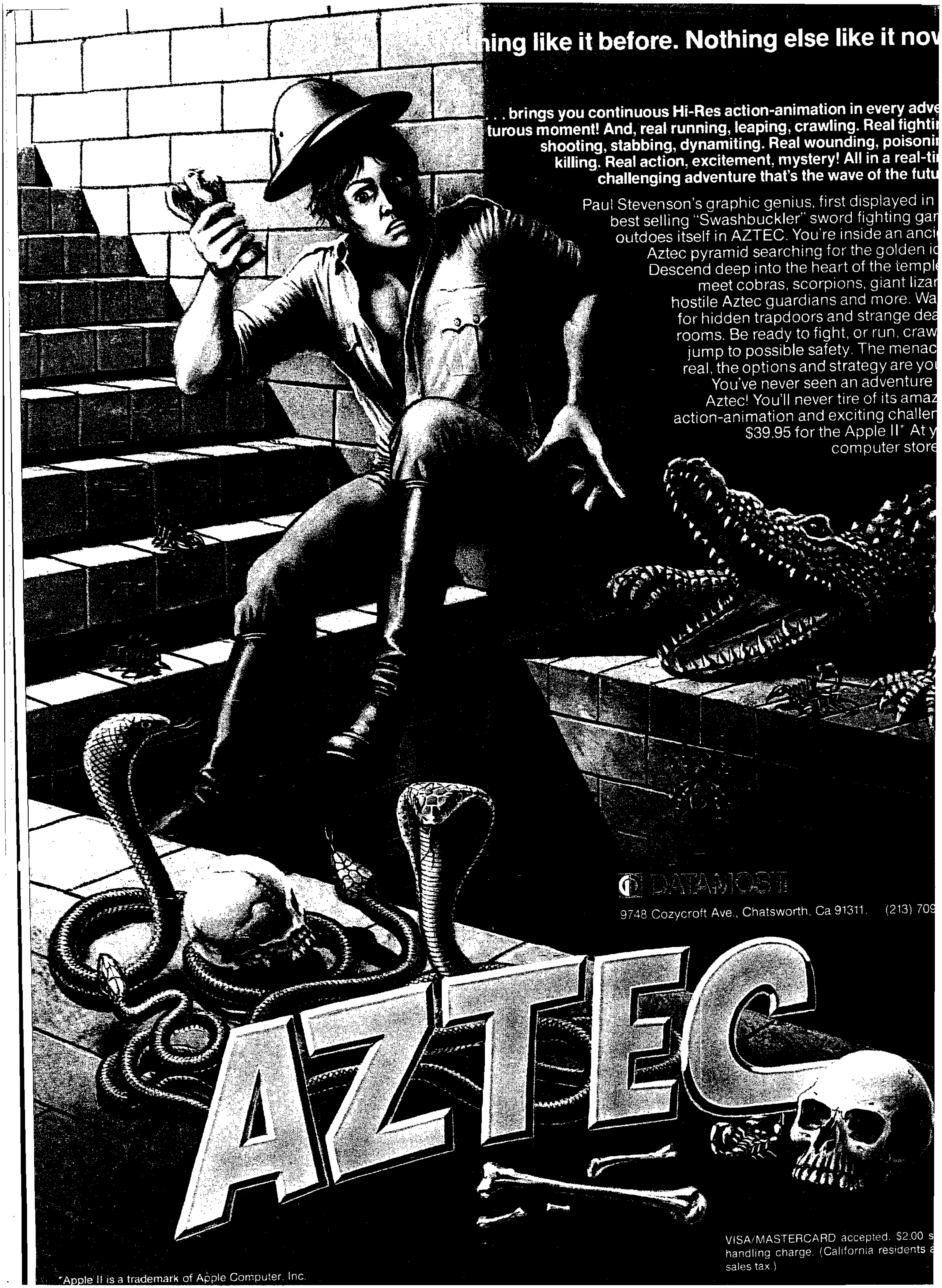
The LISZT program in the May, 1982 issue of MICRO (48:37) makes readable BASIC listings. The authors used a disk zap utility program to get lower-case characters in the DATA statements. Lacking such, I used the temporary patch, shown in listing 1, appended to LISZTER.

This patch creates new DATA strings after converting all alphabetic characters to lower case except the first one in each string. These new strings are read into a TEXT file named "DF". When this file is EXECed it replaces the LISZTER DATA statements with the new ones and displays the result for confirmation. The patch itself is removed so the converted program may be SAVEd.

To operate the zap bypass program, LOAD LISZTER, type in the enclosed statements, and save the combined program as "TEMP" in case something goes wrong. Then type "RUN 1000". If the run is successful, save the program now in memory as your new copy of LISZTER.

(Continued on page 98)

```
 1000   ***************   ZAP BYPASS FOR LISZT

 1005   D$ = Chr$(4)
        QT$ = Chr$(162)
        BR$ = QT$ + ","
 1010   Print D$"OPENDF"
        Print D$"DELETEDF"
        Print D$"OPENDF"
        Print D$"WRITEDF"
 1015   Print "SAVELISZTER.PATCH
 1020   Print 87"DATA";
        A = 1
        B = 25
        Gosub 2005
 1025   Print 88"DATA";
        A = 26
        B = 50
        Gosub 2005
 1030   Print 89"DATA";
        A = 51
        B = 51
        Gosub 2005
 1035   Print 90"DATA";
        A = 52
        B = 75
        Gosub 2005
 1040   Print 91"DATA";
        A = 76
        B = 107
        Gosub 2005
 1045   Print "DEL 1000,3040"
        Print "INVERSE:?"QT$"DATA CONVERTED"
 1050   Print "NORMAL:SPEED=180:LIST 87-91:SPEED=255
 1055   Print D$"CLOSE"
        Print D$"EXEC DF"
 1060   End

 2000   ***************   CONVERT ONE LINE

 2005   For J = A To B
 2010        Read ST$
             Print QT$;
 2015        LF = 0
             L = Len(ST$)
 2020        If L Then
                  Gosub 3005
 2025        If J = B Then
                  Print QT$
 2030        If J < B Then
                  Print BR$;
 2035   Next
 2040   Return
 3000   ***************   CONVERT ONE STRING

 3005   For I = 1 To L
 3010        C$ = Mid$(ST$,I,1)
 3015        If "@" < C$ And C$ < Chr$(219) Then
                  C$ = Chr$(Asc(C$) + 32 * LF)
                  LF = 1
 3020        Print C$;
        Next
        Return


        END OF LISTING

PROGRAM LENGTH = 659 BYTES,   TOTAL OF 27 LINE NUMBERS

51 TOTAL NON-REM STATEMENTS,   3 TOTAL REMARKS

END
```


ing like it before. Nothing else like it no
. . . brings you continuous Hi-Res action-animation in every adv
turous moment! And, real running, leaping, crawling. Real fight
shooting, stabbing, dynamiting. Real wounding, poisoni
killing. Real action, excitement, mystery! All in a real-ti
challenging adventure that's the wave of the futu
Paul Stevenson's graphic genius, first displayed in
best selling "Swashbuckler" sword fighting ga
outdoes itself in AZTEC. You're inside an anc
Aztec pyramid searching for the golden i
Descend deep into the heart of the templ
meet cobras, scorpions, giant liza
hostile Aztec guardians and more. Wa
for hidden trapdoors and strange de
rooms. Be ready to fight, or run, craw
jump to possible safety. The mena
real, the options and strategy are yo
You've never seen an adventure
Aztec! You'll never tire of its ama
action-animation and exciting challe
$39.95 for the Apple II* At y
computer stor
DATAMOST
9748 Cozycroft Ave., Chatsworth, Ca 91311. (213) 70
AZTEC
VISA/MASTERCARD accepted. $2.00 s
handling charge. (California residents
sales tax.)
*Apple II is a trademark of Apple Computer, Inc.

Utilizing the 6502's Undefined Operation Codes

by Curt Nelson, Richard Villarreal, and Rod Heisler

This method allows you to use the 6502's undefined op codes to design new and individualized pseudo-instructions under program control. A simple hardware device attached to the data bus forces a simulated BRK command when an illegal op code is detected.

Utilizing Undefined Op Codes
requires:

Hardware modification to a 6502 microcomputer

Fetch Cycle

Before the Central Processing Unit (CPU) can execute an instruction it must first get the hexadecimal code from memory. This process is called a fetch cycle. The fetch cycle is identical to the data read cycle except for the SYNC line operation, which rises to a logic level one (5V) shortly after the fetch cycle is initiated.

The fetch cycle (figure 1) starts when the system clock, Ø2, falls to a logic level 0 (0V). For a 1MHz system clock the fetch cycle normally requires 1000 nano seconds, or one micro second. During this 1000 nano-second period several events occur in well-ordered sequence. First, the CPU outputs the current value of the program counter on the address bus. This is the address location of the next instruction. The specified memory then outputs the op code to the data bus. The CPU reads the op code from the data bus just before the end of the cycle.

The interval in which the Trapper has to operate extends from the time the memory device presents the op code to the data bus until the CPU latches it internally. In this time it must determine if the op code is valid or not, and force a BRK (00) if it is illegal. The Trapper described in the next section requires a maximum of 150 nano seconds to operate, leaving a minimum of 525 nano seconds for the memory to present valid data to the data bus. This, of course, precludes the use of very slow memory devices but is adequate for most microcomputer systems.

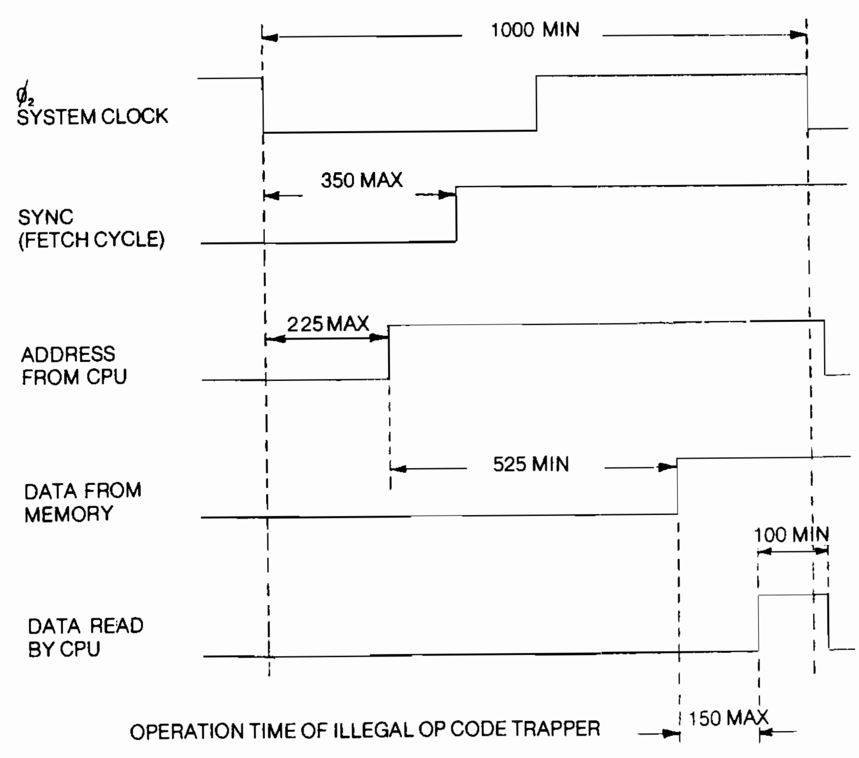
Hardware

The Trapper (figure 2) samples the data bus in a parallel mode. The data lines are first buffered through IC4 and IC5 and then used to form the address to IC3, a 256 × 4 PROM. IC3 is always enabled and is programmed to output a logic state one for an illegal op code and a logic state zero for a legal code. Only one of the three PROM outputs is used; the others are not programmed.

The falling edge of the Ø2 clock initiates the timing cycle for IC1, a monostable multivibrator. The output of IC1 goes high after a period of time determined by the RC network. The time-out is set for approximately 750 nano seconds. The leading edge time out from IC1 is used to clock IC2, a dual D flip-flop. The SYNC line is tied to the clear input of IC2 through two buffers. This combination of inputs to IC2 assures that its output will go high only if these three conditions are met: the SYNC line is high (fetch cycle), an illegal op code has been fetched, and IC1 has timed out.

The outputs of IC2 are used to drive open collector inverters tied directly to the data bus. When the inputs to the in-

Figure 1: **Timing Diagram for the 6502 Fetch Cycle (All times in nano (10^{-9}) seconds)**



verters are high (illegal op code), the outputs force the data lines to a logic state zero, simulating a BRK command. When the inputs to the inverters are low, as under non-trapping conditions, the output appears as a high impedance to the data bus. If the data lines are pulled low, they are released when the SYNC line goes low during the next clock cycle.

Software

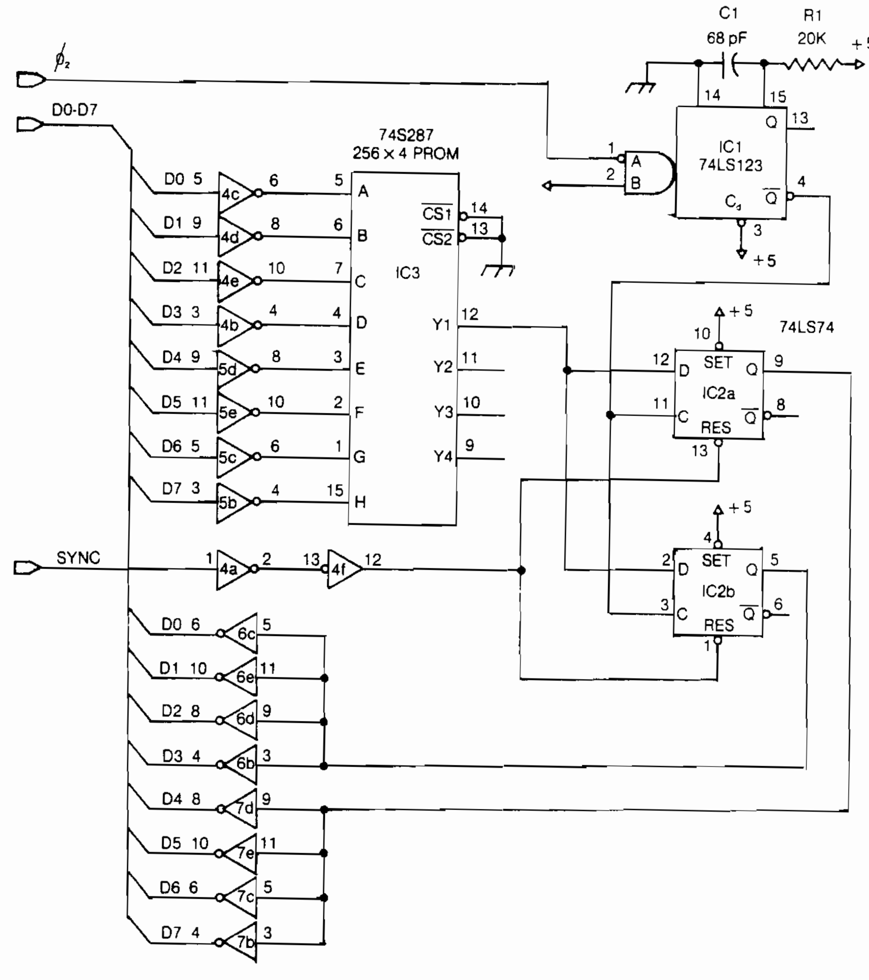
The task of the software is two-fold. First, it must determine if the break was the result of an illegal op code or a BRK instruction. Second, if the Trapper forced the break, it must retrieve the illegal op code and direct the CPU to the proper software routines.

The CPU handles the software BRK and an IRQ (Interrupt ReQuest) similarly, except for one small feature. A BRK command sets the break bit (bit four) in the processor status register. The CPU will then do an indirect jump through the IRQ vector at FFFE and FFFF. The user must load the address of the break-handling routine into the IRQ vector prior to the detection of an illegal op code, to direct the CPU to the user routine. Listing 1 shows the software used to change the IRQ vector. A starting address of $0300 was used for the break service routine, but this is arbitrary.

The user's break-handling routine must determine whether a BRK or an IRQ was encountered. This is done by retrieving the processor status from the stack (it was automatically pushed there when the break occurred) and examining the break bit. If it is determined that bit four is set and hence a break has occurred, it retrieves the last op code. This is easily done because the address of this instruction plus two was also pushed on the stack when the program was interrupted. If this instruction was a BRK, control is passed back to the system monitor. If, on the other hand, it was an illegal op code, control is passed to a user program that implements new micro-coded instructions.

There are several methods to jump to the user code corresponding to each new instruction. The most straightforward way is to use a CMP instruction followed by a BEQ for each element in a list of new hex op codes. If more than just a few instructions are added, a more elaborate scheme may be necessary to reduce the execution time and program length. In this situation

***Figure 2:* Schematic diagram of the illegal op code Trapper. The board is compatible with any 6502 system bus. All lines to the board are generated by the 6502 CPU. C1 is a silver mica capacitor and R1 is a low-temperature coefficient, precision resistor.**



you may want to use a jump table to build this case/select structure.

The break service routine in listing 2 is completely transparent (i.e., all registers are preserved). The illegal op code is returned at address $0042. The address is arbitrary and can be changed to any convenient location.

If the user exits the break service routine at line 23, indicating an IRQ, he should use the following sequence to restore the original registers:

```
PLA
TAX
PLP
PLA
```

If the routine is exited at line 40, indicating a normal BRK command, the following sequence should be used:

```
PLP
PLA
```

Programming the PROM is understood by examining figure 2. Since the system data bus is connected to the address lines of the PROM, the hex op codes become the address to this device. Therefore, all legal op code-based addresses store 0000 and all illegal addresses store 0001.

Conclusion

This method of detecting illegal op codes is really a hardware implementation of a macro assembler directive. Although the execution time and memory space required are more than the standard JSR technique, writing and debugging programs is more straightforward when microcoded routines are

Figure 3

Number	Type	+5V	Gnd
IC1	74LS123	16	8
IC2	74LS74	14	7
IC3	74S287	16	8
IC4,5	74LS04	14	7
IC6,7	7405	14	7

Listing 1: Software to modify the IRQ vector to point to a user program.

```
0800                ;SETTING UP THE IRQ VECTOR
0800                ;
0800                ;
0200                        ORG $200
0300                USRPRG  EQU $0300           ;ADDRESS OF USER PROGRAM
FFFE                IRQLOW  EQU $FFFE           ;LOW ADDRESS OF IRQ VECTOR
FFFF                IRQHIG  EQU IRQLOW+$1       ;HIGH ADDRESS OF IRQ VECTOR
0200                ;
0200                ;
0200                ;INITIALIZATION
0200                ;
0200                ;
0200 A9 00                  LDA #USRPRG         ;SET IRQ VECTOR TO USER BREAK
                                                 ROUTINE
0202 8D FE FF               STA IRQLOW
0205 A9 03                  LDA /USRPRG
0207 8D FF FF               STA IRQHIG
020A                ;
020A                ;
020A                ;
020A                ;
020A                ;MAIN PROGRAM
```

Listing 2: Program to handle a break service routine. Determines whether a break or an IRQ has interrupted the system and transfers control to the proper location.

```
0800                ;BREAK SERVICE ROUTINE
0800                ;
0800                ;
0800                ;
0300                        ORG $300
0380                IRQSER  EQU $380            ;STANDARD IRQ SERVICE
03A0                USRBRK  EQU $3A0            ;STANDARD BREAK SERVICE
0040                SAVLOW  EPZ $40
0041                SAVHIG  EPZ SAVLOW+$1
0042                SAVOPC  EPZ SAVHIG+$1
0104                FLAG    EQU $104
0105                ADDLOW  EQU $105
0106                ADDHIG  EQU ADDLOW+$1
0300                ;
0300                ;
0300 48                     PHA                 ;PRESERVE ACC
0301 08                     PHP                 ;PRESERVE FLAGS
0302 8A                     TXA
0303 48                     PHA                 ;PRESERVE X
0304 BA                     TSX
0305 BD 04 01               LDA FLAG,X          ;GET FLAGS
0308 29 10                  AND #$10
030A F0 74                  BEQ IRQSER
030C BD 06 01               LDA ADDHIG,X        ;GET ADD + 2 FROM STACK
030F 85 41                  STA SAVHIG
0311 BD 05 01               LDA ADDLOW,X
0314 85 40                  STA SAVLOW
0316 D0 02                  BNE SKIP            ;BR IF NOT ON PAGE BOUNDRY
0318 C6 41                  DEC SAVHIG          ;DEC PAGE
031A C6 40          SKIP    DEC SAVLOW          ;DEC ILLEGAL OPCODE ADDRESS
031C D0 02                  BNE SKIP1           ;BR IF NO PAGE CROSSED
031E C6 41                  DEC SAVHIG          ;DEC PAGE
0320 C6 40          SKIP1   DEC SAVLOW          ;DEC ADDRESS AGAIN
0322 A2 00                  LDX #$00            ;INDEX
0324 A1 40                  LDA (SAVLOW,X)      ;GET ILLEGAL OP CODE
0326 85 42                  STA SAVOPC          ;PRESERVE IT
0328 68                     PLA
0329 AA                     TAX                 ;RESTORE X
032A A5 42                  LDA SAVOPC          ;RETRIEVE ILLEGAL OP CODE
032C F0 72                  BEQ USRBRK          ;BR FOR NORMAL BREAK
032E 28                     PLP                 ;RESTORE FLAGS
032F 68                     PLA                 ;RESTORE ACC
0330                ;
0330                ;
0330                ;
0330                ;USER ROUTINES
0330                ;
0330                ;
0330                ;
0330                ;RETURN TO MAIN PROGRAM
0330                ;
0330                ;
0330 E6 40                  INC SAVLOW          ;BUMP LOW ADDRESS
0332 D0 02                  BNE SKIP2           ;BR IF NO PAGE CROSSED
0334 E6 41                  INC SAVHIG          ;BUMP PAGE
0336 6C 40 00       SKIP2   JMP (SAVLOW)
0339                        END
```

incorporated into your program as simple instructions.

A few words of caution: first, it is necessary to acquaint yourself with the user-available monitor subroutines on your system. The SYM-1, for example, has monitor routines to do some of the functions in listing 2. The Apple, as well, has monitor routines that can be used to shorten this program. Second, the illegal op code FF rearranges the stack and hence should be avoided.

You are now in a position to expand the instruction set of your 6502-based system. What instructions should you add? Here are a few suggestions: integer multiply and divide, double precision math operations, jump indirect-indexed, push and pull to a user stack, and memory to memory transfer. You can even add a pseudo B accumulator and a 16-bit index register.

The authors may be contacted at the School of Engineering, Walla Walla College, College Place, Washington 99324.

MICRO

MICRO Short Subjects

VIC Jitter Fixer by David Malmberg

SuperPET Characters by Terry M. Peterson

SuperPET Characters

Terry M. Peterson, 8628 Edgehill Ct., El Cerrito, CA 94530

The SuperPET contains a 4K character-generator ROM in place of the 2K ROM found in normal CBM 8032s. The 4K ROM contains four character sets. In addition to the two PET/CBM character sets found in the 2K ROM, there are two new sets designed by Waterloo Computing Systems — ASCII and APL. The Waterloo ASCII character set is used in the SuperPET by all the Waterloo Micro languages except MicroAPL. This article describes some of the features of the Waterloo ASCII character set that are not well-covered in the Waterloo documentation accompanying the SuperPET.

All the printable ASCII characters — codes 32 to 127 — in the Waterloo ASCII set are pure ASCII. By this I mean they are all recognizable duplicates of the corresponding character found in an ASCII table. Furthermore, the PRINTed codes are *identical* to the screen POKE codes for a given character! Many of the screen control codes are consistent with normal printer usage; e.g., cursor-down = 10 (LF), cursor-back = 8 (BS), and clear-screen = 12 (FF). This means that turning neatly formatted CRT output into neatly formatted hardcopy on an ASCII printer (like the MX-80) is much easier than with the CBM character set (the one Gary Huckel of TNW so appropriately calls 'half-ASCII').

Notice I said the printable characters, 32 to 127, have the same PRINT and POKE codes; but what about POKEing the ASCII control codes 0 to 31? By experiment you will find these codes do not all cause the same action when POKEd as when PRINTed. The POKE characters and PRINT actions of these codes are shown in table 1. The codes 0 and 14-30 give an odd little white box when POKEd or PRINTed. Code 31 gives the Greek letter μ, POKEd or PRINTed. Codes 1-11, when POKEd, give eleven line graphic characters that are useful for drawing outline boxes or grids. These characters are similar to the graphics characters available on the Epson MX printers with Graphtrax Plus. They are also very like one subset of the CBM graphics characters; the shifted-zero is an example (see table 1). When PRINTed, most of the codes from 1 to 13 perform some sort of control function, as shown in table 1.

What about the high-order bit that gives the codes 128 to 255? Either PRINTed or POKEd, all the codes from 128 to 255 reproduce, in reverse field, their X-minus-128 POKEd counterparts. Although all these reverse-field characters are available (and Waterloo didn't usurp the RVS key for another function), Waterloo ASCII apparently has no reverse control code such as in the CBM character set. Therefore, to print a reverse-field string, each character must be extracted from the string and transformed by adding 128. For example in microBASIC:

```
FOR I = 1 TO LEN(CHARSTRING$)
CHAR$ = STR$(CHARSTRING$,I,1)
RVSCHAR$ = CHR$(128+ORD
   (CHAR$))
PRINT RVSCHAR$;
NEXT I
```

Perhaps this encumbrance is the reason reverse-field characters aren't mentioned in Waterloo's documentation?

VIC Jitter Fix

David Malmberg, 43064 Via Moraga, Fremont, CA 94539

In my October 1981 MICRO article (41:54), "VIC Light Pen-Manship," I pointed out that the locations in the VIC chip that return the light pen's horizontal screen position ($9006) and vertical screen position ($9007) are

Table 1

Code	Mnemonic	ASCII Name	Print Action	POKE Character	CBM Graphics Equivalent	Epson Graphtrax+ Equivalent
1	SOH	Start Heading	Home cursor	Vertical line	CHR$(221)	CHR$(156)
2	STX	Start TeXt	? (Run)	Horizontal line	CHR$(195)	CHR$(157)
3	ETX	End TeXt	? (Stop)	Lower right corner	CHR$(189)	CHR$(154)
4	EOT	End Transmission	Delete	Lower left corner	CHR$(173)	CHR$(153)
5	ENQ	ENQuiry	Insert	Upper left corner	CHR$(176)	CHR$(134)
6	ACK	ACKnowledge	Erase to EOL	Upper right corner	CHR$(174)	CHR$(149)
7	BEL	ring BELl	Cursor right(!)	Bottom middle corner	CHR$(177)	CHR$(158)
8	BS	Back Space	Cursor left	Left middle corner	CHR$(171)	CHR$(150)
9	HT	Horizontal Tab	Tab	Top middle corner	CHR$(178)	CHR$(152)
10	LF	Line Feed	Cursor down	Right middle corner	CHR$(179)	CHR$(151)
11	VT	Vertical Tab	Cursor up	Cross	CHR$(219)	CHR$(159)
12	FF	Form Feed	Clear screen	Little white box		
13	CR	Carriage Return	Carriage return	Little white box		

Updates and Microbes

(Continued from page 91

Robert R. Ringel of Comstock Park, MI, found a bug in COMPRESS (52:89):

If COMPRESS is processing the token for NEXT ($82) one byte before a page boundary, it can lose that token when it goes to update its addresses for the new page.

To correct this problem, replace the STX instruction at $9088 with $86E3 and the corresponding LDX instruction at $908E with $A6E3. Zero page location $E3 is an unused location that works well for a temporary location in this instance.

COMPRESS Removes Variables

Warren Friedman, from Berkeley, CA, sent in this update:

The program COMPRESS, well written and clearly described by Barton M. Bauers (MICRO 52:89) removes any variable names appearing after NEXT statements. It does this by ignoring all characters until the following colon or the end of the program line (see $93EC - $93EF). This could cause problems in two cases.

The first problem occurs when several variables are used with one NEXT, as in NEXT I,J. The second case is when a NEXT variable *must* be stated. This may occur with nested loops in which the inner loop NEXT is the result of an IF...THEN statement. *(Editor's note:* A poor programming practice. Loops should be cleared before exiting or else stack overflow can occur.)

These problems with NEXT can be solved by treating NEXT in the same way an IF statement is dealt with, which is to leave it as the programmer wrote it. (Bauers calls this a Terminal Command.) This is done by changing one byte of COMPRESS. First BLOAD COMPRESS, then, in BASIC, POKE 37871,72 (or, in the monitor, enter 93EF:48). Then BSAVE COMPRESS, A$9000,L$600.

Similarly, programmers who use & statements (and who do not mind haveing LET statements remain in the program, if there are any) can change lines 460 and 461. In BASIC, POKE 37873,202 : POKE 37874,240 : POKE 37875,68 (or, in the monitor, enter 93F1:CA F0 44). The two lines of COMPRESS become

```
C9 CA   CMP #$CA   ;is it '&'?
F0 44   BEQ IF     ;yes
```

MICRO™

Short Subjects *(continued)*

subject to noise. These noisy registers can cause the pen's readings to jitter about the screen. The October article presented a machine-language routine that eliminated this jitter problem by taking seven separate readings of the pen's coordinates, sorting them, and returning the median readings (thus ignoring the jittery readings that should be at one extreme or the other of the sorted list). This routine also calculated the light pen's screen row and column for the special case of an Atari or Commodore light pen.

Having recently experimented with the use of the Atari VCS's game paddles with the VIC, I discovered that the left ($9008) and right ($9009) game paddle registers also suffer from jitter problems. This can be very frustrating when you are playing a paddle game like PONG or BREAKOUT and the paddles occasionally bounce around the screen as if they were possessed by evil computer spirits. The severity of the problem seems to be a function of the game paddle unit itself — my neighbor's paddles are much noisier than mine.

The BASIC subroutine, given in listing 1, POKEs into the VIC's cassette buffer a machine-language routine that provides a general solution to this jitter problem. To use the routine in your paddle programs, follow these steps: 1. append the subroutine to your game paddle program, 2. GOSUB 1000 at the start of the program to load the machine code into the cassette buffer, 3. SYS(828) to read *both* paddle registers, and 4. get the left paddle's un-jittered reading by PEEKing 936 and the right by PEEKing 937. Be sure to use this routine cautiously in any program that is doing tape input or output because of the risk of clobbering the machine code in the cassette buffer.

This same routine may also be used to un-jitter the light pen reigsters by deleting lines 1190 and 1200. The resulting machine code is more universal than the version given in the October 1981 article because it can be used with any light pen, rather than just the Atari and Commodore pens.

Should other VIC chip registers be discovered that suffer from jitter, they can be easily handled with this routine by merely POKEing the low byte of their addresses into locations 835 and 857. See line 1190 of the listing where this is done for the game-paddle registers.

Because this program is very similar to the one presented in my previous article, a full assembly listing is not given.

Jitter Fixer Subroutine

```
1000 REM MACHINE LANGUAGE ROUTINE TO READ 'JITTERY' VIC LOCATIONS
1010 REM SUCH AS LIGHT PEN COORDINATES OR GAME PADDLE SETTINGS
1020 REM SYS(828) TO READ --- VALUES RETURNED IN LOCATIONS 936 AND 937
1030 FOR I= 828 TO 938 :READ DC:POKE I,DC:NEXT I
1040 DATA 162,0,160,3,132,152,173,6,144
1050 DATA 160,171,132,151,32,133,3,165
1060 DATA 151,24,109,170,3,133,151,144,2
1070 DATA 230,152,173,7,144,32,133,3,232,236
1080 DATA 170,3,240,9,165,162,197,162,240
1090 DATA 252,76,62,3,173,170,3,74,168
1100 DATA 177,151,141,169,3,169,171,133
1110 DATA 151,169,3,133,152,177,151,141
1120 DATA 168,3,96,142,168,3,172,168,3
1130 DATA 192,0,240,22,136,209,151,200
1140 DATA 176,16,136,141,168,3,177,151
1150 DATA 200,145,151,136,173,168,3,56
1160 DATA 176,230,145,151,96,0,0,7
1170 REM ROUTINE WILL NORMALLY READ GAME PADDLES
1180 REM TO READ LIGHT PEN COORDINATES, DELETE THE NEXT TWO STATEMENTS
1190 POKE 835,8:POKE 857,9
1200 POKE 868,169:POKE 869,255:POKE 870,233:POKE 871,1:POKE 872,208
1210 RETURN
```

MICRO™

MICRO™

New Publications

So we can list more of the many new books now available, we are offering New Publications in a different format. We think you'll find this increased sampling of computer literature useful.

Library of PET Subroutines, by Nick Hampshire. Hayden Book Company, Inc. (Rochelle Park, NJ), 1982, 140 pages, paperback.
ISBN: 0-8104-1050-8 $14.95

PET Graphics, by Nick Hampshire. Hayden Book Co., Inc. (Rochelle Park, NJ), 1982, 218 pages, paperback.
ISBN: 0-8104-1051-6 $16.95

Computer Consciousness: Surviving the Automated 80's, by H. Dominic Covvey and Neil Harding McAlister, Addison-Wesley Publishing Company, Inc. (Reading, MA), 1982, 211 pages, paperback.
ISBN: 0-201-01939-6 $6.95

Atari Sound and Graphics, by Herb Moore, Judy Lower, and Bob Albrecht. John Wiley & Sons, Inc. (605 Third Ave., N.Y.C., NY 10158), 1982, 234 pages, paperback.
ISBN: 0-471-09593-1 $9.95

The Creative Apple, Edited by Mark Pelczarski and Joe Tate. Creative Computing Press (Morris Plains, NJ), 1982, 448 pages, paperback.
ISBN: 0-916688-25-9 $15.95

The VisiCalc Book, Apple Edition, by Donald H. Beil, Reston Publishing Company, Inc. (Reston, VA), 1982, 301 pages, paperback.
ISBN: 0-8359-8398-6 $14.95

The Third Book of Ohio Scientific, by S. Roberts. ELCOMP Publishing, Inc. (Postbox 1194, Pomona, CA 91769), 1982, 137 pages, 5¼ × 8¼ inches, paperback.
ISBN: 3-921682-77-0 $17.95

Kilobaud Klassroom, by George Young and Peter Stark. Wayne Green Books (Peterborough, NH 03458), 1982, 419 pages, 6 × 9 inches, paperback.
ISBN: 0-88006-027-1 $14.95

Computers for Kids, by Sally Greenwood Larson. Creative Computing Press (P.O. Box 789-M, Morristown, NJ 07960), 1981, 73 pages, paperback.
ISBN: 0-916688-21-6 $4.95

Ciarcia's Circuit Cellar, Volume III, by Steve Ciarcia. BYTE/McGraw-Hill (70 Main St., Peterborough, NH 03458), 1982, 228 pages, 8¼ × 11 inches, paperback.
ISBN: 0-07-010965-6 $12.95

Techniques for Creating Golden Delicious Games for the Apple Computer, by Howard M. Franklin, Joanne Koltnow, and Leroy Finkel. John Wiley and Sons, Inc. (605 Third Ave., N.Y.C., NY 10158), 1982, 150 pages, paperback.
ISBN: 0-471-09083-2 $12.95

BASIC for Business by Douglas Hergert. SYBEX (2344 Sixth Street, Berkeley, CA 94710), 1982, 223 pages, 7 × 9 inches, paperback.
ISBN 0-89588-080-6 $12.95

Computers for People by Jerry Willis and Merl Miller. Dilithium Press (P.O. Box 606, Beaverton, OR 97075), 1982, 200 pages, 5¼ × 8½ inches, paperback.
ISBN: 0-918398-64-9 $7.95

MICRO™

MICRO™

Reviews in Brief

Product Name: **Spellmaster**
Equip. req'd: Commodore 80-column screen and dual disk (40- and 64-column versions expected soon)
For Wordpro files (Wordcraft & Silicon Office versions planned)
Uses functional 4K ROM at $9000
Price: $195
$ 75 for legal or medical dictionary options
Manufacturer: Management Systems Alternatives
6219 Thirteenth Avenue South
Gulfport, FL 33707

Description: Finally, a decent spelling checker for CBM computers! Highly recommended for word-processing writers who do not spell well.

Pluses: It is far faster than its only competitor and has an honest 40,000-word dictionary. *Spellmaster* presents suspect words for editing in context in reverse field on a typical Wordpro screen display. Suspect words may then be easily corrected or added to the dictionary for future reference (up to 3,000 more words on the 4040, and 20,000 more on the 8050). Corrected files are resaved to disk, avoiding the hassle of reloading the word processor and searching for the errors. The program is mostly self-documenting, though it comes with a typical manual. There is a HELP screen in the program and useful prompts throughout.

Minuses: When editing, it is easy to skip past a word that needs to be repaired or added to the dictionary. At present, there is no way to back up except by aborting and restarting the edit. The company is attempting a fix.

Skill level required: Users should be fairly familiar with Wordpro and willing to spend about an hour reading the *Spellmaster* manual before use.

Reviewer: Jim Strasma

Product Name: **Electric Duet**
Equip. req'd: Apple II or Apple II Plus
Price: $29.95
Manufacturer: Insoft
10175 Barbur Blvd., Suite 202B
Portland, OR 97219
Author: Paul Lutus
Copy Protection: Yes
Language: 6502 Assembly

Description: A software-only music synthesis system for generating 2-part music on an Apple with no additional hardware required.

Pluses: An external speaker can be used to improve fidelity *via* the cassette port. The package includes a music editor for constructing tunes, with several sample tunes. A combined display allows for the simultaneous entering and playing of music. Entered scores can be transposed both in key and in tempo. Each note played may have one of four voices. Notes can be entered either into an editor or played directly from the keyboard. Then the music can be incorporated directly into user programs! The storage format of the music is described for the more advanced programmer who may wish to access the binary score directly.

Minuses: The manual is brief (17 pages) but complete. Although the author has permitted the user to play music directly from the Apple keyboard (using the upper row of keys for one note and the lower for the other), I personally found this feature awkward to use. The editor is much more complete for entering music from the keyboard. As mentioned in the manual it is included only for familiarization. Deletion of a line using the music editor is not a single stroke command. To accomplish a line deletion, a file must be opened so that the line to be deleted is the last. Then deletion will remove it. After working with Musicomp, Paul Lutus' first music editor, I was spoiled by his hi-res display of notes in motion. I would love to have seen that feature retained in Electric Duet. However, by obtaining 2-part music with no hardware, at a fraction of the cost of popular music boards, this program should be considered carefully before investing in more expensive alternatives.

Skill level required: Fairly easy for the novice to master with a little practice.

Reviewer: David Morganstein

Product Name: **Terminal-40**
Equip. req'd: VIC-20
8K (or more) of extra memory
VICMODEM or RS-232 compatible modem
Price: $29.92
Manufacturer: Midwest Micro Associates
P.O. Box 6148
Kansas City, MO 64110
Author: Dr. Jim Rothwell

Description: *Terminal-40* is an extremely powerful telecommunications program for the VIC-20. This machine-language program is fast enough to support up to 2400 baud, is quite flexible, and allows you to specify duplex, parity, wordsize, stopbits, linefeed, and baud rate options. Through software, *Terminal-40* displays a 40-character line with each character represented by a 3 × 6 matrix. All characters are shown as upper case and are quite readable. *Terminal-40* also has a 4K or larger buffer, which can be used to capture copies of the material being transmitted or received for later study or dumping to the printer.

Pluses: A versatile and exceedingly well-done package. The 40-column display is great!

Minuses: Although *Terminal-40* supports the printer, it does not handle the disk, nor is there any way to use it to transmit or receive a program. The program comes on an "auto-start" tape and cannot be copied to disk or another tape.

Documentation: The 20-page manual is clear and comprehensive.

No special skills required.

Reviewer: David Malmberg

Product Name:	**Doubletime Printer**
Equip. req'd:	Apple II Plus Any of the popular printers
Price:	$99.95
Manufacturer:	Southwestern Data Systems P.O. Box 582 Santee, CA 92071 (714) 562-3221

Description: *Double Printer* permits printing to take place as a background task. You can continue to use your computer while it is printing rather than being "frozen out." This should prove particularly valuable in word processing applications.

Pluses: The product is extremely versatile. Applesoft, binary, or text files are printed without conversion. Formatting commands are available and easy to use.

Minuses: The product is not easy to get up and running. It requires a ROM chip change, a board installation, and a diskette boot. All this could be dealer-performed for the more timid user. It *is* worth the trouble.

Documentation: The instructions are well-written but quite technical.

Skill level required: An intermediate familiarity with the Apple is necessary.

Reviewer: Chris Williams

Product Name:	**Apple-Cillin II**
Equip. req'd:	Apple II or Apple II Plus with disk drive (13- or 16-sector)
Price:	$49.95
Manufacturer:	XPS, Inc. 323 York Road Carlisle, PA 17013

Description: This diagnostic utility tests RAM and ROM chips, the disk system, peripheral cards, keyboard, CRT display, printer, tape recorder, game controls, and CPU

(Continued on next page)

Reviews in Brief *(continued)*

registers. Disk tests include sequential and random writing and reading, random track seeking, and drive speed.

Pluses: Single or multiple tests may be repeated continuously, with results optionally printed. The program is menu-driven, user-friendly, fast, and crash-resistant.

Minuses: The style and depth of the documentation are marginal.

Documentation: The 24-page manual is neatly formatted and printed. The writing is comprehensible but often awkward and unpolished. It describes in detail how to use the program, but gives almost no help to analyze and correct problems it finds.

Skill level required: Little skill is needed to run it, but moderate hardware knowledge is required to know what to do about reported problems.

Reviewer: Jon R. Voskuil

Product Name:	**SPELL 'N FIX**
Equip. req'd:	TRS-80C, with disk or cassette, 32K; other versions available for FLEX, OS-9, and other systems.
Price:	$69.29 (FLEX version $89.29)
Manufacturer:	Star Kits P.O. Box 209 Mt. Kisco, NY 10549

Description: *SPELL 'N FIX* is a package of program files that provides a dictionary for Color Computer text files. The main program, SPELLFIX, loads and executes a 6809 machine-language dictionary look-up program. A 20,000-word dictionary file is used to check ASCII files for spelling and typographical errors. Other files included are utilities for writing and reading ASCII files, a sample text file, binary-to-ASCII conversion programs, and a program to expand the dictionary. These programs allow you to use SPELLFIX with processors that create binary files.

Pluses: The dictionary program is expandable when using the disk version, and you can create your own dictionary that fits your writing style. Questionable words are displayed, and/or printed in alphabetical order for checking. The disk version also allows marking of questionable words for later correction, or they may be corrected immediately. Large files usually take only slightly longer to correct than smaller files. It will work on most files that are larger than RAM memory. The disk version can be easily converted to tape, and *vice versa*.

Minuses: The tape version cannot mark or immediately correct text files. Not directly compatible with Color Scripsit files, though, Scripsit can print an ASCII file to tape, which can be read by the dictionary.

Documentation: A 25-page manual is included that thoroughly explains the proper operation of the programs. Information is also provided on modifying and creating new dictionaries. No instructions were included foı removing words from the dictionary.

Skill level required: With only a few minutes of study, anyone should be able to operate the program.

Reviewer: John Steiner

MICRO

MICRO™

Software Catalog

Name: **Data Tape Maker**
System: OSI C1P/Superboard II
Memory: 4K
Language: 8K BASIC in ROM
Description: *Data Tape Maker* is a relatively short program that allows you to save machine-language code or any other data stored in consecutive memory locations in DATA statements on tape. The sign space for each number is eliminated to allow for compact storage of data. A FOR/NEXT loop is automatically generated to restore the data into memory at a later time.
Price: $4.00 for tape
$3.00 for listing
Author: Brian Zupke
Available:
B.C. Software
5152 Marcell Ave.
Cypress, CA 90630

Name: **Air Navigation Trainer**
System: Apple II or Apple II Plus, Applesoft in ROM or Language Card
Memory: 48K
Language: Applesoft and Machine Language
Hardware: One disk drive (DOS 3.3) and game paddles
Description: *Air Navigation Trainer* is a real-time simulation of aircraft navigation with hi-res instrumentation and ground-track map, sound effects (including station IDs), dial-in wind magnitude and direction, four different simulations, dual independent VORs (VHF Omnirange Radar) with adjustable OBS (just like the real thing), ADF, NDBs, and more.
Price: $40.00
Includes program diskette and full documentation.
(Not for pilots only!)
Author: Ken Winograd
Available:
Space-Time Associates
20-39 Country Club Drive
Manchester, NH 03102
(603) 625-1094

Name: **Spellmaster (ProofReading Software)**
System: CBM 8032, CBM 8096, SuperPET, Commodore 64
Memory: 32K minimum
Language: Assembly (6502)
Description: *Spellmaster* identifies and allows correction of misspellings from wordprocessing text. It has a 40,000-word capacity on the CBM 8050. Suspect words are displayed on screen, and direct screen editing of mistakes is provided. Available for WordPro, Wordcraft, Silicon Office. It will proofread a large WordPro file in two minutes or less. Legal and medical dictionaries are available for $75.
Price: $199.00
Author: Dwight Huff and Joe Spatafora
Available:
Spellmaster Systems Software
6219 13th Avenue South
Gulfport, FL 33707
(813) 347-6733

Name: **Rail Runner**
System: TRS-80 Color Computer or TDP System 100
Memory: 16K
Language: Assembly
Hardware: Cassette or disk
Description: Your railroad engineer must scurry over the track of the busiest train switchyard ever, dodging speeding trains and handcars, to rescue the poor little hoboes on the wrong side of the tracks. You have only so much time to save all the hoboes! With many levels of difficulty, this action graphics game is fun for everyone.
Price: $21.95 cassette
$26.95 disk
plus $2 shipping
Includes cassette or disk with instructions.
Author: BJ
Available:
Computerware
Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **K-Star Patrol™**
System: Atari 400/800
Memory: 8K
Language: Machine Code
Hardware: ROM cartridge
Description: An exciting galactic encounter between the player's patrol flight and an onslaught of attacking alien craft. The player's mission is further complicated by a voracious intergalactic leech, and the aliens' low-level avoidance system. High degree of challenge and entertainment for even the most experienced player.
Price: $39.95 suggested retail
Includes ROM cartridge and full color instruction booklet.
Author: Dr. Keith Dreyer and Torre Meeder
Available:
K-Byte
1705 Austin
Troy, MI 48084
or your local computer software retailer

Name: **Death Race '82**
System: Apple II with Applesoft in ROM
Memory: 48K
Language: BASIC/Assembler
Hardware: One disk drive, game paddles
Description: *Death Race '82* combines the skill of perilous driving with the thrill of a high-speed chase. Behind you is a robot car fully equipped with high-technology lasers. Your successful escape depends on maneuvering your turbo car through the enigmatic curves of ten consecutive mazes, and foiling your pursuer through the clever use of bazooka rockets and oil slicks. Ten different speeds ranging from novice to expert offer hours of fun before proficiency is achieved.
Price: $29.95
Includes disk and documentation.
Author: Don Fudge
Available:
Avant-Garde Creations
P.O. Box 30160
Eugene, OR 97403
or local dealers

Name: **Single Entry Ledger**
System: 6809 Using FLEX or UniFLEX, TRS-80 Model III and Color Computer
Memory: 56K
Language: Extended BASIC
Hardware: 8" or 5¼" disk
Description: *Single Entry Ledger* is a simple bookkeeping system for tracking income and expenses. It is an ideal accounting system for tax purposes saving the user both time and money. The data files may contain any number of accounts or transactions. Any number of reports may also be written from comparison reports of the previous year to transactions by account number.
Price: $95.00
Includes disk and manual.
Author: K. Orlowski
Available:
Universal Data Research Inc.
Dept. A
2457 Wehrle Drive
Buffalo, NY 14221

Name: **Prelab Studies in General Organic and Biological Chemistry**
System: Apple II with 3.3 DOS
Memory: 48K
Language: Applesoft
Description: This package provides a review of selected chemical concepts highlighting important ideas, techniques, and calculations encountered in the laboratory. The programs are in a tutorial format, using demonstrations, interactive exercises, animated sequences, and simulations.
Price: $550.00 (tentative)
Includes nine disks and complete documentation.
Author: Sandra L. Olmsted and Richard D. Olmsted
Available:
John Wiley & Sons, Inc.
Eastern Distribution Center
Order Processing Department
1 Wiley Drive
Somerset, NJ 08873

Name: **System/ASM 3A**
System: Apple II Plus
Memory: 48K minimum. Language card is supported.
Language: Assembly
Hardware: Disk II required, Silentype printer optional
Description: *System/ASM 3A* is an assembly-language development system that features a two-pass assembler, full screen editor, and disk-file management system. The system is easy to use but powerful enough to write very complex programs. *System/ASM 3A* is written in its own assembly language and is DOS 3.3-compatible.
Price: $35.00
$5.00 for manual only
Includes no shipping and handling charges. Ohio residents add appropriate sales tax.
Available:
The Mike Piaser Company
15401 Maple Park Drive #11
Maple Heights, OH 44137

Name: **Factoring Whole Numbers**
System: PET DOS 2.1
Memory: 16K
Language: BASIC
Hardware: Disk drive or cassette
Description: Twelve programs (on six tapes or three diskettes) present the concepts of factoring in a carefully-designed sequential preparation for fractions and algebraic expressions. A tutorial and practice program precedes six motivating and interactive enrichment programs.
Price: $90.00
Includes diskettes or tapes and a teacher's guide.
Author: Joanne Benton
Available:
Quality Educational Designs
P.O. Box 12486
Portland, OR 97212

Name: **Android Attack**
System: Atari 400/800
Memory: 16K cassette
32K disk
Language: Hybrid
Hardware: Cassette or disk system
Description: The nuclear reactor in our top-secret underground lab is in danger of melting down! Only you can save it by manually releasing the coolant water. Unfortunately, there isn't time to disarm the security Androids guarding the installation, so you'll have to fight your way down. Once you've released the water, you've got to get back out before you drown! *Android Attack* has electric robots and walls, bonus points, and up to eight different levels to challenge you!
Price: $18.95 plus $2 shipping (Mail order price)
Author: John Wilson
Available:
Pretzelland Software
2005 D. Whittaker Rd.
Ypsilanti, MI 48197
(313) 483-7358
or local dealers

Name: **The Last One**
System: Apple II Plus
Memory: 48K
Language: BASIC/Machine
Hardware: Two disk drives, printer optional
Description: *The Last One* is a computer program code generator that designs a program and enters flowchart-type statements in an easy-to-use menu style. *The Last One* then begins to code the program, asking the user questions about "where to branch," etc. A BASIC program is created as output which then can be run, listed, or modified like any other BASIC program. *The Last One* is not required to execute the output program.
Price: $600.00
Includes complete documentation, numerous sample flowcharts that will produce software worth several hundred dollars.
Author: D.J. 'AI' Systems Ltd.
Available:
Krown Computing
1282 Conference Dr.
Scotts Valley, CA 95066
(408) 335-3133

Name: **Assemblers Package I**
System: The UCSD p-System™
Memory: 48Kb runtime environment; 64Kb development environment
Language: Assembly
Hardware: 8086, Z80, 8080, 8085, 6502, 9900, 6809, 68000, and LSI-11/PDP-11
Description: This collection of native code-generating macro cross-assemblers allows you to program on the host machine of your choice for the object machine of your choice.
Price: $375.00
Includes object code.
Available:
SofTech Microsystems, Inc.
9494 Black Mountain Rd.
San Diego, CA 92126
(714) 578-6105

Name: **Galactic Gladiators**
System: Apple II with Applesoft ROM card, Apple II Plus, or Apple III
Memory: 48K
Hardware: Monitor and disk drive
Description: *Galactic Gladiators* is a fast and furious computer game of alien combat for two players or against the computer. The creatures are rated for strength, endurance, speed, dexterity, experience, weapons, skill, and armor. The scenario permutations are as infinite as the Universe.
Price: $39.95
Includes rulebook, disk, and data card.
Author: Tom Reamy
Available:
Strategic Simulations Inc.
465 Fairchild Dr.
Suite 108
Mountain View, CA 94043
(415) 964-1353

Name: **The Animator**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft/Assembly
Hardware: Disk drive
Description: This program produces animated 'film' strips that enter only key frames, then *The Animator* calculates the in-between frames. The key frames are easily entered — either visually, numerically, or from a library. The demo includes a ballet sequence showing a ballerina with 12 independently moving body parts.
Price: $51.95
Includes 57-page manual, three tutorials, and a shape generator.
Author: Ray Balbes
Available:
Balbesoftware Systems
#6 White Plains Dr.
St. Louis, MO 63017
(314) 532-5377

Name: **The Apple Family Sing-Along Christmas Disk**
System: Apple II, Apple II Plus, Apple III
Memory: 48K
Language: Applesoft or Integer Basic (runs in emulation mode on Apple III)
Hardware: Disk drive
Description: Sixteen favorite carols, complete with words to all the verses, containing multiple-voices and four-part harmony, are pitched so you can sing along if you want to. The choice of an internal speaker or cassette port output is given. The Christmas music is tuneful, well arranged, and lots of fun to listen to. Just the thing to lend novelty and a festive background to Christmas parties, office parties, and Apple family get-togethers.
Price: $24.50
Includes everything needed to play the songs — no hardware required.
Author: Product of the *Music Maker*™ utility from SubLogic Communications Corp.
Available:
Solutions Softworks
Box 72280
Roselle, IL 60172
$1.50 shipping costs
or from Apple dealers

Name: **Anova II**
System: Apple II or Apple II Plus
Memory: 48K
Language: ROM Applesoft
Hardware: One or two disk drives, printer optional
Description: *Anova II* performs up to a five-way analysis of variance with equal or unequal numbers. It can analyze randomized designs, between and within designs, and repeated measures of designs. *Anova II* can also perform an analysis of co-variance for all designs. The *Anova* table output tests all factors and interactions.
Price: $150.00
Includes program disk and backup disk, documentation, and binder.
Authors: Stephen Madigan, Ph.D. and Virginia Lawrence, Ph.D.
Available:
Human Systems Dynamics
9249 Reseda Blvd.
Suite 107
Northridge, CA 91324

(continued)

Name: **UniFLEX**
System: Gimix 6809 Winchester Systems
Memory: 128K minimum
Language: Available: BASIC, Pascal, Assembler, FORTRAN 77, C
Hardware: 2MHZ 6809 CPU with memory, disk controllers, 19MB 5¼" Winchester
Description: *UniFLEX* is a true multi-tasking, multi-user operating system. Each user communicates with the system through a terminal and may execute any of the available system programs. This implies that one user may be running the text editor while another is running BASIC while still another is running the C compiler. Not only may different users run different programs simultaneously, but one user may be running several programs at a time.
Price: $550.00
Includes UniFLEX Operating System, documentation.
Author: Technical Systems Consultants, Inc.
Available:
Gimix Inc.
1337 W. 37th St.
Chicago, IL 60609
(312) 927-5510

Price: $99.95/Sinclair tape
$129.95/Apple/Atari disk
$129.95/Atari tape
Includes 34 pages of documentation.
Author: Bob Nadler
Available:
F/22 Press
P.O. Box 141
Leonia, NJ 07605

Name: **Lovers or Strangers**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: One disk drive
Description: *Lovers or Strangers* is a computer game with a serious side. It is a compatiblity evaluator that tells two people how likely they are to have a successful relationship. A couple's likes and dislikes, philosophies, and lifestyles in seven major areas of compatibility are explored.
Price: $29.95
Includes program disk and written instructions.
Author: Stanley Crane
Available:
Alpine Software, Inc.
2120 Academy Circle, Suite E
Colorado Springs, CO 80909
(303) 591-9874

Name: **The Football Comput-Stat**
System: Apple II, IBM PC, Radio Shack MIII
Memory: 48K
Language: BASIC
Hardware: One disk drive, printer optional
Description: *Compu-Stat* contains programs and related data for the analysis of pro-football's regular season — both point-spread records and the underlying box-score statistics. It performs analyses for the 1981 and 1982 regular seasons. A related program product, Tally Sheet, keeps a running tally on your predictions.
Price: $100 - $3500 depending on programs and equipment ordered.
Includes user manual, program diskette, and security chip.
Author: Dr. John Page
Available:
Interactive Sports Systems
P.O. Box 15952
New Orleans, LA 70175

Name: **Elements of Mathematics**
System: Apple II
Memory: 48K
Language: BASIC
Hardware: One disk drive
Description: This program was developed to assist students in adding fractions, reducing fractions, and adding fractions with unlike denominators. Materials were developed and tested by the authors before being published.
Price: $90.00
Author: Ray E. Zubler
Susan Sarapata
Available:
Electronic Courseware Systems, Inc.
P.O. Box 2374, Station A
Champaign, IL 61820
(217) 359-7099
or computer retail stores and book stores

(continued)

Software Catalog *(continued)*

Name: **Basic Aid**
System: TRS-80 Color Computer
Memory: 16K - 64K
Language: 6809 Machine Language
Hardware: ROMPAK
Description: *Basic Aid* is a utility program to help and assist Color BASIC and Extended BASIC users. Some of the features are: automatic line numbering, program merging, and moving program segments. It comes with a plastic keyboard overlay that contains most of Extended Color BASIC's commands.
Price: $34.95
Includes detailed instruction manual, plastic keyboard overlay.
Author: Eigen Systems
Available:
Spectrum Projects
93-1586 Drive
Woodhaven, NY 11421

Name: **S-C Macro Cross Assemblers 6800, 6809, and Z-80**
System: Apple II or Apple II Plus
Memory: 48K (RAM card version included)
Language: Machine
Hardware: Disk drive
Description: You can easily develop programs for 6800, 6809, or Z-80 computers with powerful macros, conditional assembly, 20 directives, and 29 commands (including a powerful EDIT command with 15 subcommands). It allows very fast cycles of modification, reassembly, and testing.
Price: $110.00 each.
Registered owners of the S-C Macro Assembler pay $32.50 each.
Includes diskette with regular and RAM card versions, 100+-page manual.
Available:
S-C Software Corporation
P.O. Box 280300
2331 Gus Thomasson
Suite 125
Dallas, TX 75228
(214) 324-2050

Name: **GL-PLUS**
System: Apple III
Memory: 128K
Language: Business BASIC
Hardware: 132-column printer and either second diskette drive or hard drive.
Description: *GL-PLUS* is an extremely flexible and easy to operate general ledger with built in receivables and payables. Reports include general ledger, month's journal, balance sheet, income statement, aged receivables and payables, receivable and payable detail, and more!
Price: $495.00
Includes operator's manual, programs, and sample company data.
Author: Dan Sargent
Available:
Great Divide Software
8060 W. Woodard Dr.
Lakewood, CO 80227

Name: **Borg**
System: Apple II or Apple II Plus
Memory: 48K
Language: Assembly
Hardware: One disk drive, paddle or joystick
Description: Deranged Grud Terrorizes Countryside! Protected by Borg, the invincible Drageroo, a notorious band of dragons, the infamous Grud has surrounded his hide-out with electrified mazes. Can no one crack the code and rid us of this menace?
Price: $29.95
Author: Dan Thompson
Available:
Sirius Software, Inc.
10364 Rockingham Dr.
Sacramento, CA 95827
(916) 366-1195

Name: **D.F.T**
System: TRS-80 Color Computer
Memory: 16K
Language: Machine
Hardware: Cassette recorder
Description: This terminal program allows you to download any type of program — BASIC or machine language — or ASCII with no conversion. It allows transfer of programs between TRS-80 Mod I's, Mod III's, and the Color Computer.
Price: $19.95
Includes one tape.
Author: Bob Withers
Available:
Computer Shack
1691 Eason
Pontiac, MI 48054

Correction: The software listing for Jinsam Executive (52:116) from JINI Micro-Systems, Inc., should have read 32K for CBM w/8050, and 128K IBM PC for BASIC and machine language. It is available from the company and participating dealers.

MICRO

MICRO™

Hardware Catalog

Name: **Guild Computer Rack**
System: Apple II
Description: The *Guild Rack* comes in a choice of beautifully finished mahogany or ash. No assembly is required. It fits comfortably over the Apple II keyboard, holds one or two disk drives, and easily supports a monitor on top.
Price: $54.95 - ash
$69.95 - mahogany
Available:
Guild Computer Rack
225 West Grand Street
Elizabeth, NJ 07202
(201) 351-3002

Name: **Disk Interface/ ROMpak Extender**
System: Color Computer
Memory: 4K and up
Hardware: Three-foot extender cable
Description: The *Disk Interface/ROMpak Extender* is a 40-pin ribbon cable that plugs into the ROMpak port and terminates three feet later with a 40-pin female connector to connect ROMpaks and the disk interface. Gold-plated contacts eliminate corrosion.
Price: $29.95 plus $1 for S/H
Includes male and female connector, three feet of 40-conductor cable.
Available:
Spectrum Projects
93 - 1586 Drive
Woodhaven, NY 11421
(212) 441-2807 Voice
(212) 441-3755 Computer

Name: **Versaclock**
System: TRS-80 Color Computer
Memory: 4K and up
Language: BASIC or Extended BASIC
Description: The *Versaclock* is a full-featured, highly accurate hardware clock for the Color Computer. It provides time of day, date, month, and year with automatic daylight savings time and leap year compensation. The clock is battery backed-up to allow removal from computer without loss of data. The clock also contains 50 bytes of battery backed-up RAM for general purpose permanent storage. The many software options include interrup handling and 12/24 hour formats.
Price: $99.95
Includes Versaclock cartridge, full instructions.
Available:
Maple Leaf Systems
Box 2190
Station "C", Downsview
Ontario, Canada M2N-2S9

Name: **Color Graphic Printer (26-1192)**
System: Compatible with TRS-80 Models I, II, III, and Model 16 computers, and DT-I Data Terminal
Description: The TRS-80 *Color Graphic Printer* can create anything from doodles to four-color pie charts, as well as more standard text and graphcis. Ninety-six ASCII characters are available in four colors (red, blue, green, black). Special graphic commands include backspace, reverse line feed, change colors, change line type (solid or 15 types of dashed lines), change print direction (normal left-to-right, top-to-bottom, upside down or bottom-to-top), move without drawing, draw between points and draw axes. The RS232-C serial interface is compatible with Radio Shack TRS-80 Color Computers.
Price: $249.95
Available:
Radio Shack Stores, computer centers, and participating dealers

---

Name: **K-Byte Stick Stand with Fastball Easy-Grip Control Knob.**
Description: K-Bytes unique *Stick Stand with the Fastball Easy-Grip Control Knob* reduces hand and wrist fatigue and frees one hand for a more skillful operation of the firebutton. This combination allows players to increase their physical dexterity and achieve higher scores. By just snapping the fastball onto the joystick and then snapping the joystick into the stick stand, the player is all set for precision arcade action.
Price: $6.99 suggested retail
Includes base stand and fastball knob.
Available:
John Mathias
K-Byte™
Div. of Kay Enterprises Co.
P.O. Box 456
1705 Austin
Troy, MI 48099
(313) 524-9878
or your local computer retailer

Name: **Fast Load — Fast Save Cassette System**
System: OSI - C1P or Superboard II
Description: Load BASIC or machine-language programs in your 8K memory in less than 30 seconds at a speed of 2400 bits per second input/output data rate. Customer supplies own tape recorder. The unit includes a 2K RAM fully decoded which may be used to hold machine-language programs. Unit plugs directly into your C1P or Superboard II.
Price: $69.95 fully assembled
$59.95 with cashier's check or money order.
$62.95 kit
$52.95 with cashier's check or money order.
Includes printed circuit board, cassette tape program, self-contained R/W memory, connectors, and user's manual.
Available:
Word-Com
P.O. Box 1122 - 28
Park Plaza Offices
303 Williams Ave.
Huntsville, AL 35801

Name: **Pro-Guard 8" Floppy Controller**
System: Apple III
Memory: Up to 2.2 megabytes
Language: SOS, DOS 3.3, Pascal
Hardware: Controls two 8" Shugart-compatible drives
Description: This 8" floppy controller resides in-line between Apple III and the drive system and connects to slot 2 *via* SVA's innovative *Smart-Cable.*
Price: $695.00
Available:
SVA Sorrento Valley Associates, Inc.
11722 Sorrento Valley Rd.
San Diego, CA 92121
Apple dealers, Micro-D, Micro House, U.S. Micro Sales

Name: **Ramex 128**
System: Apple II or Apple II Plus
Memory: 48K
Description: This 128K RAM expansion board includes disk-emulation software that features super-fast mounts and dumps from card to disk (20-25 seconds for an entire 128K). Also available for VisiCalc is super expander software that gives the same super-fast loading and saves of VisiCalc files (136K in 20 seconds).
Price: $499.00
Includes disk emulation software and memory management.
Available:
Omega Microware, Inc.
222 S. Riverside Plaza
Chicago, IL 60606

Name: **Multi-Port 232**
Description: *The Multi-Port 232* is a 4- or 8-port multidrop data router that allows merging or splitting of RS232, fiber optic, and current loop in any source/destination combination. It provides local networking for word processors, printers, modems, video displays, computers, teletypes, and instruments.
Price: $435.00 - 4-port
VISA/Master Charge
Includes nine user-selectable preprogrammed routes.
Available:
Park Computer Corporation
Box 13010
Minneapolis, MN 55414

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

86. Color Computer News, Issue No. 11 (August, 1982)

Ostrom, Steven M., "Graphics and Animation for the Color Computer," pg. 30-42.
A tutorial for the TRS-80 Color Computer graphics with a number of demo routines.

Dawson, Don, "Color Yahtzee," pg. 44-47.
A game for the 6809-based Color Computer.

Phelps, Andrew, "Comment Corner," pg. 49-50.
A tutorial on RAM hooks, places where the program jumps, and which then jump elsewhere in memory.

McClenahan, Shawn A., "A Real Keyboard for the Color Computer," pg. 55-60.
A hardware project for the Color Computer.

Field, E.C., "Electro-Sketch," pg. 67-69.
A graphics program for the 6809-based Color Computer which allows one to draw simple schematics and save or print them.

Lee, Paul, "Educating Your Preschooler with the Color Computer," pg. 71.
A simple teaching program for young children using the Color Computer.

Weiss, Arnold, "Cryptogram," pg. 72-76.
A program to present cryptograms on the TRS-80 Color Computer screen or to make printed copies.

Harper, Jeff, "Word Processor," pg. 77-79.
A word-processor program for the 16K or 32K Extended BASIC Color Computer.

Foster, Robert D., "Monitor," pg. 81-82.
A simple monitor to allow one to see how the Color Computer actually works.

Tenny, Ralph, "Extra Tricks with Color Scripsit," pg. 84-85.
An accessory listing to aid in using Color Scripsit.

Aldrich, F.C., "Magic Square," pg. 87-89.
A contest-winning listing for the 6809-based Color Computer.

87. '68' Micro Journal, 4, Issue 8 (August, 1982)

Anderson, Ronald W., "FLEX User Notes," pg. 11-14.
Miscellaneous notes on FLEX for the 6809-based systems. Includes a multiply program in assembly language.

Nay, Robert L., "COLOR User Notes," pg. 14-16.
Discussion of some new items for the 6809-based Color Computer.

Abrams, Clayton W., "F-Mate," pg. 16-17.
A utilities package for the TRS-80 Color Computer.

Distefano, Tony, "Color Clinic," pg. 17-18.
Discussion of hardware modifications for the TRS-80 Color Computer.

Commo, Norm, " 'C' User Notes," pg. 19-24.
Discussion of major C compilers for 6809 systems.

Watson, Ernest Steve, "Home Accounting Program — Part II," pg. 25-28.
A program for 6809 systems.

Hartman, William, "Diskfix9," pg. 29-36.
A utility for 6809-based systems.

88. The Rainbow, 2, No. 2 (August, 1982)

Nolan, Bill, "Let's Call JOYIN To Learn ROM Call Technique," pg. 8.
A short program illustrating how to call one of the built-in ROM routines in the TRS-80 Color Computer.

Lishnak, Pat, "Sort Numeric Arrays Fast with Machine Language," pg. 9.
A bubble sort technique for the Color Computer.

Boston, William, "Here's An Easy Way to Place Orders by Mail," pg. 19-20.
An order-writing program for the Color Computer.

Lewandowski, Dennis S., "The Assembly Corner," pg. 22-25.
A tutorial on 6809 assembly-language programming.

Clements, Bill, "Rockin' Through the ROM," pg. 29-30.
Documenting the ROM routines of the TRS-80 Color Computer.

Preble, Laurence D., "FLEX System is Powerful Addition to World of 80C," pg. 32-33.
All about the FLEX system for the 6809 micro.

Scerbo, Fred B., "Alpine Aliens," pg. 34-37.
A game for the Color Computer.

Blyn, Steve, "Good Reinforcement Means You Can't Frown at Me!", pg. 41-46.
Tips and demo program for educational use.

Mir, Jorge, "Now, Make Your Own Adventure with ADVMAKER," pg. 47-53.
A program designed to simplify the programming of Adventures written for the Color Computer.

Nolan, Bill, "Dragons Are Nice Folks, Too... Almost All 1,440 of Them," pg. 62-69.
The program "Dragon Roller" will assist with the chore of devising a dragon for your dungeon program.

89. Byte, 7, No. 8 (August, 1982)

Williams, Gregg, "LOGO for the Apple II, the TI-99/4A, and the TRS-80 Color Computer," pg. 230-290.
Discussion of LOGO for several micros, including the 6809-based Color Computer.

90. The Target (March/April, 1982)

Staff, "News", pg. 1.
An assembly which converts an AIM 65 into a 6809-based computer.

91. Compute! 4, No. 8 (August, 1982)

Chastain, Linton S., "Energy Monitor," pg. 116-118.
This program for the TRS-80 Color Computer will show you the effects of home energy conservation.

92. Microcomputing, 6, No. 9 (September, 1982)

Avery, Mike, "Prime Number Nonsense," pg. 16.
Comments on the 6809 *versus* the 6502, Z-80, or 6800 microprocessors.

93. Color Computer News (August, 1982)

Sias, Bill, "REMarks," pg. 6-7.
Announcement of the 6809 Achievement Award being given monthly to the most innovative use for a 6809.

Gray, Don, "Number Conversion," pg. 7-8.
Three listings for number conversion programs for the 6809-based TRS-80 Color Computer.

Anon, "Color Computer Bulletin Board System," pg. 11.
A BBS for the Color Computer is up in the Toronto area. Call (416) 494-7001 evenings and weekends.

Donahue, Mike, "Cross-Reference Generator," pg. 15-25.
A utility for the 6809-based TRS-80 Color Computer.

Grady, Larry, "Review of Master Control," pg. 29-33.
Problems encountered with the program "Master Control" and some reprogrammed sections to alleviate difficulties.

Graham, Randy W., "Modems, Terminals, and Bulletin Boards," pg. 35-38.
Using the Color Computer in telecommunications.

COMMODORE 64

The Commodore 64 is a 6510-based color-and-sound computer that connects to a color TV via an RF modulator. 64K RAM is standard, with 39K of it available for BASIC programs.

- Graphics
  - 3 character modes
  - 2 bit-map modes
  - sprite graphics
- Sound
  - 4 programmable voices
  - attack, sustain, decay, and release
  - output compatible with stereos

Z-80 option for CP/M
RS-232, expansion/cartridge, parallel, cassette and controller interfaces

Commodore 64 Memory Map

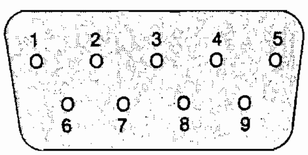
Address	Function
$00-$FF	Page zero, operating system storage, pointers, floating point accumulators, flags, etc.
$100-$1FF	Microprocessor system stack
$100-$10A	Floating-to-string work area
$100-$13E	Tape input error log
$200-$2FF	Operating system buffers, tables, vectors, I/O flags, keyboard handling
$300-$3FF	Vectors, tape I/O
$400-$7FF	Normally video memory, sprite data pointers, etc.
$800-$9FFF	Normally BASIC program space
$8000-$9FFF	VSP Cartridge ROM
$A000-$BFFF	BASIC ROM
$C000-$CFFF	RAM
$D000-$DFFF	I/O devices and color RAM or character-generator ROM
$E000-$FFFF	Kernal ROM

Control Port 1

Pin	Function
1	JOYA0
2	JOYA1
3	JOYA2
4	JOYA3
5	POT AY
6	BUTTON A/LP
7	+5V
8	GND
9	POT AX

Control Port 2

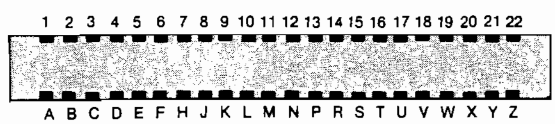
Pin	Function
1	JOYB0
2	JOYB1
3	JOYB2
4	JOYB3
5	POT BY
6	BUTTON B
7	+5V
8	GND
9	POT BX



Cartridge Expansion Slot

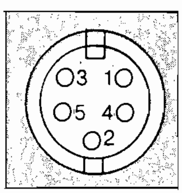
Pin	Function
1	GND
2	+5V
3	+5V
4	IRQ
5	CR/W
6	Dot Clock
7	I/O1
8	GAME
9	EXROM
10	+I/O2
11	ROML
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

Pin	Function
A	GND
B	ROMH
C	RESET
D	NMI
E	S02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND



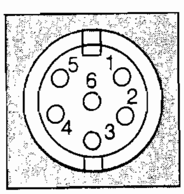
Audio/Video

Pin	Function
1	LUMINANCE
2	GND
3	AUDIO OUT
4	VIDEO OUT
5	AUDIO IN



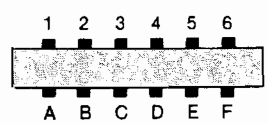
Serial I/O

Pin	Function
1	SERIAL SRQIN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET



Cassette

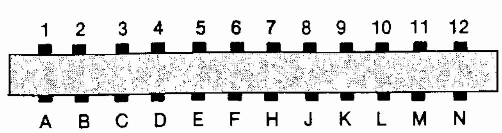
Pin	Function
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE



User I/O

Pin	Function
1	GND
2	+5V
3	RESET
4	CNT1
5	SP1
6	CNT2
7	SP2
8	PC2
9	SER. ATN IN
10	9 VAC
11	9 VAC
12	GND

Pin	Function
A	GND
B	FLAG2
C	PB0
D	PB1
E	PB2
F	PB3
H	PB4
J	PB5
K	PB6
L	PB7
M	PA2
N	GND



MOS 6510 I/O Registers

Hex	Decimal	Bits	Function
0000	0	7-0	MOS 6510 Data Direction Register (xx101111)
			Bit = 1: output Bit = 0: input x = don't care
0001	1		MOS 6510 on-chip I/O port
		0	/LORAM signal (0 = switch BASIC ROM out)
		1	/HIRAM signal (0 = switch kernal ROM out)
		2	/CHAREN signal (0 = switch char. ROM in)
		3	Cassette data output line
		4	Cassette switch sense (1 = switch closed)
		5	Cassette motor control (0 = on 1 = off)
		6-7	Undefined

MOS 6566 Video Interface Controller (VIC II)

Hex	Decimal	Bits	Function
D000	53248		Sprite 0 — X Pos
D001	53249		Sprite 0 — Y Pos
D002	53250		Sprite 1 — X Pos
D003	53251		Sprite 1 — Y Pos
D004	53252		Sprite 2 — X Pos
D005	53253		Sprite 2 — Y Pos
D006	53254		Sprite 3 — X Pos
D007	53255		Sprite 3 — Y Pos
D008	53256		Sprite 4 — X Pos
D009	53257		Sprite 4 — Y Pos
D00A	53258		Sprite 5 — X Pos
D00B	53259		Sprite 5 — Y Pos
D00C	53260		Sprite 6 — X Pos
D00D	53261		Sprite 6 — Y Pos
D00E	53262		Sprite 7 — X Pos
D00F	53263		Sprite 7 — Y Pos
D010	53264		Sprites 0-7 X Pos (msb of X coord.)
D011	53265		VIC Control Register
		7	Raster compare: (bit 8) See 53266
		6	Extended color text mode: 1 = enable
		5	Bit-map mode: 1 = enable
		4	Blank screen to border color: 0 = blank
		3	Select 24/25-row text display: 1 = 25 rows
		2-0	Smooth scroll to Y dot-position (0-7)
D012	53266		Read/write raster value for compare IRQ
D013	53267		Light pen latch — X Pos
D014	53268		Light pen latch — Y Pos
D015	53269		Sprite display enable: 1 = enable
D016	53270		VIC Control Register
		7-6	Unused
		5	ALWAYS SET THIS BIT TO 0!
		4	Multicolor mode: 1 = enable (text or bit-map)
		3	Select 38/40-column text display: 1 = 40-columns
		2-0	Smooth scroll to X dot position (0-7)
D017	53271		Sprites 0-7 expand 2x horizontal (Y)
D018	53272		VIC Memory Control Register
		7-4	Video matrix base address (inside VIC)
		3-0	Character dot-data base address
			(Bit = 1: IRQ occurred)
D019	53273		VIC Interrupt Flag Register (bit = 1:IRQ occurred)
		7	Set on any enabled VIC IRQ condition
		3	Light pen-triggered IRQ flag
		2	Sprite-to-sprite collision IRQ flag
		1	Sprite-to-background collision IRQ flag
		0	Raster compare IRQ flag
D01A	53274		IRQ mask register: 1 = interrupt enabled
D01B	53275		Sprite-to-background display priority (1 = sprite)
D01C	53276		Sprites 0-7 multicolor mode select (1 = MCM)
D01D	53277		Sprites 0-7 expand 2x vertical (Y)
D01E	53278		Sprite-to-sprite collision detect
D01F	53279		Sprite-to-background collision detect
D020	53280		Border color
D021	53281		Background color 0
D022	53282		Background color 1
D023	53283		Background color 2
D024	53284		Background color 3
D025	53285		Sprite multicolor register 0
D026	53286		Sprite multicolor register 1
D027	53287		Sprite 0 color
D028	53288		Sprite 1 color
D029	53289		Sprite 2 color
D02A	53290		Sprite 3 color
D02B	53291		Sprite 4 color
D02C	53292		Sprite 5 color
D02D	53293		Sprite 6 color
D02E	53294		Sprite 7 color

MOS 6581 Sound Interface Device (SID)

Hex	Decimal	Bits	Function
D400	54272		Voice 1: Frequency control — low byte
D401	54273		Voice 1: Frequency control — high byte
D402	54274		Voice 1: Pulse waveform width — low byte
D403	54275	7-4	Unused
		3-0	Voice 1: Pulse waveform width — high nibble
D404	54276		Voice 1: Control register
		7	Select random noise waveform 1 = on
		6	Select pulse waveform 1 = on
		5	Select sawtooth waveform 1 = on
		4	Select triangle waveform 1 = on
		3	Test bit: 1 = disable oscillator 1
		2	Ring modulate osc. 1 with osc. 3 output 1 = on
		1	Synchronize osc. 1 with osc. 3 freq. 1 = on
		0	Gate bit: 1 = start att/dec/sus 0 = start release
D405	54277		Envelope generator 1: attack/decay cycle ctrl.
		7-4	Select attack cycle duration: 0-15
		3-0	Select decay cycle duration: 0-15

COMMODORE 64

Advertiser's Index

Aardvark Technical Services, Ltd. 76
ABM Products 24
Amplify Inc. 62
Anthro-Digital Software 17
Apex Co. 24
Appletree Electronics 51
Ark Computing 12
Artsci, Inc. .... IFC
Aurora Software Associates 83
Bedford Micro Systems 31
CGRS Microtech 63
Cleveland Consumer Computer Components 80
Collegiate Microcomputer 67
Commander 62
Compu $ense 49
CompuTech 28
Computer Mail Order 56-57
Computer Science Engineering 89
Computer Trader 99
Datamost, Inc. .... 34, 90, 92, IBC
Decision Systems 67
Digicom Engineering, Inc. 96
Digital Acoustics 84
D&N Micro Products, Inc. 21
Eastern House Software 39
Educational Computing Systems 10
Execom Corp. 40
Gimix, Inc. 1
Gooth Software 51
Hayden Software 36
Hudson Digital Electronics Inc. 68
Human Systems Dynamic 41
Interesting Software 95
Leading Edge .... BC
Logical Devices 99
Lyco Computer 6
MICRObits (Classifieds) 53, 86, 87
MICRO INK 25, 31, 41, 44, 45
Micro Motion 28
Micro Signal 50
Micro-Spec, Ltd. 40
Micro-Ware Distributing Inc. 96
Midnight Software 49
MMS, Inc. 25
Modular Systems 83
Orion Software 18
Perry Peripherals 73
Privac, Inc. 2
Pterodactyl Software 105
Quentin Research 29
SGC 4
SJB Distributing 64
Skyles Electric Works 46, 58
Softel 72
Software Farm 33
Software Options 31
Southwestern Data Systems 106
Spectrum Systems 83
Spies Laboratories 43
Star Micronics 8
Tau Lambda 111
Unique Data Systems 102
Unique Software 96
Universal Data Research 111
Victory Software 20
XPS, Inc. 101

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

Next Month in MICRO

January: Simulations/Applications/Math

- **Apple Math Editor —** This Apple Pascal program allows you to construct, edit, and print mathematical formulas easily.

- **Sun and Moon —** This Applesoft program produces a high-resolution graphic simulation of the apparent orbits of the sun and moon with respect to the Earth.

- **Measurement of a 35mm Focal Plane Shutter —** The program SHUTTER uses inexpensive hardware to measure the accuracy and repeatability of a focal plane shutter commonly found in 35mm cameras. Although written for the Atari 800, the program can be modified for any computer if you have access to three input pins, a ground, and the +5V power supply.

- **Methods to Evaluate Complex Roots —** A standard procedure to compute complex roots of polynomial equation.

- **Discrete Event Simulation on the Apple —** An explanation of techniques used in simulating real-world situations on a computer. An example program involving the flow of bank customers is presented.

Department Highlights

Apple Slices
PET Vet
From Here to Atari
CoCo Bits
Reviews in Brief
Software and Hardware Catalogs

Plus...

VIC Hi-Res Graphics Explained
Dealing with Atari's New Languages
Microcomputer Design of Transistor Amplifiers
More 68000 Instruction Set Tables