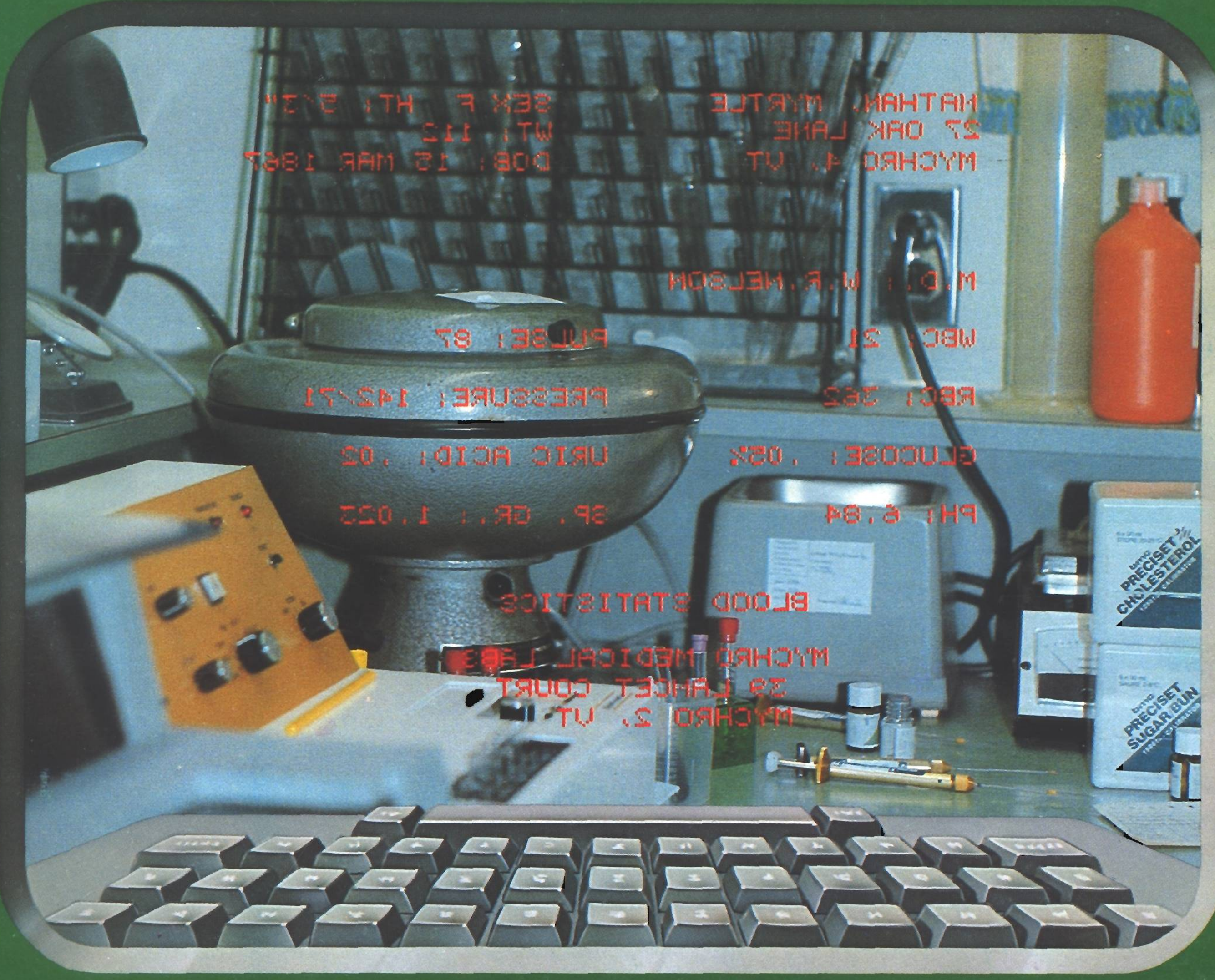
NO. 34 $2.00 MARCH 1981

MICRO™

THE 6502 JOURNAL



Rapid Bubble Sort of Numerical Elements Using BASIC/ASL

UnwrApple

A Second Cassette for PET

A Relocating Loader for AIM Tape

SYM-ple Sym-on

The 6502 Dream Machine

The Perfect Fit

The Micromodem II data communications system and the Apple II* computer. What better combination to maximize the capabilities of your personal computer!

This popular direct connect modem can transmit data between an Apple II and another Apple II, a terminal, another microcomputer, minicomputer or even a large time-sharing computer anywhere in North America. The Micromodem II has unique automatic dialing and answer capabilities which further increases the communications possibilities between the Apple II and another computer or terminal.

You can send and/or receive messages or data when you are out of your office, home or out of town. Your branch business locations can communicate with each other regarding inventory and other matters over the phone. Or you can communicate with friends across the country. And you can access information utilities like the SOURCE for various business and personal applications.

The Micromodem II consists of two parts. One part includes the printed circuit board which holds the Micromodem II, ROM firmware and the serial interface. The board plugs directly into the Apple II providing all the functions of a serial interface card plus programmable auto dialing and auto answer capabilities. The on-board ROM firmware enables the Micromodem II to operate in any of three modes to perform different tasks-terminal mode, remote console and program control mode.

The other part of the Micromodem II datacomm system is a Microcoupler which connects the Micromodem board and Apple II to a telephone line. The Microcoupler gets a dial tone, dials numbers, answers the phone and hangs up when a transmission is over. There are none of the losses or distortions associated with acoustic couplers. The Microcoupler is compatible with any North American standard telephone lines and is FCC-approved for direct connection in the U.S. It works with standard dial phone service or Touch-tone service.

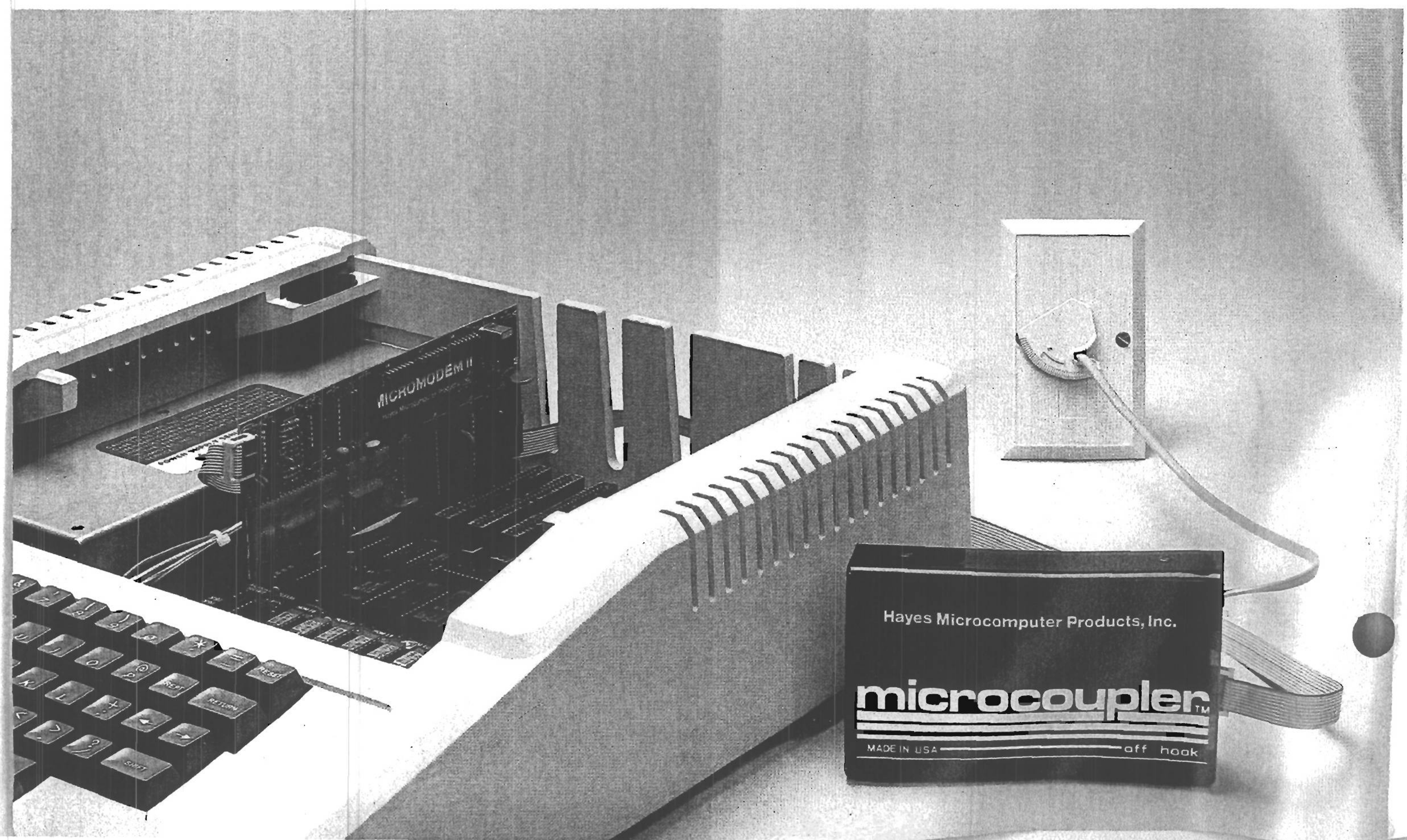
The Micromodem II is completely compatible with Bell 103-type modems. Full and half-duplex operating modes are available as well as speed selectable transmission rates of 110 and 300 bps.

Why not increase your Apple II's capabilities by outfitting it with the sophisticated Micromodem II data communications system? The Micromodem II is available at retail computer stores nationwide. For the store nearest you, call or write:

Hayes™

Hayes Microcomputer Products Inc.
5835 Peachtree Corners East, Norcross, Georgia 30092 (404) 449-8791

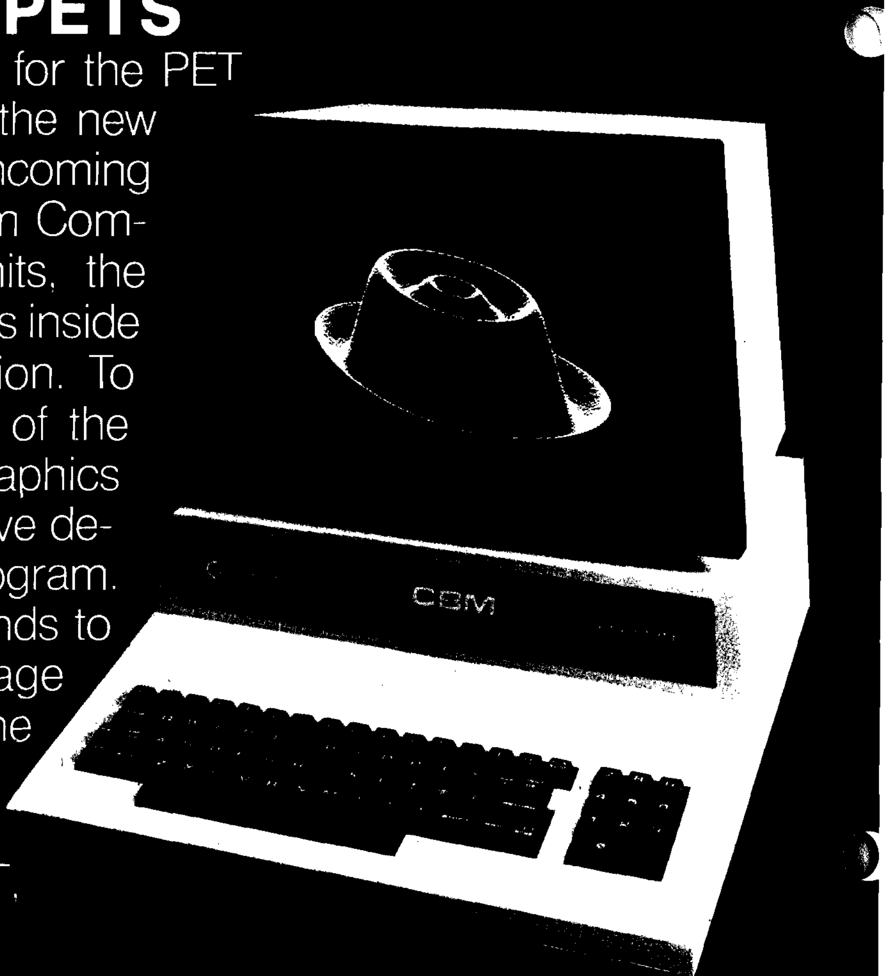
™ Micromodem II is a trademark of Hayes Microcomputer Products, Inc.
*Apple II is a registered trademark of Apple Computer Inc.
The Micromodem II can also be used with the Bell & Howell computer.



GRAPHICS

FOR 80 COLUMN PETS

The Integrated Visible Memory for the PET has now been redesigned for the new 12" screen 80 column and forthcoming 40 column PET computers from Commodore. Like earlier MTU units, the new K-1008-43 package mounts inside the PET case for total protection. To make the power and flexibility of the 320 by 200 bit mapped pixel graphics display easily accessible, we have designed the Keyword Graphic Program. This adds 45 graphics commands to Commodore BASIC. The image on the screen was created by the program below. If you have been waiting for easy to use, high resolution graphics for your PET, isn't it time you called MTU?



NOW 80 COLUMN PETS CAN HAVE MTU HIGH RESOLUTION GRAPHICS

```
 10 VISMEM: CLEAR
 20 P=160: Q=100
 30 XP=144: XR=1.5*3.1415927
 40 YP=56: YR=1: ZP=64
 50 XF=XR/XP: YF=YP/YR: ZF=XR/ZP
 60 FOR ZI=-Q TO Q-1
 70 IF ZI<-ZP OR ZI>ZP GOTO 150
 80 ZT=ZI*XP/ZP: ZZ=ZI
 90 XL=INT(.5+SQR(XP*XP-ZT*ZT))
100 FOR XI=-XL TO XL
110 XT=SQR(XI*XI+ZT*ZT)*XF: XX=XI
120 YY=(SIN(XT)+.4*SIN(3*XT))*YF
130 GOSUB 170
140 NEXT XI
150 NEXT ZI
160 STOP
170 X1=XX+ZZ+P
180 Y1=YY-ZZ+Q
190 GMODE 1: MOVE X1,Y1: WRPIX
200 IF Y1=0 GOTO 220
210 GMODE 2: LINE X1,Y1-1,X1,0
220 RETURN
```

K-1008-43M Manual only $10 (credited toward purchase)

K-1008-43 Complete ready to install package $495

MASTERCHARGE & VISA accepted

Write or call today for our full line catalog describing all MTU 6502 products, including our high speed 8" Floppy Disk Controller for up to 4 megabytes of PET storage.

MTU

Micro Technology Unlimited

2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605, U.S.A.
(919) 833-1458

NO. 34 MARCH 1981

MICRO™

THE 6502 JOURNAL

STAFF

Editor/Publisher
ROBERT M. TRIPP

Associate Publisher
RICHARD RETTIG

Associate Editor
MARY ANN CURTIS

Special Projects Editor
MARJORIE MORSE

Art Director
GARY W. FISH

Typesetting
EMMALYN H. BENTLEY

Advertising Manager
L. CATHERINE BLAND

Circulation Manager
CAROL A. STARK

MICRO Specialists
APPLE: FORD CAVALLARI
PET: LOREN WRIGHT
OSI: PAUL GEFFEN

Comptroller
DONNA M. TRIPP

Bookkeeper
KAY COLLINS

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional offices
Publication Number: COTR 395770
ISSN: 0271-9002

Subscription Rates:	*Per Year*
U.S.	
through March 31	$15.00
thereafter	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

For back issues, subscriptions, change of address or other information, write to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

CONTENTS

DEPARTMENTS

MICRO Editorial

Copyright/Copywrong

MICRO is unconditionally opposed to the illegal copying of software listings, cassettes, diskettes or any other protected material. I am embarrassed that the need has arisen to make what should be an obvious statement of policy. Due to the publication of an advertisement in our January and February issues, a number of advertisers and readers have expressed concern that MICRO appeared to be supporting the illegal copying of protected disks. In retrospect, I believe that I made a mistake in allowing the ad to run, as will be discussed below.

MICRO could not exist if it were cheaper to copy MICRO than to buy a copy, and if it did not receive protection from being copied through the various copyright laws. This is not usually true of a disk full of software. First, the copyright laws have been very unclear about the protection afforded this type of material. Second, a $395 program on diskette that can be quickly copied to another diskette which only costs $3 is a bargain. Or so it appears to many microcomputer owners. There are, however, many hidden costs in illegal copies or "copywrongs."

1. An author or company that does not get a fair return on its work may fail in business, with the result that none of its later work will be available.
2. A vendor with reason to believe his product will be "ripped-off" may have to increase the price of the product to cover "copywrong" losses.
3. If a vendor is forced to protect software through some hardware or software technique making the software difficult to copy, a legitimate user may then have unnecessary difficulties making a copy for the valid purpose of system backup.
4. A vendor may be forced to make an otherwise straightforward piece of software complex and devious to protect it, and may then only provide the barest operating instructions for its use. Such a presentation causes the product to lose inherent educational value for a programmer, limits the dissemination of useful programming techniques, makes the product difficult to modify and customize, and makes the software difficult for users to understand.
5. Many excellent programs may never be offered for sale at all.

There are certainly other economic and technologic losses besides these. The only one who profits from "copywrong" is the thief. Everyone else loses in the long run.

The advertisement referred to above was for a program that would copy "protected" diskettes. A member of the staff who thought that it should not be run brought it to my attention. I considered a number of factors and then allowed the ad to run. My reasons at the time were:

1. MICRO had never rejected an ad due to content and I hoped that we would not have to start censoring ads.
2. I thought that the ad would appear in other magazines anyway. We have since checked with several national magazines with the result that: one would definitely not print that type of ad; a second might reject it but had not seen it; a third would have the entire editorial staff review it if there were any question in anyone's mind; and a fourth has run the same ad. So, I was partially right, but not entirely.
3. No protection scheme is really safe anyway, as this ad shows, and if an individual really wants to make a copy, he will find a way.

With these thoughts in mind, I approved the ad. The first calls and letters arrived after the next issue was already to press, so I could not stop the second run of the ad. Having since discussed the whole matter with the rest of the staff, and my wife who was incredulous that I approved it in the first place, I realize that I made a bad decision, and apologize to anyone who may have been injured by it. The staff of MICRO will now review every ad and, where there are problems, take appropriate steps.

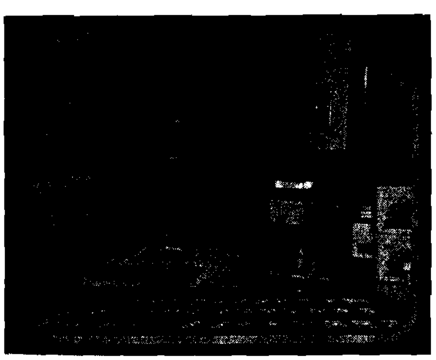
We all pay the price of "copywrong", a price that can never be fully calculated since it results in a loss of software, talent, ideas and concepts whose value is impossible to measure. Unfortunately, illegal copying is similar to voting. No individual feels that his action really makes a difference, but of course it does. The numerous "copywrongs" add up to one of the most serious problems facing the microcomputer industry.

That is MICRO's position; what is yours?

Robert M. Tripp

Editor/Publisher

About the Cover



Our cover this month shows a medical laboratory with centrifuge and a hypothetical blood-test report on the screen. Microcomputers have already found their way into the laboratory and into medicine. "Microprocessors in Medicine: The 6502", a MICRO column by Jerry W. Froelich, M.D. (pages 63-64 in this issue), continues to address some of the biomedical applications of the 6502.

A microcomputer can be the heart of a complex of instruments, controlling the actions of each, collecting and analyzing data, and sending the collected data to other instruments for display, printout, or storage. Digital meters, frequency counters, oscilloscopes, spectrum analyzers, plotters, and printers, are just a few of the many instruments that can be interfaced to microcomputers. Bookkeeping duties, such as generating the blood-test report, are trivial matters for the computer.

In medicine, possible uses for microcomputers are amazing, but current applications are already impressive. For example, a computer-patient interview has many advantages over conventional methods, not only saving the physician time, but also providing him with a more thorough analysis of the patient's history and symptoms. The interview program is written in consultation with experts in each medical specialty, thereby effectively providing the patient with the combined knowledge and experience of all. The attending physician has therefore more time to treat the patient or can more confidently direct him to the right specialist.

MICRO Letterbox

Dear Editor:

Since issue number one, Micro has been presenting assembler work in a form that I could only term the "MICRO" format of mnemonics. At first, this seemed well and good, as some obvious advantages are present. There were some pros and cons on this by readers in the first few issues, then the discussion diminished, but the mnemonics remained.

I would like to ask the MICRO staff to re-consider their position. It's been 3 years, and MICRO is the only magazine espousing these mnemonics. Use by others has not caught on for one reason or another. With the exception of the assembler that MICRO uses for its articles, very few commercially available assemblers handle these mnemonics.

For programmers like myself, with assemblers that don't use the MICRO mnemonics, they are a drawback, requiring extra effort to re-write prior to entering, or tending to encourage mistakes if on-the-fly translation is attempted while entering the program. I think you may be encouraging extra work by the majority of programmers who don't have assemblers which handle these mnemonics.

I suggest a retreat from your previous position, and the use of an assembler which uses standard mnemonics and syntax, still retaining the concept of using standardized listings which is very valuable.

Frank Lawyer
126 Demott Lane
Somerset, NJ 08873

We have begun to use standard mnemonics (see MICRO 33:50) and will continue to do so.

The Editor

Dear Editor:

For a quick and dirty listing of Tim Finkbeiner's "non-listable" program in January's "Letterbox", try this:

a. Use the monitor to examine and write down the contents of bytes 0301 hex and 0302 hex which will be wiped out by the following restart.
b. Now break and cold start your OSI C1P but answer "MEMORY SIZE" with 770 (terminal width is normal). This allows BASIC to reset pertinent pointers but prevents the destruction of your program.
c. Use the monitor to restore 0301 hex and 0302 hex to their former values.
d. Now break, warm start, and list.

This routine also helps recover programs which may become non-listable due to an inappropriate POKE or some other error.

Robert J. Murrell
228 Springwood Drive
Verona, PA 15147

A 6502 Assembler in BASIC

This article describes a 6502 assembler written in BASIC and tuned up for an OSI C2-4P computer. It is usable in a 4K machine and can be adapted to other makes and models of personal computers that use the 6502 microprocessor.

Edward H. Carlson
3872 Raleigh Dr.
Okemos, Michigan 48864

The native tongue of a 6502 chip is "machine language." The native tongue of the programmer isn't. Nevertheless, you can speak to your computer in its native language. Just sit down and POKE its keyboard. Say: "AD 03 D2" and it will understand you. But it is difficult to carry on a prolonged conversation this way. Many programmers got their start in machine language by writing out programs on paper, looking up the op codes in a table, then pecking the resulting gibberish into the machine. This works. I wrote several useful programs this way shortly after getting my Ohio Scientific C2-4P.

The coolie labor is so great, however, that one soon urgently covets an assembler. An assembler is a longish program (about 5K for the Ohio Scientific "6500 Assembler and Editor") which does a number of onerous tasks. Some assemblers are more elaborate than others, but all will do two of the most mind-numbing tasks needed to produce machine code: the translation of mnemonic operators into op code, and the calculation of offsets in branch instructions when given the address of the target line.

Assemblers are notoriously demanding of memory space, and it usually costs several decibucks for that little cassette with its machine language incantations. So I was

```
1 GOTO 1990:REM *****ASSEMBLER *****
2 M1=INT(M/16):M2=M-M1*16:M1=FNH(M1):M2=FNH(M2)
3 Z=Z+1:POKE Q+Z,M1:Z=Z+1:POKE Q+Z,M2:RETURN
4 Z=Z+1:GOSUB 2:POKE AD,M:AD=AD+1:RETURN
5 HI=INT(N/256):LO=N-256*HI:BY=3
7 II=INT(AD/256):JJ=AD-II*256:M=II:Z=1:GOSUB 2
8 M=JJ:GOSUB 2:M=OP:Z=Z+1:GOSUB 4
10 IF BY>1 THEN M=LO:GOSUB 4
11 IF BY=3 THEN M=HI:GOSUB 4
12 GOTO 100:REM START MAIN LOOP
20 FOR Z=1 TO LEN(C$):POKE N+Z,ASC(MID$(C$,Z,1)):NEXT:RETURN
99 C$="NO":N=Q+21:GOSUB 20
100 PRINT AD;:INPUT C$:L$=LEFT$(C$,3):L=LEN(C$)
101 IF L>4 THEN C$=RIGHT$(C$,L-4):L=L-4
102 IF L$="HEX" THEN 4000:REM MUST HAVE 4 DIGITS (NO $ SIGN)
103 IF L$="ADD" THEN AD=VAL(C$):GOTO 100
104 IF L$="CON" THEN CA=0:OP=VAL(C$):GOTO 200
105 IF L$="DIS" THEN AD=VAL(C$):OP=PEEK(AD):CA=0:GOTO 200
106 IF L$="ASC" THEN M=ASC(C$):Z=21:GOSUB 2:GOTO 100
123 REM IDENTIFY THE 3 LETTER NMEMOMIC
124 FOR I=1 TO 4:FOR J=1 TO 56:N=4*J-3
130 IF L$=MID$(C$(I),N,3) THEN II=I:JJ=J:GOTO 155
144 NEXT J,I:GOTO 99
155 N=14*(II-I)+JJ
160 REM GET OP CODE AND CATAGORY
161 CA=VAL(MID$(E$,N,1))
163 OP=VAL(MID$(F$(II),JJ*4-3,3))
200 BY=1:IF CA=0 THEN 7
210 IF C$="A" AND CA=3 THEN OP=OP+8:GOTO 7
213 IF C$="A" THEN 99
219 REM ABOVE: 1 BYTE CODES, BELOW: 2 BYTE ONES
220 BY=2:C1$=LEFT$(C$,1)
221 II=OP-8*(CA=1)
223 JJ=CA=1 OR CA=4 OR CA=5
224 IF C1$="#" AND JJ THEN LO=VAL(RIGHT$(C$,L-1)):OP=II:GOTO 7
228 IF C1$="#" THEN 99
230 IF C1$<>"(" THEN 260
231 LO=VAL(MID$(C$,2,L-4))
232 IF RIGHT$(C$,3)=");Y" AND CA=1 THEN OP=OP+16:GOTO 7
240 IF RIGHT$(C$,3)=";X)" AND CA=1 THEN 7
250 IF RIGHT$(C$,1)<>")" OR CA<>6 THEN 99
254 N=VAL(MID$(C$,2,L-2))
258 OP=OP+32:GOTO 5
260 IF RIGHT$(C$,2)<>";X" THEN 280
261 REM FOUND: 3 BYTE CODES, GO TO 7; 2 BYTES, TO 5; NOT FOUND TO 99
262 N=VAL(LEFT$(C$,L-2))
264 IF N>255 THEN 274
266 LO=N:IF CA=2 THEN OP=OP+16:GOTO 7
268 IF CA=1 OR CA=3 OR CA=5 THEN OP=OP+20:GOTO 7
269 GOTO 99
274 IF CA=2 THEN OP=OP+24:GOTO 5
276 IF CA=1 OR CA=3 OR CA=5 THEN OP=OP+28:GOTO 5
278 GOTO 99
280 IF RIGHT$(C$,2)<>";Y" THEN 300
282 N=VAL(LEFT$(C$,L-2))
284 IF N>255 THEN 292
286 LO=N
287 IF CA=2 OR CA=5 THEN OP=OP+16-4*(CA=5):GOTO 7
292 IF CA=1 OR CA=5 THEN OP=OP+24-4*(CA=5):GOTO 5
299 GOTO 99
300 N=VAL(C$)
304 REM GO TO 340 FOR BRANCH INSTRUCTIONS
305 IF CA=8 THEN 340
310 IF N>255 THEN 332
312 LO=N
314 IF CA=2 OR CA=7 THEN 7
316 IF CA=1 OR CA=3 OR CA=4 OR CA=5 THEN OP=OP+4:GOTO 7
332 IF CA=2 OR CA=7 THEN OP=OP+8:GOTO 5
334 IF CA=1 OR CA=3 OR CA=4 OR CA=5 THEN OP=OP+12:GOTO 5
336 IF CA=6 OR CA=9 THEN 5
```

(continued)

delighted to see an assembler (for a PET) written in BASIC by Mark Zimmermann and published in *Personal Computing*, December 1978. I eagerly typed it into my 4K machine and hit RUN. The computer said "MEMORY OVERFLOW."

The problem is that lengthy tables are required in an assembler, and this particular one, quite naturally, put them in source code as DATA statements and then transferred them to arrays for use. Arrays are expensive of space if integers or short strings are being stored. A much more space-efficient way to enter fixed tables in a program is to put them in as strings, and to disect the strings for use by calling on functions such as RIGHT$, MID$, and LEFT$.

This memory compression required extensive rewriting of the program. When done, I hit RUN again. This time the machine ran a few steps, then fell into a sort of catatonic trance. Stuck, except for a rhythmic screen flicker.

With dismay, I realized that my program had been bitten by the dreaded Screenflickerus stringarrayus bug, which infests Ohio Scientific machines using BASIC-in-ROM Version 1.0, Rev. 3.2, and attacks programs containing string arrays. I sadly set my ravaged project aside.

There now came an interlude in which my computer's memory grew to 20K and I found the bucks to purchase the Ohio Scientific Assembler and the time to learn to use it.

So what am I doing foisting a bug-eaten assembler in BASIC off on you? A few weeks ago, a friend of mine who owns an Ohio Scientific Superboard II inquired about assemblers. He has only 4K of memory and so I thought of my moribund assembler in BASIC. You see, our intrepid scientific establishment, in the person of Mark Minasi, has studied the life habits of S. stringarrayus and found that she never nibbles on string arrays that have length 3*(any integer) + 2. Mr. Minasi published his results in a letter in issue 3 of *PEEK(65)*. So I DIMensioned my string arrays at 5 instead of the otherwise suitable 4, and began a final tune-up of the program. The results are displayed in listing 1.

```
339 GOTO 99
340 N=N-AD-2:IF N<-128 OR N>127 THEN PRINT"CAN'T BRANCH";N:GOTO 100
342 IF N<0 THEN N=N+256
344 LO=N:GOTO 7
1800 REM ***** ASSEMBLER 1.4 *****
1802 REM
1804 REM        FOR OSI C2-4P COMPUTERS
1806 REM
1808 REM        EDWARD H. CARLSON
1810 REM        3872 RALEIGH DR.
1812 REM        OKEMOS MI 48864
1814 REM
1820 REM L$ IS THE 3 LETTER OP MNEMOMIC LIKE "LDA"
1822 REM C$ FIRST IS THE WHOLE INSTRUCTION, THEN JUST THE
1824 REM    ARGUMENT LIKE "55;Y"
1826 REM OP IS THE MACHINE LANGUAGE OP CODE
1828 REM AD IS THE CURRENT ADDRESS
1830 REM BY IS THE NUMBER OF BYTES IN THE MACHINE LANGUAGE
1832 REM    INSTRUCTION
1834 REM N  IS THE 1 OR 2 BYTE NUMBER IN THE INSTRUCTION
1836 REM    ALSO A TEMPORARY VARIABLE
1838 REM CA IS THE CATAGORY NUMBER OF THE INSTRUCTION
1840 REM    IT IS OBTAINED FROM THE TABLE E$
1842 REM Q  LOCATES THE CODE ON THE TV SCREEN
1844 REM    THE REMAINDER OF THE VAR.S ARE TEMPORARY
1899 REM
1990 DIM C$(5),F$(5)
1992 DEF FNH(D)=D+48-7*(D>9)
1995 Q=55058
2000 C$(1)="ADC AND ASL BCC BCS BEQ BIT BMI BNE BPL BRK BVC BVS CLC"
2012 C$(2)="CLD CLI CLV CMP CPX CPY DEC DEX DEY EOR INC INX INY JMP"
2013 C$(3)="JSR LDA LDX LDY LSR NOP ORA PHA PHP PLA PLP ROL ROR RTI"
2016 C$(4)="RTS SBC SEC SED SEI STA STX STY TAX TAY TSX TXA TXS TYA"
2020 E$="113888788808800001442001200691553010000330010001220000000"
2021 F$(1)="097 033 002 144 176 240 036 048 208 016 000 080 112 024"
2023 F$(2)="216 088 184 193 224 192 198 202 136 065 230 232 200 076"
2025 F$(3)="032 161 162 160 066 234 001 072 008 104 040 034 098 064"
2027 F$(4)="096 225 056 248 120 129 134 132 170 168 186 138 154 152"
2029 G$="0123456789ABCDEF"
2030 AD=546:GOTO 100:REM END OF INITIALIZATION, START AT $0222
3999 REM HEX TO DECIMAL SUBROUTINE
4000 N=0:L=4096:FOR I=1 TO 4
4020 M=ASC(MID$(C$,I,1))-48
4040 IF M>9 THEN M=M-7
4050 N=N+M*L:L=L/16:NEXT:C$=STR$(N):N=Q+20:GOSUB 20:GOTO 100
```

Writing Fast BASIC

The tune-up had 3 purposes: to gain still more room, to regain some speed, and to add some utilities which greatly ease the task of the program hacker. So extensive was the tune-up that scarcely a line remains of the original program. An unfortunate side effect is that much of the clarity of the original has also been lost. I refer you to Mr. Zimmermann's original article for an explanation of the logical flow of the assembly process. Here I want to discuss two aspects of the program: the details of the tune-up, and a manual for use.

In a BASIC program that has quite a few GOSUBs and GOTOs, one of the most time-consuming activities of the interpreter is searching for line numbers. Upon encountering "100 GOTO 2040", the BASIC interpreter starts at the first line of the program and examines each line, in turn, to see if its number is 2040. This costs 0.85 milliseconds per line examined. It can really add up unless the sought-for line is near the top of the program. To maximize running speed, I now write all my BASIC programs in an inverted order. All the initialization is done at the end of the program, reached by a GOTO in line 1. In listing 1, initialization starts at line 1800 with the REM description of the program and the preparation of the arrays. All subroutines are at the beginning of the program, where their line numbers can be found quickly. The main loop is in the middle, starting at line 100 of listing 1.

In the same vein, the program runs faster if one can eliminate unnecessary flow-diverting statements altogether. A fruitful place to start is by looking at IF statements. The program is faster if the most probable value of the argument of an IF is "false", so that the flow continues down the stream of lines, rather than "true" so the THEN option is followed and a GOTO is exercised.

A second expensive activity of the BASIC interpreter is the conversion of numerical constants to floating point numbers. Each time the line "100 A = 53248 + B" is encountered, the constant "53248" has to be converted to a 4-byte floating point number. The time required for conversion is proportional to the number of digits and is quite long. The interpreter never remembers that it has already done this conversion on previous encounters with line 100. So it is better to write "2000 Q = 53248" in the initialization part of the program and "100 A = Q + B" in the main loop. A conversion must also be done for the target line numbers in

GOTO statements. This is another reason for putting target lines at the front of the program, so their target addresses have fewer digits needing conversion. Line numbers at the beginning of lines do not need conversion. They are stored in source code as 2-byte binary numbers.

Putting several statements on one line, separated by colons, is another space and time saver. Finally, savings can be made by using 1-letter variable names, and by leaving out spaces when writing BASIC code. I draw the line at this however, clarity counts too!

User Manual

Now let's write a program. First punch in the assembler. If you have a 4K machine, you must leave out all REMarks in order to fit it in. This will give you over one page of memory for your machine language program, which you must reserve at the top of memory space by answering, say, 3750 to the question MEMORY SIZE ? at cold start time. (You can put line "100 ? FRE(8)" to display how much memory is left as you run.) In addition to high memory space, you have available most of page $02: the part above $0222 which is unused by BASIC. This adds up to quite a lot of space since a machine language program that is 1 page long is getting on up in size. Much larger than that and you will want to be using a "real" assembler. But if you do need more room, you can get over 100 bytes more by not allowing any spaces in lines 1 to 1995.

After you hit RUN, the machine answers "546 ?" and waits for you to enter your line of assembly code. The assembler then POKEs its response into the remaining space on the line, increments the line counter and prints the next line number. (The number "546" is decimal for $0222, the default start address. This address is suitable for OSI machines and prevents you from absent-mindedly POKEing a hole in your BASIC source code. You can change the starting address by using the ADD command described below.)

Regrettably, I had to make one departure from standard assembly language syntax: the use of a semicolon in place of a comma. This was necessary because on INPUT, BASIC treats the comma as a field separator and not as an ASCII character, so that an indexed command such as "LDA $5A,Y" looks like "LDA 90;Y" in this assembler's syntax.

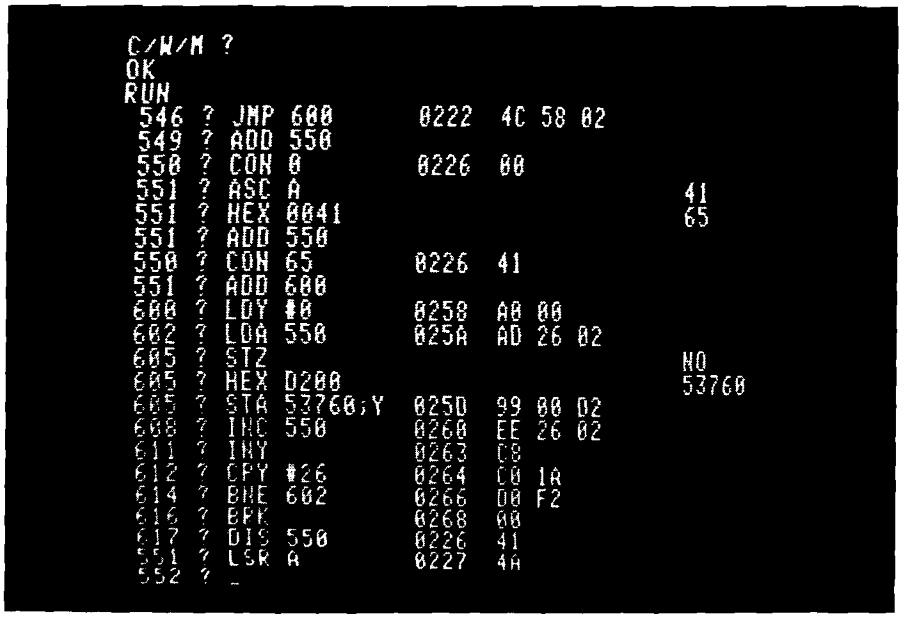


Photo 1. A machine language program written using the assembler in BASIC.

Photo 1 shows a short program that I composed "at the keyboard", without a precursor on paper. It is a realistic test of the assembler, because I made errors and recovered from them. The program writes the alphabet on the screen and photo 2 shows the results. I have used all the utilities that I added to this version of the assembler. They include ADD which starts assembly at a new address, and CON which allows a one-byte constant to be inserted at the current address. Therefore, the argument of CON must be between 0 and 255. The main use of CON is in the construction of tables of constants. DIS displays the contents of a location in memory. Unfortunately, it also sets the address one past the displayed address location. Although this feature is not at all convenient, I couldn't see any way to fix it up easily. ASC y gives the ASCII code (in hex) for the character y.

Finally, HEX gives the decimal equivalent of a 4-digit hexadecimal number. The assembler expects all numbers to be in decimal form. It converts all these to hexadecimal for use and display. The function of HEX is thus needed to supply decimal numbers from hex numbers, so you can "close the loop" in your thinking as you construct the machine language program.

As you work, you will certainly make mistakes, or at least change your mind. The machine is quite tolerant of this. Single incorrect characters can be erased by the "SHIFT/O" as usual. Incorrect mnemonics cause the computer to say "NO". If you disturb BASIC too much, it may undergo an ERROR BREAK and require you to restart the assembler with a RUN. This will do no harm as long as you use ADD to get back to the point in memory where you were last working.

By the way, it takes a perceptible time for the assembler to massage a given input. The bottleneck is the loop starting at line 124. Codes near the end of the alphabet, such as TYA, take longer than ones near the beginning, such as BIT. It should be possible to speed up this loop by doing a preliminary search on the first letter of the 3-letter mnemonic.

When you are ready to run your program, do a BREAK to MONITOR, put in your start address, and hit "G" for go. However, you lose all that good mnemonic stuff that you had put on the screen, so copy it onto a piece of paper first.

Photo 2. The run image of the program shown in Photo 1.

So there you have it: a simple assembler. Anyone who has used a more elaborate assembler will immediately miss some important features such as lables, comments, editor functions, and the ability to write the assembly language code to tape. However, anyone who has only done machine language programming on paper, with op code tables and hex arithmetic, will immmediately be grateful for the help which this assembler program gives to the programmer.

MICRO

UnwrApple

This output pre-processor prevents words from being split by the right margin.

David Lubar
249 Loring Ave., Apt. 3
Edison, New Jersey 08817

In certain programs, such as conversational games, there is a good chance that words printed near the right margin will be bisected in strange places. One solution would be to handle printed portions as string variables which are edited before being sent to the screen. This approach, however, is slow and requires extensive changes in existing programs. Another route would be to do the work with a machine-language subroutine. This article describes such a program.

I had several goals in mind when I wrote UnwrApple:

1. It had to work with any setting of the text window.

2. It had to be simple to use.

3. There had to be a minimum chance of conflict with any portion of a BASIC program.

Of the three goals, 2½ were achieved. (The routine doesn't cope very well when the left and right margins are only one or two characters apart, but this shouldn't be much of a limitation.)

Using UnwrApple

The subroutine is activated with a CALL 881. This can be done either in immediate mode or as a line in the BASIC program, but the BASIC program should be in memory before UnwrApple is called. The subroutine is written for use with Integer BASIC. With a slight modification, it can also be used with Applesoft. (More on that later.) To turn UnwrApple off, hit RESET, or enter POKE 54,240 followed by POKE 55,253.

How It Works

The CALL 881 does two things. First, the address of the subroutine is put into CSWL and CSWH (locations $36,37). Most of you are probably familiar with the COUT function in the monitor. If not, see Bob Sander-Cederlof's article "A Slow List for Apple BASIC" in *The Best of MICRO*: Vol. 1, pg. 94 for a good description of COUT. After setting these pointers, an area one page (256 bytes) below the end of the BASIC program is reserved for temporary storage. This is done using the pointers at $CA,CB, which hold the value of the last location used for program storage.

Now for the actual editing routine. First, since I wasn't sure whether BASIC used the Y register between jumps to COUT, I saved Y at the start of the subroutine. Then CH ($24), which is the cursor displacement from the left margin, is stored in Y. The ASCII value of the character to be printed, already contained in the A register, is stored in the reserved area for later reference. A check must be made to see if this character is the last one of the current line. If it isn't, it is sent to the screen through the print routine COUT1 ($FDF0). With the last character, there are three possibilities.

1. The last character is a space.

2. The last character isn't a space and it is followed by

   a. A space.
   b. Another character.

```
0800          ;********************************
0800          ;*                              *
0800          ;*          UNWRAPPLE           *
0800          ;*                              *
0800          ;*       BY DAVID LUBAR         *
0800          ;*                              *
0800          ;*     MICRO #34--MARCH '81     *
0800          ;*                              *
0800          ;********************************
0800          ;*
0800          ;*
0800          ;ENTER FROM BASIC WITH CALL 881
0800          ;*
0800          YSAVE  EPZ $00
0800          TEMPLO EPZ $02
0800          TEMPHI EPZ $03
0800          ASAVE  EPZ $04
0800          CHSAVE EPZ $05
0800          LWIND  EPZ $21
0800          CH     EPZ $24
0800          CSWL   EPZ $36
0800          CSWH   EPZ $37
0800          PPLO   EPZ $CA
0800          PPHI   EPZ $CB
0800          COUT1  EQU $FDF0
0800          ;*
```

(continued)

In case #1, the space is printed and the routine goes on without making any changes. In case 2a, the space is not printed, thus keeping the left margin justified. In 2b, the previous line has to be edited. For 2a or 2b, the routine first goes to FIX where it changes the COUT pointers. The first character of the new line will now be sent to CHECK instead of START. If editing is needed, the temporary storage area is searched until a space is found. Next, a backspace is printed, moving the cursor back one line. (After printing the final character of the line, the monitor moves the cursor down. The cursor has to be moved back up before the partial word can be erased.) After replacing the partial word with blanks on the screen, the erased letters are reprinted. From here, there are just two more steps. The letter that originally started the new line, but which hasn't yet been printed, is sent to the screen. Finally, the COUT pointers are again set to START.

In other words, whenever a word is cut off by the left margin, the fragment is removed from that line and reprinted on the next line.

Modifications

If your programs make heavy use of the zero page (for tone subroutines or whatever), you can use locations other than $0-$5 for storage. The area directly above UnwrApple from $383-$3FF could be used (keep 3F8 free if you use control-Y). Of course, using non-zero page locations will add a few bytes to the subroutine. To use UnwrApple with Applesoft, replace the listed values of PPLO and PPHI ($CA,$CB) with $6D,$6E. Since only 40 bytes are actually used for temporary storage, TEMPLO and TEMPHI could be given values pointing to page 3 (using $3B0-$3FF, for example). In this case, the lines CALLed from BASIC will have to be changed.

That about covers everything. UnwrApple can be used in the direct mode. I wouldn't recommend leaving it on while entering a program since the BASIC interpreter might not appreciate some of the extra spaces. But you can see it in action by CALLing it, then typing some words on the screen. For example, when the cursor reaches a new line, try hitting the space bar. The first space won't be printed, but any following ones will be sent to the screen.

One last thought—UnwrApple could be incorporated as part of a word processing routine, but that's another story.

```
0300                 ;*
0300                 ;ON ENTRY ACCUMULATOR HOLDS ASCII VALUE
0300                 ;*
0300 8400     START  STY YSAVE         ;SAVE Y ON ENTRY INTO ROUTINE
0302 A424            LDY CH            ;GET DISPLACEMENT FROM LEFT WINDOW
0304 9102            STA (TEMPLO),Y    ;SAVE CHAR. FOR LATER REFERENCE
0306 C8              INY
0307 C421            CPY LWIND         ;LAST CHAR. OF LINE
0309 F005            BEQ MARG          ;YES
030B A400            LDY YSAVE         ;NO. RESTORE Y
030D 4CF0FD          JMP COUT1         ;PRINT IT
0310 C9A0     MARG   CMP #$A0          ;SPACE?
0312 D005            BNE FIX           ;NO
0314 A400            LDY YSAVE         ;YES
0316 4CF0FD          JMP COUT1         ;PRINT THE SPACE
0319 A026     FIX    LDY #CHECK        ;RESET COUT POINTERS TO SEND
031B 8436            STY CSWL          ;FIRST CHAR. OF NW LINE TO CHECK
031D A003            LDY /CHECK
031F 8437            STY CSWH
0321 A400            LDY YSAVE         ;RESTORE Y
0323 4CF0FD          JMP COUT1         ;PRINT LAST CHAR. OF LINE
0326 8400     CHECK  STY YSAVE
0328 C9A0            CMP #$A0          ;IS FIRST CHAR. OF NEW LINE A SPACE
032A F040            BEQ NULL          ;YES
032C A421            LDY LWIND         ;NO. PUT WINDOW LENGTH IN Y
032E 8504            STA ASAVE         ;SAVE FIRST CHAR. OF NEW LINE
0330 88              DEY               ;SET Y TO VALUE OF OLD CH
0331 88       COUNT  DEY               ;BEGIN SEARCH FOR START OF LAST WORD
0332 F029            BEQ CANT          ;SEARCH UNSUCCESSFUL
0334 B102            LDA (TEMPLO),Y    ;CHECK CHARS. OF LAST LINE
0336 C9A0            CMP #$A0          ;SPACE
0338 D0F7            BNE COUNT         ;NO. KEEP LOOKING
033A A988            LDA #$88          ;YES. BACKSPACE TO LAST LINE
033C 20F0FD          JSR COUT1         ;USE JSR SO CONTROL WILL RETURN HERE
033F 8405            STY CHSAVE        ;SAVE START OF LAST WORD
0341 8424            STY CH            ;SET CH FOR PRINTING SPACES
0343 A9A0     BLANK  LDA #$A0          ;THIS LOOP ERASES THE PARTIAL WORD
0345 20F0FD          JSR COUT1         ;PRINT A SPACE
0348 C8              INY
0349 C421            CPY LWIND         ;END OF LINE?
034B F003            BEQ REPR          ;YES
034D 4C4303          JMP BLANK         ;NO. KEEP GOING
0350 A405     REPR   LDY CHSAVE
0352 C8              INY               ;MOVE PAST FIRST SPACE
0353 B102     REPR1  LDA (TEMPLO),Y    ;GET ERASED CHAR.
0355 20F0FD          JSR COUT1         ;PRINT IT ON NEW LINE
0358 C8              INY
0359 C421            CPY LWIND         ;DONE?
035B D0F6            BNE REPR1         ;NO.
035D A504     CANT   LDA ASAVE         ;YES. RESTORE ORIGINAL FIRST CHAR.
035F A000     SET    LDY #START        ;RESET COUT POINTERS
0361 8436            STY CSWL
0363 A003            LDY /START
0365 8437            STY CSWH
0367 A400            LDY YSAVE         ;RESTORE Y
0369 4CF0FD          JMP COUT1         ;PRINT RESTORED CHAR.
036C A980     NULL   LDA #$80          ;PRINT NULL INSTEAD OF SPACE TO
036E 4C5F03          JMP SET           ;JUSTIFY LEFT MARGIN
0371                 ;*
0371                 ;CALL FROM BASIC GOES TO HERE
0371                 ;*
0371 A900            LDA #START        ;SET COUT POINTERS
0373 8536            STA CSWL
0375 A5CA            LDA PPLO          ;GET END OF PROGRAM STORAGE
0377 8502            STA TEMPLO        ;ANDSET UP AN AREA ONE PAGE
0379 A4CB            LDY PPHI          ;BELOW FOR TEMPORARY STORAGE
037B 88              DEY               ;OF PRINTED CHARS.
037C 8403            STY TEMPHI
037E 60              RTS               ;BACK TO BASIC
```

MICRO

New Publications *(continued from page 79)*

articles, columns, book reviews, hardware and software reviews, etc. To provide a "current awareness" service, abstracts are arranged by periodical and issue, so that a reader can quickly scan each issue of the periodicals covered.

CONTENTS: *80 Microcomputing, Apple Orchard, Byte, Call A.P.P.L.E, Compute, Compute II, Creative Computing, Dr. Dobb's Journal, Interface Age, Kilobaud Microcomputing, MICRO: The 6502 Journal, Nibble, onComputing, Personal Computing, Purser's Magazine, Recreational Computing, S-100 Microsystems,* and *Sourceworld.*

SYM-ple Sym-on

The SYM does a fine imitation of a popular electronic game.

Len Green
15 Yotam Street
Achuza
34 675 HAIFA
ISRAEL

MICRO-induced Homebrew Printer and Terminal

Shawn Spilman's excellent article "Writing for MICRO" in the September 1979 issue of MICRO (16:59), strongly advocates submission of '... a working source of the actual computer program; that is, the assembler or compiler output listing ... let the computer generate the program listing ...' Ay, there's the rub! A precondition for this is some form of hard copy device, and in some countries even a second-hand one is unobtainable or prohibitively expensive. So my son, David Green, decided to turn a junked fifty-year old, purely mechanical typewriter into a crude sort of electrical daisy-wheel printer which delivers about 2 to 3 characters per second. The idea became feasible when he discovered, in our local junk shop, a 52-contact electric rotary selector once used in telephone exchanges. Software and a VIA in SYM drive this printer. It's primitive and temperamental, but provides me with legible hard copy. However, the accompanying listings for this article were reassembled by MICRO.

My project involved a few hundred hours of electrical, mechanical and electronic engineering, making it highly educational but not exactly practicable! However, if anyone is interested I will be glad to supply details.

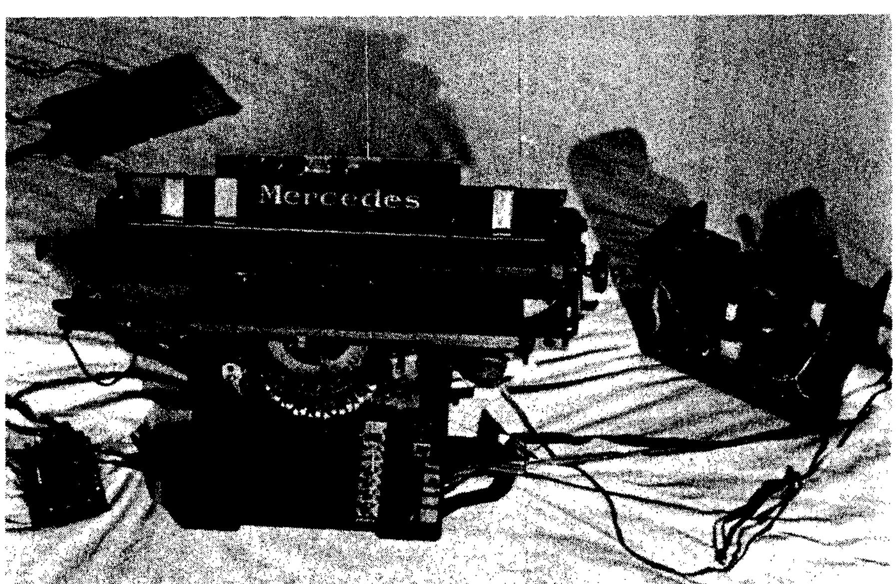
My present "terminal" is a cannibalized calculator keyboard which inputs all essential ASCIIs, with either SYM's six 7-segment LEDs scrolled from right to left as output display, or the onboard single-line 32-character 'scope interface. All do-it-yourself! I am shortly going to make a proper keyboard video terminal.

In USA, I understand that SIMON was, and probably still is, a very popular toy. In some other countries it is almost unknown. It is an absorbing game for 1, 2, 4 or more players, but could also be adapted for more serious educational or diagnostic purposes.

If you own a SYM-1, less than 220 bytes of RAM will give you the main facilities of SIMON and another couple in addition. No offboard hardware is needed. Some ideas about adding external audio and adapting for other microprocessors will be presented at the end of this article.

To simplify description later, keys 0 to 3 produce four different musical notes while simultaneously lighting four associated LEDs, simulating the 4 different colored lights of SIMON. Key 0 sounds the lowest pitch tone on SYM's onboard beeper and activates the extreme lefthand onboard 7-segment display LED. Key 1 produces a higher tone and activates the second LED, etc. The fifth, and extreme righthand sixth LEDs, are exclusively reserved for score display.

In case you haven't met SIMON, simply load the program and key in GO/265/CR. One of the four possible musical notes will play and its corresponding display LED will light up with a triple bar. At the same time 01 will appear in the score display, informing you that your tune is only one note long. You must then press the appropriate key from 0 to 3 in order to replay the identical "tune". If you have done this correctly, your microprocessor will respond with the original



note and displays followed by a new one, and accompanied by 02 in the score display. You must now key in the correct two-note tune. Each round, the microprocessor plays the complete preceding tune and adds one additional note at the end.

When you make a mistake, your microprocessor sounds a raspberry and displays 4 question marks (segment code = D3) together with the number of notes you have succeeded in remembering correctly. The whole process then automatically starts over again from 01 with the same tune. In order to add a winning target, I have arbitrarily chosen 09, which can be altered, at will, in location $0278. When you complete nine notes successfully, the LEDs will display "H.H.H.H.09" (H. = Home = segment code, F6) and the whole game will restart again from 01 with the same tune. I have made this target auto-adjusting. Each time you fail, the winning target is decremented, and each time you win, it is incremented. The score display is in hex; a few extra bytes will alter this to decimal if desired. The "?"s and "H."s can easily be replaced by slogans.

The basic parameters can all be altered according to personal taste. They are: the four tone frequencies, the duration of the notes, the pause between notes, the maximum tune length, (set arbitrarily at 70 notes), and the original target number. Their locations in the program are easily identifiable. The crude method of tone generation inevitably causes the lower frequency notes to have relatively longer duration. If considered worthwhile, this could be obviated by utilizing one of SYM's many timers. The 6 byte pseudo-random note selector routine RAND is not very scientific, but is perfectly effective here.

To check that you have keyed in the program correctly or that it has not subsequently modified itself, use SYM's onboard VERIFY command. The correct checksum from $0200 to $02E4 is #5F3F. The whole routine is fully and easily relocatable. Simply modify the seven addresses whose high byte is 02 in module MAIN.

This program is an incredibly powerful and versatile tool for such little RAM and is excellent, particularly for limited systems and relative beginners like me. 2K-SA imposes certain inevitable disciplines on assembly. One restriction is that absolute addressing is impossible within a module. To overcome this, I have

```
0800                ;SYM-PLE SYM-ON SOURCE
0800                ; BY LEN GREEN (MICRO #34)
0200                        ORG $0200
0200                        OBJ $0800
0200                ;PAGE ZERO LABELS
0200                TUNE    EPZ $0007
0200                TEMP2   EPZ $0006
0200                TEMP1   EPZ $0005
0200                INDADL  EPZ $0004
0200                INDADH  EPZ $0003
0200                TARGET  EPZ $0002
0200                SCORE   EPZ $0001
0200                TNEIND  EPZ $0000
0200                ;SYM MONITOR LABELS
0200                TV      EQU $A656
0200                DISBUF  EQU $A640
0200                TIMER   EQU $A41E
0200                DDRB    EQU $A403
0200                ORB     EQU $A402
0200                TIL     EQU $A004
0200                ACCESS  EQU $8B86
0200                CONFIG  EQU $89A5
0200                SCAND   EQU $8906
0200                GETKEY  EQU $88AF
0200                DELAY   EQU $835A
0200                OUTBYT  EQU $82FA
0200                DBOFF   EQU $80D3
0200                ;TABLES. 4(TONE) PERIOD FACTORS
0200 80             TABLES  BYT $80
0201 40                     BYT $40
0202 18                     BYT $18
0203 02                     BYT $02
0204                ;WAIT. SUBROUTINE: DELAY FOR ABOUT 1 SECOND
                    , VARIABLE THROUGH PARAMETERS
0204                ; IN X AND Y. DBOFF IS ANY SUITABLE TRANSPA
                    RENT MONITOR SUBROUTINE TO PAD
0204                ; OUT TIMING PERIOD. X PRESERVED.
0204 8A             WAIT    TXA
0205 48                     PHA
0206 A280                   LDX #$80
0208 A070           LOOP1   LDY #$70
020A 20D380         LOOP2   JSR DBOFF
020D 88                     DEY
020E D0FA                   BNE LOOP2
0210 CA                     DEX
0211 D0F5                   BNE LOOP1
0213 68                     PLA
0214 AA                     TAX
0215 60                     RTS
0216                ;
0216                ;BLANK. SUBROUTINE: FILL LEFT-HAND 4 LOCATI
                    ONS OF DISPLAY BUFFER (DISBUF TO DISBUF+3)
                    WITH BLANK SEGMENT CHARACTER
0216                ; #00. X PRESERVED.
0216 A900           BLANK   LDA #$00
0218 A003                   LDY #$03
021A 9940A6         NEXT    STA DISBUF,Y
021D 88                     DEY
021E 10FA                   BPL NEXT
0220 60                     RTS
0221                ;PLAY. SUBROUTINE: TRANSFER PERIOD
0221                ; TONE, EACH TIME SINGLE SCANNING
0221                ; DISPLAY WITH CONTENTS OF DISBUF.
0221                ; DURATION VARIABLE THRO' PARAMETERS IN TEM
                    P1 AND TEMP2.
0221                ; (BEEP) ENABLES ONBOARD BEEPER.
0221 8500           PLAY    STA TNEIND
0223 A90F                   LDA #$0F
```

```
0225 8D03A4            STA DDRB
0228 A903              LDA #$03
022A 8505              STA TEMP1
022C A970       LOOP3  LDA #$70
022E 8506              STA TEMP2
0230 8A         LOOP4  TXA
0231 48                PHA
0232 200689            JSR SCAND
0235 68                PLA
0236 AA                TAX
0237            ;(BEEP)
0237 A90D              LDA #$0D
0239 20A589            JSR CONFIG
023C A908              LDA #$08
023E 8D02A4            STA ORB
0241 A400              LDY TNEIND
0243 88         DEC1   DEY
0244 D0FD              BNE DEC1
0246 A906              LDA #$06
0248 8D02A4            STA ORB
024B A400              LDY TNEIND
024D 88         DEC2   DEY
024E D0FD              BNE DEC2
0250 C606              DEC TEMP2
0252 D0DC              BNE LOOP4
0254 C605              DEC TEMP1
0256 D0D4              BNE LOOP3
0258 60                RTS
0259            ;FIXPLY. SUBROUTINE: NOTE - ID
0259            ; (00 TO 03) IN A LOADS TRIPLE BAR SEGMENT
                CODE #49 INTO APPROPRIATE
0259            ; SEGMENT POSITION IN DISBUF AND   APPROPRI
                ATE PERIOD FACTOR BACK INTO A.
0259            ; PERFORM PLAY.
0259 A8         FIXPLY TAY
025A A949              LDA #$49
025C 9103              STA (INDADH),Y
025E B90002            LDA TABLES,Y
0261 202102            JSR PLAY
0264 60                RTS
0265            ;INIT. START MAIN ROUTINE: ACCESS TO UN-WRI
                TE-PROTECT SYSTEM RAM. GENERATE
0265            ; AND STORE 70 PSEUDO-RANDOM NOTE ID'S (00
                TO 03) INTO TUNE BUFFER.
0265            ; INITIALIZE TARGET#, INDIRECT ADDRESS (IND
                AD) OF DISBUF.
0265 20868B     INIT   JSR ACCESS
0268 A245              LDX #$45
026A AD1EA4     RAND   LDA TIMER
026I 6D04A0            ADC TIL
0270 2903              AND #$03
0272 9507              STA TUNE,X
0274 CA                DEX
0275 10F3              BPL RAND
0277 A909              LDA #$09
0279 8502              STA TARGET
027B A940              LDA #$40
027D 8503              STA INDADH
027F A9A6              LDA #$A6
0281 8504              STA INDADL
0283            ;MAIN. CONTINUE MAIN ROUTINE:
0283            ; INIT SCORE. RUN COMPLETE ROUND OF GAME AN
                D EXHIBIT SCORES.  IF TARGET
0283            ; ACHIEVEDDISPLAY "H."S AND SCORE, INCREMEN
                T TARGET AND FORCE BRANCH BACK
0283            ; TO MAIN. ELSE SOUND RASPBERRY, DISPLAY "?
                "S AND SCORE, DECREMENT TARGET
0283            ; AND FORCE BRANCH BACK TO MAIN.
0283 A900       MAIN   LDA #$00
```

(continued)

employed a forced relative branch instruction instead of an absolute jump instruction in 4 places in module MAIN: which also saves one byte each time. 2K-SA uses a modular approach and encourages assembly of the main routine after the subroutines and tables to which it refers. This, incidentally, is the reason for the GO/265 instead of the ubiquitous 200. Absolute program addresses are not listed; instead they are listed relative to the start address of each module in the extreme righthand field. All labels, except local module labels, are listed at the end of assembly. 2K-SA, although providing many of the highly sophisticated utilities of a full scale assembler, has no comment field, but the expanded module headings should provide ample clarification. The mnemonics are almost identical to those used by MICRO, incorporating the addressing mode information in the opcode rather than with the operand. The few extremely minor discrepancies should be very easily comprehensible.

This routine exploits SYM's 6532 RIOT to control the onboard beeper speaker. If you like, you can eliminate the 10 bytes marked * in module PLAY since they are superfluous with the beeper. If, however, the poor tone is disconcerting, a miniature offboard loudspeaker can easily be hooked up to one of the 4 onboard buffers connected to 6522 VIA #3. I tried a very cheap dynamic microphone capsule in lieu of speaker and obtained very pleasing results with no additional amplification. Alternatively, use any of SYM's three 6522's to connect up to an external audio system. If you use a 6522, the following alterations must be made to module PLAY. Disable the onboard beeper by eliminating the 5 bytes starting at (BEEP). Substitute the appropriate 6522 for the 6532 by changing the 5 bytes controlling DDRB and the 5 bytes switching on and off at ORB in two places, for their 6522 equivalents. With small changes in programming, four different colored LEDs can be driven direct from the aforementioned buffers for an improved display.

For any who may need it, here are a few guidelines for adapting the modules for KIM or other 6502 microprocessors:

WAIT: No alteration should be necessary except substituting for DBOFF.

BLANK: Use the appropriate routine for your microprocessors display.

PLAY: Whatever routines are necessary to produce the audio frequencies and operate the display LEDs. Eliminate (BEEP).

FIXPLY: No alteration should be necessary.

INIT: Eliminate ACCESS. RAND will doubtless need modifying to produce the pseudo-random tune.

MAIN: Includes some specific SYM monitor routines which must be replaced by their equivalents or near equivalents in other systems. DISBUF to DISBUF+5 is SYM's 7-segment display buffer in System-RAM. DELAY displays the segment code contents of this buffer for a period of time determined logarithmically by the factor pre-stored into TV. GETKEY scans the display continuously from the display buffer while waiting to pick up the keypad input as an ASCII in the accumulator. OUTBYT pushes the segment codes of the 2 hex digits in the accumulator into DISBUF, from the right.

Only after finishing this program did I discover that Steve Ciarcia had published an article about 15 months ago in *BYTE* of April 1979. His program however is in BASIC, and concentrates particularly on the *hardware* details for the colored lights and tone generator of the player console. If you want a real professional SIMON, I strongly recommend this article.

In conclusion, can you remember and play a 70 note tune without goofing? I'm sure I can't! I'd need a "computer" for that!

Len Green was born and educated in London. He has taught physics at almost all levels in Israel, in Hebrew, for the past 30 years. He served as a UNESCO educational expert in this field from 1963 to 1967 in two West-African countries. He has written two physics text books, as well as a previous article for MICRO.

MICRO

```
0285 8501             STA SCORE
0287 200402    RUN    JSR WAIT
028A A601             LDX SCORE
028C E402             CPX TARGET
028E F009             BEQ HOME
0290 E8               INX
0291 8A               TXA
0292 8501             STA SCORE
0294 20FA82           JSR OUTBYT
0297 1016             BPL PLAYON
0299 E602      HOME   INC TARGET
029B A9F6             LDA #$F6
029D A003      MESSGE LDY #$03
029F 9940A6    CHARS  STA DISBUF,Y
02A2 88               DEY
02A3 10FA             BPL CHARS
02A5 A90C             LDA #$0C
02A7 8D56A6           STA TV
02AA 205A83           JSR DELAY
02AD 10D4             BPL MAIN
02AF A900      PLAYON LDA #$00
02B1 201602    NOTES  JSR BLANK
02B4 B507             LDA TUNE,X
02B6 205902           JSR FIXPLY
02B9 200402           JSR WAIT
02BC E8               INX
02BD E401             CPX SCORE
02BF D0F0             BNE NOTES
02C1 A200             LDX #$00
02C3 201602    TRIES  JSR BLANK
02C6 20AF88           JSR GETKEY
02C9 38               SEC
02CA E930             SBC #$30
02CC D507             CMP TUNE,X
02CE D00A             BNE GOOF
02D0 205902           JSR FIXPLY
02D3 E8               INX
02D4 E401             CPX SCORE
02D6 D0EB             BNE TRIES
02D8 10AD             BPL RUN
02DA A9FF      GOOF   LDA #$FF
02DC 202102           JSR PLAY
02DF C602             DEC TARGET
02E1 A9D3             LDA #$D3
02E3 30B8             BMI MESSGE
```


MR. RAINBOW
presents our valuable free catalog (over 100 pages). He PROMPTS you to PEEK at the latest collection of software and hardware products for your APPLE II™
A STELLAR TREK
the definitive Hi-Res color version of the classic Startrek game. Three different Klingon opponents. Many command prerogatives from use of weapons to repair of damages. Needs 48K Applesoft ROM.
Disk...$24.95
VERSAWRITER II
A drawing tablet, simply plugs into your game I/O port. Trace, draw, design, or color any type of graphic. Adds words to pictures. Creates schematics. Computes Distance/Area of any figure. New - fill any area on the screen in seconds with over 100 different and distinct colors. Needs 32K Applesoft ROM and disk drive. A bargain at...
$249.95
BOWLING DATA SYSTEM
This data mangement program provides accurate record keeping and report generation for bowling leagues of up to 40 teams with 6 bowlers per team. Needs 80-column printer, 32K Applesoft ROM.
Disk...$79.95
SUPER SOUND
Musical rhythms, gunshots, sirens, laser blasts, explosions...add these and many more exciting sounds to your Apple. Use them in your programs, or create your own SUPER SOUNDS. Needs 16K Applesoft.
Have a blast for only
$12.95...Tape
$16.95...Disk
ADD $2.00 U.S. $10.00 FOREIGN FOR SHIPPING
CALIFORNIA RESIDENTS ADD 6% SALES TAX
Open Tues. - Sun.
Don't see what you want here, then write or call today for your free catalog. We're saving one just for you.
RAINBOW COMPUTING INC.
GARDEN PLAZA SHOPPING CENTER
9719 RESEDA BOULEVARD DEPT. 1MI
NORTHRIDGE, CALIFORNIA 91324
PHONE (213) 349-0300
Visa/Mastercharge welcome.

Rapid Bubble Sort of Numerical Elements Using BASIC/ASL

The implementation of sorting routines, e.g., bubble sort, solely by means of BASIC (Applesoft) generally leads to long sorting times, especially when the amount of data is large. Using 8-bit numeric elements as data, this article shows that by using assembly language (ASL) for comparisons and swapping during sorting, as part of a BASIC program employing a bubble sort routine, sorting times are dramatically reduced even when compared with a relatively rapid Quicksort routine which employs only Applesoft.

L.S. Reich
3 Wessman Drive
W. Orange, New Jersey 07052

There is a general need to be able to sort data, and much effort has been expended in attempts to devise efficient sorting procedures. Thus, some procedures developed were: Bubble Sort, Shell Sort, and Quicksort. (See MICRO, 13:21 and 26:13.) The efficiency of these methods increases as we proceed from Bubble Sort to Shell Sort to Quick Sort. For randomly generated data, the number of comparisons involved in Bubble Sort, Shell Sort, and Quick Sort is approximately proportional to N^2, $N^{1.5}$, and $N\log_2N$, respectively. If it is further assumed that Applesoft BASIC sorting time (AST) is related to the number of comparisons made, then as the amount of data (N) is increased, AST for Bubble Sort should be greater than that for Shell Sort, and AST for Shell Sort greater than that for Quick Sort. As N becomes large, these differences should become more significant. From the preceding, it can be readily perceived that Bubble Sort becomes relatively inefficient as compared with Shell Sort or Quick Sort when N becomes large.

Listing 1

```
0800              ;********************************
0800              ;*                              *
0800              ;*      SORT ROUTINE            *
0800              ;*                              *
0800              ;*      DR. L. S. REICH         *
0800              ;*                              *
0800              ;********************************
0300                     ORG $300
0300                     OBJ $800
0300              ;*
0300 A000         SORT   LDY #$00
0302 84DA                STY $DA
0304 A5D8                LDA $D8
0306 AA                  TAX
0307 C8                  INY
0308 B90060       NXT1   LDA $6000,Y
030B D90070              CMP $7000,Y
030E 9011                BCC NXT2
0310 F00F                BEQ NXT2
0312 48                  PHA
0313 B90070              LDA $7000,Y
0316 990060              STA $6000,Y
0319 68                  PLA
031A 990070              STA $7000,Y
031D A9FF                LDA #$FF
031F 85DA                STA $DA
0321 CA           NXT2   DEX
0322 F01C                BEQ CHND
0324 B90070              LDA $7000,Y
0327 C8                  INY
0328 D90060              CMP $6000,Y
032B 9013                BCC CHND
032D F011                BEQ CHND
032F 48                  PHA
0330 B90060              LDA $6000,Y
0333 88                  DEY
0334 990070              STA $7000,Y
0337 68                  PLA
0338 C8                  INY
0339 990060              STA $6000,Y
033C A9FF                LDA #$FF
033E 85DA                STA $DA
0340 E000         CHND   CPX #$00
0342 D0C4                BNE NXT1
0344 24DA                BIT $DA
0346 30B8                BMI SORT
0348 60                  RTS
```

Listing 2

```
5 N = 500
10  DIM X(N)
20  FOR J = 1 TO N:X(J) =  INT (255 *  RND (1) + 1): NEXT
25  PRINT "UNSORTED NUMBERS ARE: ": FOR J = 1 TO N: PRINT X(J);" ";: NEXT

30  IF N / 2 =  INT (N / 2) THEN A = N / 2:B = N / 2: POKE 216,A: GOTO 35

33 A =  INT (N / 2) + 1:B =  INT (N / 2)
34  POKE 28672 + A,255: POKE 216,A
35  FOR J = 1 TO A: POKE 24576 + J,X(J): NEXT
40  FOR J = 1 TO B: POKE 28672 + J,X(J + A): NEXT
50 HEX$ = " 300: A0 00 84 DA A5 D8 AA C8 B9 00 60 D9 00 70 90 11 F0 0F 48
      B9 00 70 99 00 60 68 99 00 70 A9 FF 85 DA CA F0 1C B9 00 70 C8 D9 0
     0 60 90 13 F0"
55 HEX$ = HEX$ + " 11 48 B9 00 60 88 99 00 70 68 C8 99 00 60 A9 FF 85 DA
     E0 00 D0 C4 24 DA 30 B8 60 ND823G"
58  FOR I = 1 TO  LEN (HEX$): POKE 511 + I, ASC ( MID$ (HEX$,I,1)) + 128:
      NEXT
60  POKE 72,0: CALL  - 144
65  CALL 768
70  PRINT : PRINT "SORTED NUMBERS ARE: ": FOR J = 24577 TO 24576 + A - 1:
      PRINT  PEEK (J);" "; PEEK (J + 4096);" ";: NEXT : PRINT  PEEK (2457
     6 + A);" ";
75  IF N / 2 =  INT (N / 2) THEN  PRINT  PEEK (28672 + A)
80  END
90  REM  LINE NUMBERS 50-60 POKE OBJECT CODE INTO MEMORY, FROM BASIC, COR
     RESPONDING TO LISTING 2
100  REM  N LIMITED TO VALUES LESS THAN 511 AND LIST ELEMENTS TO 8 BITS
110  REM  4-5K BYTES REQUIRED FOR N= 510
```

However, the algorithms in Bubble Sort for comparisons and swaps can be relatively easily written in assembly language. In this manner, the Bubble Sort sorting time should be dramatically reduced. The purpose of this article is to examine the AST for Bubble Sort in comparison with the sorting time for Bubble Sort using an Applesoft-assembly language program.

The Applesoft-assembly language program is depicted in listing 1. (The corresponding assembly language portion is shown in listing 2.) The sorting time for the Applesoft-assembly language program was taken as the elapsed time (sec.) from line number 30 up to, but not including, line number 70. Table 1 shows a comparison of approximate sorting times (sec.) between Applesoft and Applesoft-assembly language programs for various values of N using the Bubble Sort algorithm. From this table, it can be seen that as the value of N was increased from 100 to 500, the ratio (R) of sorting times of assembly language to Applesoft-assembly language increased from 13 to 89. Also, from this table, the Applesoft-assembly language sorting time is seen to be roughly proportional to N. Since, as indicated previously, AST for Bubble Sort is approximately proportional to N^2, the ratio of sorting times (column 4 in table 1) should be related to N, that is, $R = kN$. From table 1, the value of k is about 1/5. Further, it may be noted here that the Applesoft-assembly language sorting times, at $N = 100$ and 500, were found to be much shorter than the AST using Shell Sort and the relatively rapid Quick Sort. Thus, for $N = 100$, the AST for Shell Sort and Quick Sort were about 51 and 20 sec., respectively. For $N = 500$, the corresponding AST values were about 413 and 151 sec. From the preceding, when BASIC is being considered for sorting procedures, it is advisable that such procedures utilize assembly language, where feasible, for the comparison and swapping portions, especially for large values of N.

In the Applesoft-assembly language program N is limited to values less than 511 and the numerical elements to 8-bits. Also, at the maximum value of N, the program required 4 to 5K bytes.

N	AS	AS/ASL	Ratio of AS to AS/ASL
100	65	5	13
200	257	7	37
255	406	8	51
300	540	10	54
400	915	12	76
500	1423	16	89

***Table 1:* Sorting times (sec.) for AS and AS/ASL for various N-values using the BS algorithm.**

L.S. Reich is an Adjunct Professor of Chemistry at Stevens Institute of Technology. One of his current major interests is the utilization of small computers, such as the Apple II, in the solution of problems in chemistry which were once considered to be tedious and time-consuming.

MICRO

A Relocating Loader for AIM Tape

With this routine you can assemble a program at one location and load it at another.

Mel Evans
1027 Redeemer
Ann Arbor, Michigan 48103

Considering its size, the AIM monitor provides a surprising number of useful functions, and does each of them well. I suppose it was inevitable that, in packing all those good things into 4K of ROM, they had to leave out a few other good things that some users might consider essential. One of these is a relocating loader.

Suppose you have recorded the object of SUBA starting at $0F00, and now want to reload it starting at $05A2 for inclusion in another program. Or suppose you have assembled a program to tape, starting at $B000 for use in ROM, and now want to reload it starting at $2000 so you can burn the EPROM. KIM lets you do such a relocating load (by using ID = FF, as explained in the user manual). So does Apple. But AIM doesn't.

Ironically, the reason it doesn't is probably because of the nifty multi-block DUMP routine that it *does* provide. It allows you to record any number of blocks of memory, all in one cassette file, and then restore the whole thing by loading just that one file. Really fine, and well worth the extra space. But it makes a relocating loader more complicated, and there is no spare room for it (or anything else) in the monitor—that ROM is fully packed.

So how do you do a relocating load? Just load the program described below, set up your memory-block assignments in page zero, hit "F1", and it acts just like the "L" command. But instead of loading it where it came from, it puts each memory block where you assign it. Furthermore, the load blocks can be different in size from what the dump blocks were. Feel free!

0000	Number of blocks (hex)
0001-0002	START address, recorded block A
0003-0004	END address, recorded block A
0005-0006	TO address, relocated block A
0007-0008	START address, recorded block B
0009-000A	END address, recorded block B
000B-000C	TO address, relocated block B
000D-000E	START address, recorded block C
000F-0010	END address, recorded block C
0011-0012	TO address, relocated block C
.	
.	
.	

and so on, up to 14 (decimal) blocks, ending at $0054

Figure 1: **Block Assignment Format for RLOAD**

Memory-Block Assignments

A memory-block assignment is defined by three addresses: START, END, and TO. START and END are the starting and ending addresses of the memory block as recorded on tape (and where it would be restored, if you weren't doing this relocation). Remember, when you dumped to tape, the monitor asked you for FROM = and TO = ? If you recorded a single memory block, and want it restored as a single block, those addresses are now START and END. (START can be smaller than FROM, and END can be larger than TO—just be sure that START-END covers the block, and doesn't overlap some other assignment.) The new TO address is where you want the byte at START to be relocated; the rest of the block will be shifted with this same offset (positive or negative).

Example: You recorded a block of memory from 0A00 through 0B2F, and now want to reload it starting at 0400. So START is 0A00, END is 0B2F, and TO is 0400. The byte recorded from 0A00, comes back at 0400, and the last byte, from 0B2F, ends up at 052F.

Another example: You recorded a big block, from 0200 through 0FFF. It consisted of program from 0200 through 061C and data from 061D through 0FFF. You want the program back where it was, but the data moved to higher memory, starting at 2000. This takes two block assignments. For block A, START = 0200, END = 061C, and TO = 0200; so the program loads with no offset. For block B, START = 061D, END = 0FFF, and TO = 2000; so the first (061D) data byte is moved to 2000, and the rest of the data follows accordingly.

Last example: You recorded three subroutines in one file from various places in memory; SUB1 (0300-033A), SUB2 (062C-0712), and SUB3 (0516-0540). You want to bring back SUB1 and SUB2, located one after the other starting at 0400, and don't want SUB3. So the three blocks are:

> Block A: START = 0300, END = 033A, TO = 0400
>
> Block B: START = 062C, END = 0712, TO = 043B
>
> Block C: START = 0516, END = 0540, TO = XXXX

Block A loads SUB1, starting at 0400. Notice that it will therefore end at 043A. So block B loads SUB2 starting at the next location: 043B. The TO address in block C should be some part of RAM you're not using at the moment, so SUB3 will load to this "trash" location. (Don't try to load it to ROM. The loader verifies each byte after storing, and so would respond with "MEM FAIL".)

The relocating memory-block assignments are set up at the bottom of page zero in the format shown in figure 1. In location 0000, store (in hex) the number of blocks to be assigned. In the next two bytes (0001-0002) store the block A START address in the usual low-high order (line number first, then page number). Put block A END in the next pair of bytes (0003-0004) and block A TO in the next pair (0005-0006).

If you're only dealing with one block, that's it. But if you need more block assignments, just continue: block B START at 0007-0008, END at 0009-000A, and TO at 000B-000C. And so on, for as many as 14 (decimal) blocks, if you need them. Just don't get carried away and continue to $0056 or above, because that's where the program starts! And watch out, of course, for overlapping START-END blocks. Ditto for overlapping TO blocks. (Guess how I found that out?)

How the Program Works

An AIM assembly listing of the relocating loader program (RLOAD) is shown, with comments, in figure 2. I have put it in the top half of page zero so it is out of the way of incoming loads, but you can put the program anywhere (including ROM) with no changes and it will still work. Just

***Figure 2:* RLOAD Assembly Listing**

```
0800                  ;RLOAD SOURCE
0800                  ; BY MEL EVANS (MICRO #34)
0800                  ;RELOCATING LOADER FOR AIM TAPE
0800                  ; MODIFICATION OF LOAD ROUTINE
0800                  ; IN AIM MONITOR AT $E2E6
0800                  ;
0800                  ;EQUATES
0800                  WHEREI EQU $E848
0800                  INALL  EQU $E993
0800                  CLRCK  EQU $EB4D
0800                  CHEKAR EQU $E54B
0800                  ADDR   EQU $A41C
0800                  CKERR  EQU $E385
0800                  CKSUM  EQU $A41E
0800                  RBYTE  EQU $E3FD
0800                  STBYTE EQU $E413
0800                  DU13   EQU $E520
0800                  START  EQU $E182
0800                  ;ZERO PAGE
0800                  NBLOKS EPZ $0000
0800                  BSTL   EPZ $0001
0800                  BSTH   EPZ $0002
0800                  BENL   EPZ $0003
0800                  BENH   EPZ $0004
0800                  BTOL   EPZ $0005
0800                  BTOH   EPZ $0006
0800                  RADL   EPZ $00FA
0800                  RADH   EPZ $00FB
0800                  OFFSTL EPZ $00FC
0800                  OFFSTH EPZ $00FD
0800                  NCHEK  EPZ $00FE
0800                  ;
010C                         ORG $010C
010C                         OBJ $0800
010C 4C                      BYT $4C          ;CREATE JUMP
010D 5600                    ADR RLOAD        ;FOR F1 KEY
010F                  ;
0056                         ORG $0056
0056                         OBJ $0804
0056 2048E8           RLOAD  JSR WHEREI       ;GET FILENAME AND FIND FILE
0059 2093E9           LOAD1  JSR INALL        ;GET NEXT RECORD
005C C93B                    CMP #$3B         ;";"
005E D0F9                    BNE LOAD1
0060 204DEB                  JSR CLRCK        ;CLEAR CHECKSUM
0063 204BE5                  JSR CHEKAR
0066 AA                      TAX              ;SAVE # OF BYTES
0067 F038                    BEQ BPTDN        ;BRANCH IF LAST RECORD
0069 204BE5                  JSR CHEKAR
006C 85FB                    STA RADH         ;SAVE RECORD ADDRESS
006E 204BE5                  JSR CHEKAR
0071 85FA                    STA RADL
0073 A500             LOAD2  LDA NBLOKS       ;FIND BLOCK ASSIG.
0075 85FE                    STA NCHEK
0077 8A                      TXA
0078 48                      PHA              ;SAVE X
0079 A900                    LDA #$00
007B A5FB             BLKCHK LDA RADH         ;CHECK NEXT BLOCK
007D D502                    CMP BSTH,X
007F 9016                    BCC NO
0081 D006                    BNE MAYBE
0083 A5FA                    LDA RADL
0085 D501                    CMP BSTL,X
0087 900E                    BCC NO
0089 B504             MAYBE  LDA BENH,X
008B C5FB                    CMP RADH
008D 9008                    BCC NO
008F D014                    BNE YES
0091 B503                    LDA BENL,X
0093 C5FA                    CMP RADL
0095 B00E                    BCS YES
0097 8A               NO     TXA              ;BLOCK NOT FOUND YET
0098 6906                    ADC #$06
009A AA                      TAX
009B C6FE                    DEC NCHEK
009D D0DC                    BNE BLKCHK       ;TRY NEXT BLOCK
009F 00                      BRK              ;ERROR: NO GOOD BLOCK
00A0 00                      BRK
00A1 F041             BPTDN  BEQ LOAD4
00A3 F0B4             BPTUP  BEQ LOAD1
00A5 38               YES    SEC              ;BLOCK FOUND, COMPUTE OFFSET
00A6 B505                    LDA BTOL,X
00A8 F501                    SBC BSTL,X
00AA 85FC                    STA OFFSTL
00AC B506                    LDA BTOH,X
```

make sure it doesn't overlap your block-assignment list in page zero, or any of the TO memory blocks defined in those block assignments. (If you change the addresses of the block-assignment list, you will have to reassemble.)

By the way, you can use the program to move itself. Load it as-is to page zero. Then rewind the tape, and use the program to reload itself to wherever you want it.

If you compare the assembly listing in figure 2 with the monitor listing of LOAD (E2E6) you will see that RLOAD is essentially a copy of LOAD, but with a patch in the middle (between LOAD 2 and JSR RBYTE). Before each byte is read from the tape buffer and stored, the patch compares its recorded address (the address it would be returned to if you weren't relocating it), stored in RADL-RADH, with each of your assigned START-END blocks in turn, until it finds the right block. (If it doesn't find that address in any of the block assignments, it breaks to the monitor, displaying "00A0 00 BRK". This means you haven't done your homework.)

After finding the right block, the patch computes the block offset (between START and TO), adds the offset (positive or negative) to RADL-RADH, and stores the byte at the offset address (ADDR). That's all there is to it.

The added code is straightforward, except for one piece that might be worth adding to your bag of software tricks. Between BLKCHK and NO is a 28-byte routine that does a two-way double-precision compare. It checks if (two-byte) address RAD is greater than or equal to address BST, and then if RAD is less than or equal to address BEN. Try doing that with double-precision subtractions, and you will see that 28 bytes is a bargain.

```
00AE F502                SBC BSTH,X
00B0 85FD                STA OFFSTH
00B2 18                  CLC             ;APPLY OFFSET TO RECORD ADDRESS
00B3 A5FA                LDA RADL
00B5 65FC                ADC OFFSTL
00B7 8D1CA4              STA ADDR        ;AND STORE IT
00BA A5FB                LDA RADH
00BC 65FD                ADC OFFSTH
00BE 8D1DA4              STA ADDR+1
00C1 20FDE3              JSR RBYTE       ;READ AND STORE BYTE
00C4 2013E4              JSR STBYTE
00C7 E6FA                INC RADL        ;INCREMENT RCORD ADDRESS
00C9 D002                BNE RSTOR
00CB E6FB                INC RADH
00CD 68          RSTOR   PLA             ;RESTORE BYTE COUNT
00CE AA                  TAX
00CF CA                  DEX
00D0 D0A1                BNE LOAD2       ;DO NEXT BYTE
00D2 20FDE3              JSR RBYTE       ;END OF RECORD, DO CHECKSUM
00D5 CD1FA4              CMP CKSUM+1
00D8 D01B                BNE JERR
00DA 20FDE3              JSR RBYTE
00DD CD1EA4              CMP CKSUM
00E0 D013                BNE JERR
00E2 F0BF                BEQ BPTUP       ;READ NEXT RECORD
00E4 A205        LOAD4   LDX #$05        ;READ LAST RECORD
00E6 20FDE3      LOAD5   JSR RBYTE
00E9 CA                  DEX
00EA D0FA                BNE LOAD5
00EC 2093E9              JSR INALL
00EF 2020E5              JSR DU13
00F2 4C82E1              JMP START       ;RETURN TO MONITOR
00F5 4C85E3      JERR    JMP CKERR       ;CHECKSUM ERROR
```

How to Use the Relocating Loader

First, enter the program in AIM memory, using figure 2 if you want to assemble it (to tape, not memory); otherwise, use figure 3 with the "I" command (or figure 4 with the "M" command, if you're old-fashioned) and then put it on tape with the "D" command. To use it, load it back in with the "L" command. Then set up your memory-block assignments at the bottom of page zero, as in figure 1. Then hit "F1" and pretend you hit "L". If you didn't overlook something in your block assignments, it will work just like the "L" command; but when it's done, all those blocks will be where you told them to go.

If, during the load, it reverts back to the monitor with "00A0 00 BRK", it means it has found a byte with an address it can't find in any of the block assignments. Function keys, perhaps? User I/O, maybe? You have overlooked something. Look at what's in 00FA-00FB. That is RADL,RADH: the offending address. Oh, yeah, now I remember!

And In Conclusion

For a half-page program, this one does a lot. It makes AIM compatible with other 6502-based computers in its ability to shuffle code around to where it is needed. It also allows you to bypass pages zero and one (which AIM regards as its own private property) while using the AIM assembler to develop code for those pages in AIM, or code for another computer, such as KIM or Apple.

MICRO

"Unassembler" for PET

An excellent way to learn programming is to study good programs. Here's a way to convert machine-language programs into a form your assembler understands.

James Strasma
3838 Benton Drive
Decatur, Illinois 62526

By now, most 650x owners interested in machine language will have a dissassembler, and often an assembler as well. Unfortunately, the two usually don't talk to each other. Wouldn't it be wonderful if the disassembler's output could be sent to the assembler? Unknown programs could be studied with all the power of a good assembler. Addresses could be labelled, tables and messages could be made readable. Most important, the results could be saved to disk, so later work wouldn't have to start over from a new disassembly.

A program to do this is called a symbolic disassembler. For several months I have used one with my PET computer. It is extremely helpful in understanding machine language programs. With its help, I also alter programs, change addresses used, even add instructions. It is worth its weight in gold. Already, it has fixed and relocated my BASIC Programmer's Toolkit to work with Commodore's new BASIC 4.0.

My symbolic disassembler is called UNASSEMBLER24. It is a group effort. First, it was a disassembler, offered by the Silicon Valley Pet Users' Group of California. Then it became a symbolic disassembler at the hands of Bill Seiler of Commodore. Finally, I changed it to work with Carl Moser's MAE assembler.

Unassembler is written in BASIC. It uses two passes to do its work, and stores its results in a disk file. It should be adaptable to several other 650x computers, and other PET assemblers, such as MACROTEA and ASM/TED.

To use Unassembler, first load the program you wish to study. Then put it where BASIC can't interfere with it. I use Bill Seiler's 'Extramon' to block move the program to the top of memory. (Extramon is available from Programma International or the ASM/TED Users' Group.) Once it is moved, I lower the high memory pointer at 52-53 (decimal) below the program. Then I use the simple disassembler in 'Extramon' to locate parts of the program that do not disassemble correctly. I jot down the addresses of problem areas. If the program is large, I note stopping points for files. MAE's default text area is only able to hold about 1K of code at a time. Then I load Unassembler. Lines 3220+ are DATA statements that tell Unassembler where to work. You will need to change them for each new program studied. As listed here, they fit Carl Moser's excellent 'Rabbit' program, (with his permission). Both MAE and Rabbit are products of Eastern House Software.

The first DATA element, in line 3240, tells the number of data files to be written, not including control and label files. Then come data lines for the files, each with four elements. The first element tells where to find the program portion to study. The next gives the normal starting address. (These first two numbers may be the same.) The third number is the end address in the current file—in relation to the normal start address. Fourth, comes the name of the file. I number them sequentially, preceeded by 'm', for 'module'. Sections that would not disassemble correctly may be left in hex byte form. To do this, end the module name with '.w', as in line 3290. No labels will be invented for these word files. That cuts debugging time later, by preventing false labels. Finally, line 3350 names a label and a control file. The label file will hold all addresses used by the program that are outside the program itself. The control file, on assembly by MAE, will load each module in turn until all are assembled.

Once all the data elements are correct, run Unassembler. It will need several minutes to prepare the needed files in a large program like the Toolkit. On the screen, you will see the address being considered and the label number last used. Its capacity as listed is 1,000 labels, though I've increased that when necessary. When all the addresses have been checked once, Unassembler will begin to write disk files of disassembled source code, showing the name of each module on the screen as it works. During this second pass through the program, each instruction is checked against the array of label addresses. If one fits, it is noted in the file, and that address is checked off as internal by Unassembler. After all modules have been written, the label file defines the unclaimed labels.

To keep the program manageable, I left two tasks to the user. You must load each module into the assembler, do two things, and save the result in place of the original. First, number the file. Unassembler gives every line the number '0000'. MAE fixes this with the command 'nu 0 10'. Next, eliminate uneeded semicolons at the end of each line. They are there to signal to MAE the end of a line. Remove them with the edit command 'ed / ;//'. Every Unassembler file also appears to fill MAE's text area, regardless of the file length. That too is for convenience, to avoid having to count characters. Removing the semicolons sets the text file to its true size.

Occasionally, a file may refuse to renumber. If this happens, reload it, and note the end of file address listed by MAE after loading. Type 'br', to go to the monitor. Once there, list out that address. MAE recognizes end-of-file when it finds $00, three bytes after the end of a text line. Unassembler places five $00's at the end of each file to be sure, (line 1310). The fix is to be sure that the third byte after the last valid line is $00. Then jump back to MAE at $5003, and renumber.

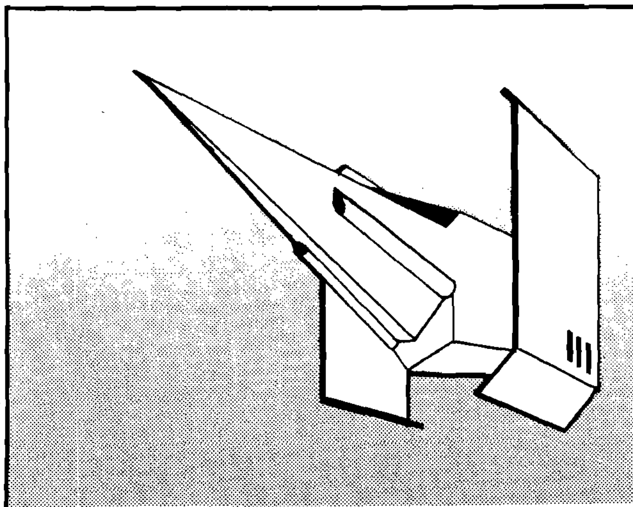
I have yet to find a program that Unassembler can't handle. For information about the ASM/TED Users' Group and the many other programs it offers in addition to Unassembler and Extramon, send a stamped reply envelope to the address at the beginning of the article.

Jim Strasma, the original sort freak, is still recruiting for his ASM/TED user group, this time by sharing one of the group's best programs. The group has two disks of programs to exchange, mostly for the PET MAE assembler.

```
100 PRINT"]QUNASSEMBLER24 FOR 650X
110 PRINT"QBASED ON A DISASSEMBLER DEVELOPED BY
120 PRINT"QTHE SILICON VALLEY PET USERS' GROUP.
130 PRINT"QORIGINAL CBM VERSION BY BILL SEILER.
140 PRINT"QALTERED FOR MAE BY JAMES STRASMA
150 PRINT"QAS OF OCTOBER 6, 1980
160 REM
170 REM WITH PROPER DISK FILES, SHOULD
180 REM WORK WITH ASM/TED & MACROTEA
190 REM
200 REM PLEASE SHARE UPDATES WITH:
210 REM  ASM/TED USERS' GROUP
220 REM  C/O JIM STRASMA
230 REM  3838 BENTON DRIVE
240 REM  DECATUR, IL. 62526
250 REM
260 PRINT"QQPUT OUTPUT DISK IN DRIVE 0 NOW.
270 REM
280 REM *** INITIALIZATION ***
290 REM
300 H$="0123456789ABCDEF":L$=CHR$(0)
310 C$=CHR$(187):MX=32767:BY=256
320 P$=CHR$(174):Q$=CHR$(167):W$=".W"
330 DIM N$(56),A(256),L$(3),T$(5)
340 DIM LB%(1000),LD%(1000),UB$(20)
350 DIM S$(20),E$(20),F$(20)
360 FOR I=0 TO 56:READ N$(I):NEXT
370 L$(0)="":L$(1)="#":L$(2)="(":L$(3)="A"
380 T$(0)="":T$(1)=",X)":T$(2)="),Y"
390 T$(3)=",X":T$(4)=",Y":T$(5)=")"
400 FOR I=1 TO 256:READ A(I):NEXT
410 LB=2:LB%(1)=-MX:LB%(2)=MX:READ FV
420 FOR I1=0 TO FV-1:READ UB$(I1),S$(I1)
430 READ E$(I1),F$(I1):NEXT:READ FF$:PRINT"]";
440 REM
450 REM *** WRITE CONTROL FILE ***
460 REM
470 OPEN 15,8,15:PRINT#15,"I0":CLOSE 15
480 READ CT$:OPEN 4,8,8,"0:"+CT$+",P,W"
490 PRINT#4,CHR$(170)L$CHR$(48)CHR$(249);
500 PRINT#4,CHR$(52)CHR$(170);:REM MAX=1K
510 PRINT#4,L$L$"; '"CT$Q$L$L$P$;
520 PRINT#4,L$L$"; ***CONTROL FILE**";
530 PRINT#4,CHR$(170)L$L$P$;
540 PRINT#4,L$L$P$L$L$" .CT "C$;
550 PRINT#4,L$L$" .BA $"S$(0)" "C$;
560 PRINT#4,L$L$" .MC $7800"C$;
570 PRINT#4,L$L$" .OS "C$L$L$P$L$L$P$;
580 PRINT#4,L$L$" .FI "CHR$(34)"0:";
590 PRINT#4,FF$CHR$(162);
600 FORI1=0TOFV-1:PRINT#4,L$L$" .FI ";
610 PRINT#4,CHR$(34)"0:"F$(I1)CHR$(162);
620 NEXT:PRINT#4,L$L$P$L$L$;
630 PRINT#4," .E"CHR$(206)L$L$L$L$L$;
640 CLOSE 4
650 GOSUB 1540:REM BUILD LABEL TABLE
660 FOR I1=0 TO FV-1:PRINT UB$(I1)"  ";
670 PRINT S$(I1)"  "E$(I1)"  "F$(I1)
680 HS$=UB$(I1):GOSUB 1460:UB=DT
690 HS$=S$(I1):GOSUB 1460:S=DT
700 HS$=E$(I1):GOSUB 1460:E=DT
710 REM
720 REM *** WRITE MODULE FILE ***
730 REM
740 OPEN 4,8,8,"0:"+F$(I1)+",P,W"
750 PRINT#4,CHR$(170)L$CHR$(48)CHR$(249);
760 PRINT#4,CHR$(79)CHR$(170);:REM MAX SIZE
770 PRINT#4,L$L$"; '"F$(I1)Q$L$L$P$;
780 PRINT#4,L$L$"; "S$(I1)" TO "E$(I1);
790 PRINT#4," "C$L$L$P$L$L$P$L$L$;
800 AC=1:OF=0:PL=LB%(AC)+MX
810 REM
820 REM *** NEXT LINE ***
830 REM
840 SS=UB+OF:I=PEEK(SS):C=A(I+1)
850 IF RIGHT$(F$(I1),2)<>W$ THEN 870
860 GOSUB 2360:GOTO 1250:REM WORD FILE
870 M=INT(C/1000):REM MNEMONIC
880 B=INT(C/100)-M*10:REM LENGTH
890 P=INT(C/10)-M*100-B*10:REM PREFIX
900 Q=C-M*1000-B*100-P*10:REM SUFFIX
910 IF C=56100 THEN M=0:REM ROR A
920 IF C=1100 THEN M=0:REM BRK
930 IF B=1 THEN 1110
940 REM
950 REM *** SECOND BYTE ***
960 REM
970 D1=PEEK(SS+1):IF B=2 THEN 1050
980 REM
990 REM *** THIRD BYTE ***
1000 REM
1010 D2=PEEK(SS+2):D1=D1+D2*256
1020 REM
1030 REM *** IF BRANCH ***
1040 REM
1050 IFM=7 THEN 1110:REM NOT RELATIVE
1060 IF M<4 THEN 1110:REM *
1070 IF M>13 THEN 1110:REM *
1080 IF D1<127 THEN 1100:REM ON
1090 D1=D1-256:REM BACK
1100 D1=S+OF+2+D1
1110 GOSUB 1950:REM TEST FOR LABEL
1120 PRINT#4," "N$(M);:REM MNEMONIC
1130 PRINT#4," "L$(P);:REM PREFIX
1140 IF M<>0 THEN 1170
```

```
1150 REM PRINT HEX OF UNUSED OPCODES
1160 DT=I:GOSUB 1370:P=1:GOTO 1190
1170 IF B=1 THEN 1250:REM 1 BYTE OP CODE
1180 DT=D1:GOSUB 1370
1190 REM P=1 MEANS IMMEDIATE MODE
1200 IF P=1 THEN PRINT#4,"$"X$;:GOTO 1250
1210 REM Z.P. MODE
1220 IF HI=0 AND B<3 AND P<>2 THEN PRINT#4,"*";
1230 PRINT#4,"Z";:REM OPERAND LABEL
1240 PRINT#4,X$T$(Q);:REM SUFFIX
1250 PRINT#4," "C$;:REM END WITH ' ;'
1260 IF M<>28 AND M<>41 AND M<>42 THEN 1290
1270 REM SPACE AFTER JMP, RTI & RTS
1280 PRINT#4,L$L$P$L$L$P$;
1290 OF=OF+B
1300 IF S+OF<=E THEN PRINT#4,L$L$;:GOTO 840
1310 PRINT#4,L$L$L$L$L$;:CLOSE 4:NEXT
1320 GOSUB 2050:REM WRITE LABEL FILE
1330 END
1340 REM
1350 REM *** DECIMAL TO HEX ***
1360 REM
1370 X$="":HI=INT(DT/BY)
1380 LO=INT(DT-BY*HI):IF HI=0 THEN 1410
1390 X$=MID$(H$,1+(240 AND HI)/16,1)
1400 X$=X$+MID$(H$,1+(15 AND HI),1)
1410 X$=X$+MID$(H$,1+(240 AND LO)/16,1)
1420 X$=X$+MID$(H$,1+(15 AND LO),1):RETURN
1430 REM
1440 REM *** HEX TO DEC ***
1450 REM
1460 DT=0
1470 L=ASC(LEFT$(HS$,1))-48
1480 IF L>9 THEN L=L-7
1490 DT=16*DT+L:IF LEN(HS$)=1 THEN RETURN
1500 HS$=MID$(HS$,2):GOTO 1470
1510 REM
1520 REM *** BUILD LABEL TABLE ***
1530 REM
1540 FOR I1=0 TO FV-1
1550 REM IGNORE WORD FILE LABELS
1560 IF RIGHT$(F$(I1),2)=W$ THEN 1610
1570 HS$=UB$(I1):GOSUB 1460:UB=DT
1580 HS$=S$(I1):GOSUB 1460:S=DT
1590 HS$=E$(I1):GOSUB 1460:E=DT
1600 GOSUB 1650:REM BUILD LABELS
1610 NEXT:RETURN
1620 REM
1630 REM *** BUILD LABELS ***
1640 REM
1650 OF=0
1660 REM
1670 REM ***NEXT LINE***
1680 REM
1690 I=PEEK(UB+OF):C=A(I+1)
1700 M=INT(C/1000)
1710 B=INT(C/100)-M*10
1720 P=INT(C/10)-M*100-B*10
1730 IF B=1 OR P=1 THEN 1900
1740 REM
1750 REM *** SECOND BYTE ***
1760 REM
1770 D1=PEEK(UB+OF+1):IF B=2 THEN 1850
1780 REM
1790 REM *** THIRD BYTE ***
1800 REM
1810 D2=PEEK(UB+OF+2):D1=D1+D2*256
1820 REM
1830 REM *** IF BRANCH ***
1840 REM
1850 IF M=7 OR M<4 OR M>13 THEN 1890
1860 IF D1<127 THEN 1880
1870 D1=D1-256
```

(continued)



```
1880 D1=S+OF+2+D1
1890 GOSUB 2220:REM PUT LABEL IN TABLE
1900 OF=OF+B:IF E<S+OF THEN RETURN
1910 GOTO 1690:REM NEXT LINE
1920 REM
1930 REM *** TEST FOR LABEL ***
1940 REM
1950 TT=S+OF
1960 IF PL>TT THEN RETURN
1970 IF PL>=TT THEN 2000
1980 AC=AC+1:PL=LB%(AC)+MX:GOTO 1960
1990 REM LABEL FIELD
2000 DT=TT:GOSUB 1370:PRINT#4,"Z"X$;
2010 LD%(AC)=1:RETURN
2020 REM
2030 REM ***WRITE LABELS FILE***
2040 REM
2050 OPEN 4,8,8,"0:"+FF$+",P,W"
2060 PRINT#4,CHR$(170)L$CHR$(48)CHR$(249);
2070 PRINT#4,CHR$(79)CHR$(170);:REM MAX SIZE
2080 PRINT#4,L$L$"; '"FF$;
2090 PRINT#4,Q$L$L$P$L$L$P$;
2100 PRINT#4,L$L$"; *** LABELS **";
2110 PRINT#4,CHR$(170)L$L$P$L$L$P$;
2120 REM SKIP INTERNAL LABELS
2130 FOR I=1 TO LB:IF LD%(I)=1 THEN 2170
2140 DT=LB%(I)+MX:GOSUB 1370
2150 REM DEFINE LABELS
2160 PRINT#4,L$L$"Z"X$" .DE $"X$" "C$;
2170 NEXT:PRINT#4,L$L$L$L$L$;
2180 CLOSE 4:RETURN
2190 REM
2200 REM ****** BINARY INSERT ******
2210 REM
2220 DL=D1-MX:T1=LB:LO=1
2230 T1=INT((T1)/2):IF T1=0 THEN 2270
2240 T2=LB%(LO+T1):IF DL<T2 THEN 2230
2250 IF DL>T2 THEN LO=LO+T1:GOTO 2230
2260 RETURN
2270 IF LB%(LO)<DL THEN LO=LO+1:GOTO 2270
2280 IF LB%(LO)=DL THEN RETURN
2290 FOR J=LB TO LO STEP-1
2300 LB%(J+1)=LB%(J):NEXT
2310 LB%(LO)=DL:LB=LB+1:DT=S+OF
2320 GOSUB 1370:PRINT "■"X$LB:RETURN
2330 REM
2340 REM *** BUILD .BY IN WORD FILE ***
2350 REM
2360 PRINT#4," .BY";:M=0:B=0
2370 IF E<=S+OF+B THEN 2390
2380 IF B<2 THEN B=B+1:GOTO 2370
2390 FOR I2=SS TO SS+B:DT=PEEK(I2):GOSUB 1370
2400 PRINT#4," $"X$;:NEXT:B=B+1:RETURN
2410 REM
2420 REM *** MNEMONICS ***
2430 REM
2440 DATA ".BY",ADC,AND,ASL,BCC,BCS,BEQ
2450 DATA BIT,BMI,BNE,BPL,BRK,BVC,BVS
2460 DATA CLC,CLD,CLI,CLV,CMP,CPX,CPY
2470 DATA DEC,DEX,DEY,EOR,INC,INX,INY
2480 DATA JMP,JSR,LDA,LDX,LDY,LSR,NOP
2490 DATA ORA,PHA,PHP,PLA,PLP,ROL,RTI
2500 DATA RTS,SBC,SEC,SED,SEI,STA,STX
2510 DATA STY,TAX,TAY,TSX,TXA,TXS,TYA
2520 DATA ROR
2530 REM
2540 REM ** MNEMONIC CODES: INCLUDE **
2550 REM ***LENGTH,PREFIX,& SUFFIX ***
2560 REM
2570 DATA 11100,35221,56100,56100
2580 DATA 56100,35200,03200,56100
2590 DATA 37100,35210,03130,56100
2600 DATA 56100,35300,03300,56100
2610 DATA 10200,35222,56100,56100
2620 DATA 56100,35203,03203,56100
2630 DATA 14100,35304,56100,56100
2640 DATA 56100,35303,03303,56100
2650 DATA 29300,02221,56100,56100
2660 DATA 07200,02200,40200,56100
2670 DATA 39100,02210,40130,56100
2680 DATA 07300,02300,40300,56100
2690 DATA 08200,02222,56100,56100
2700 DATA 56100,02203,40203,56100
2710 DATA 44100,02304,56100,56100
2720 DATA 56100,02303,40303,56100
2730 DATA 41100,24221,56100,56100
2740 DATA 56100,24200,33200,56100
2750 DATA 36100,24210,33130,56100
2760 DATA 28300,24300,33300,56100
2770 DATA 12200,24222,56100,56100
2780 DATA 56100,24203,33203,56100
2790 DATA 16100,24304,56100,56100
2800 DATA 56100,24303,33300,56100
2810 DATA 42100,01221,56100,56100
2820 DATA 56100,01200,56200,56100
2830 DATA 38100,01210,56130,56100
2840 DATA 28325,01300,56300,56100
2850 DATA 13200,01222,56100,56100
2860 DATA 56100,01203,56203,56100
2870 DATA 46100,01304,56100,56100
2880 DATA 56100,01303,56303,56100
2890 DATA 56100,47221,56100,56100
2900 DATA 49200,47200,48200,56100
2910 DATA 23100,56100,53100,56100
2920 DATA 49300,47300,48300,56100
2930 DATA 04200,47222,56100,56100
2940 DATA 49203,47203,48204,56100
2950 DATA 55100,47304,54100,56100
2960 DATA 56100,47303,56100,56100
2970 DATA 32210,30221,31210,56100
2980 DATA 32200,30200,31200,56100
2990 DATA 51100,30210,50100,56100
3000 DATA 32300,30300,31300,56100
3010 DATA 05200,30222,56100,56100
3020 DATA 32203,30203,31204,56100
3030 DATA 17100,30304,52100,56100
3040 DATA 32303,30303,31304,56100
3050 DATA 20210,18221,56100,56100
3060 DATA 20200,18200,21200,56100
3070 DATA 27100,18210,22100,56100
3080 DATA 20300,18300,21300,56100
3090 DATA 09200,18222,56100,56100
3100 DATA 56100,18203,21203,56100
3110 DATA 15100,18304,56100,56100
3120 DATA 56100,18303,21303,56100
3130 DATA 19210,43221,56100,56100
3140 DATA 19200,43200,25200,56100
3150 DATA 26100,43210,34100,56100
3160 DATA 19300,43300,25300,56100
3170 DATA 06200,43222,56100,56100
3180 DATA 56100,43203,25203,56100
3190 DATA 45100,43304,56100,56100
3200 DATA 56100,43303,25303,56100
3210 REM
3220 REM ***FILE DATA--1K LIMIT EACH***
3230 REM
3240 DATA 7:REM # OF MODULES
3250 REM
3260 REM   .MC, .BA, .EN, NAME
3270 REM
3280 DATA 7800,7000,707E,RABBIT.M1
3290 DATA 787F,707F,7086,RABBIT.M2.W
3300 DATA 7887,7087,741D,RABBIT.M3
3310 DATA 7C1E,741E,749A,RABBIT.M4.W
3320 DATA 7C9B,749B,74F6,RABBIT.M5
3330 DATA 7CF7,74F7,7500,RABBIT.M6.W
3340 DATA 7D01,7501,77FF,RABBIT.M7
3350 DATA RABBIT.LABELS,RABBIT.CT
```

MICRO

SOFTWARE FOR OSI

OSI — OSI

A JOURNAL FOR OSI USERS!!

The Aardvark Journal is a bimonthly tutorial for OSI users. It features programs customized for OSI and has run articles like these:

1) Using String Variables.

2) High Speed Basic On An OSI.

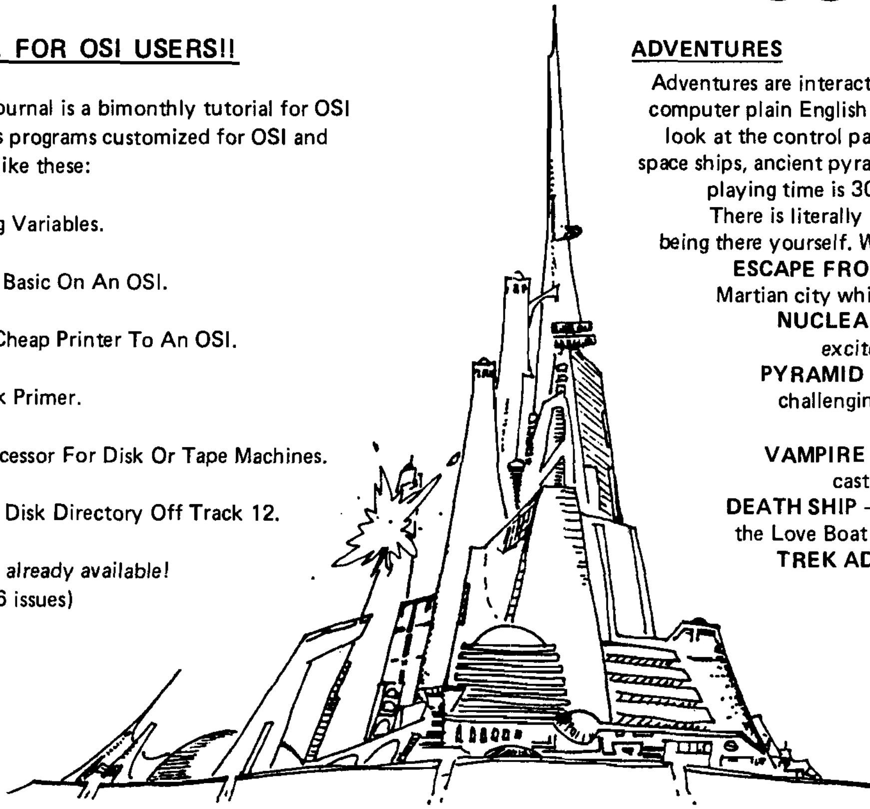
3) Hooking a Cheap Printer To An OSI.

4) An OSI Disk Primer.

5) A Word Processor For Disk Or Tape Machines.

6) Moving The Disk Directory Off Track 12.

Four back issues already available!
$9.00 per year (6 issues)



ADVENTURES

Adventures are interactive fantasies where you give the computer plain English commands (i.e. take the sword, look at the control panel.) as you explore alien cities, space ships, ancient pyramids and sunken subs. Average playing time is 30 to 40 hours in several sessions. There is literally nothing else like them — except being there yourself. We have six adventures available.

ESCAPE FROM MARS — Explore an ancient Martian city while you prepare for your escape.

NUCLEAR SUBMARINE — Fast moving excitement at the bottom of the sea.

PYRAMID — Our most advanced and most challenging adventure. Takes place in our own special ancient pyramid.

VAMPIRE CASTLE — A day in old Drac's castle. But it's getting dark outside.

DEATH SHIP — It's a cruise ship — but it ain't the Love Boat and survival is far from certain.

TREK ADVENTURE — Takes place on a familiar starship. Almost as good as being there.

$14.95 each

NEW SUPPORT ROMS FOR BASIC IN ROM MACHINES

C1S — for the C1P only, this ROM adds full screen edit functions (insert, delete, change characters in a basic line.), Software selectable scroll windows, two instant screen clears (scroll window only and full screen.), software choice of OSI or standard keyboard format, Bell support, 600 Baud cassette support, and a few other features. It plugs in in place of the OSI ROM. NOTE: this ROM also supports video conversions for 24, 32, 48, or 64 characters per line. All that and it sells for a measly $39.95.

C1E/C2E for C1/C2/C4/C8 Basic in ROM machines.

This ROM adds *full screen editing, software* selectable scroll windows, keyboard correction (software selectable), and contains an extended machine code monitor. It has breakpoint utilities, machine code load and save, block memory move and hex dump utilities. A must for the machine code *programmer replaces* OSI support ROM. Specify system $59.95

DISK UTILITIES

SUPER COPY — Single Disk Copier

This copy program makes multiple copies, copies track zero, and copies all the tracks that your memory can hold at one time — up to 12 tracks at a pass. It's almost as fast as dual disk copying. — $15.95

MAXIPROSS (WORD PROCESSOR) — 65D polled keyboard only - has global and line edit, right and left margin justification, imbedded margin commands, choice of single, double or triple spacing, file access capabilities and all the features of a major word processor — and it's only $39.95.

P.C. BOARDS

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

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

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

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

ARCADE AND VIDEO GAMES

ALIEN INVADERS with machine code moves — for fast action. This is our best invaders yet. The disk version is so fast that we had to add selectable speeds to make it playable.
Tape - $10.95 — Disk - $12.95

TIME TREK (8K) — real time Startrek action. See your torpedoes move across the screen! Real graphics — no more scrolling displays. $9.95

STARFIGHTER — a real time space war where you face cruisers, battleships and fighters using a variety of weapons. Your screen contains working instrumentation and a real time display of the alien ships. $6.95 in black and white - $7.95 in *color and sound.*

MINOS — A game with amazing 3D graphics. You see a maze from the top, the screen blanks, and then you are in the maze at ground level, finding your way through on foot. Realistic *enough to cause claustrophobia.* — $12.95

SCREEN EDITORS

These programs all allow the editing of basic lines. All assume that you are using the standard OSI video display and polled keyboard.

C1P CURSOR CONTROL — A program that uses no RAM normally available to the system. (We hid it in unused space on page 2). It provides real backspace, insert, delete and replace functions and an optional instant screen clear.
$11.95

C2/4 CURSOR. This one uses 366 BYTES of RAM to provide a full screen editor. Edit and change lines on any part of the screen. (Basic in ROM systems only.)

FOR DISK SYSTEMS — (65D, polled keyboard and standard video only.)

SUPERDISK. Contains a basic text editor with functions similar to the above programs and also contains a renumberer, variable table maker, search and new BEXEC* programs. The BEXEC* provides a directory, create, delete, and change utilities on one track and is worth having by itself. — $24.95 on 5" disk - $26.95 on 8".

AARDVARK IS NOW AN OSI DEALER!

Now you can buy from people who can support your machine.

—THIS MONTH'S SPECIALS—

Superboard II	$279
C1P Model II	429
C4P	749

. . . and we'll include a free Text Editor Tape with each machine!

Video Modification Plans and P.C. Boards for C1P as low as $4.95

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

VISA — master charge

OSI **Aardvark Technical Services • 1690 Bolton • Walled Lake, MI 48088** OSI
(313) 669-3110

Encryption with RND and USR

A simple text-encoding scheme in BASIC using the RND and USR functions.

Sherwood Hoyt
1924 S. 169th West Ave.
Sands Springs, Oklahoma 74063

Those of you who use Microsoft BASIC know of the problem it has when inputting string variables, especially if you have ever tried to write an editor in BASIC. You have to use quotes to input commas and colons. Besides, if you're dealing with more than 72 characters, which means more than one string variable, and you want to manipulate them very much, you run into the garbage collector problem. All of which makes the possibility of switching between BASIC and machine language while executing your program, very nice.

How the USR Works

The USR function serves two purposes: to switch between BASIC and machine language programs, and to pass values between them.

When the USR instruction is executed, locations $0A to $0C are used to jump to your machine language program. So put the starting address of your program in $0B (low byte) and $0C (high byte). Normally this address points to BASIC's function call error. To get back to BASIC and continue operating where you left off, use an RTS at the end of your machine language routine.

```
1  REM  CODE, BY SHERWOOD HOYT
2 :
3  REM  HIT RUBOUT TO END INPUT
4 :
10  REM  INITIALIZE USR POINTER
20  POKE 11,60: POKE 12,2
25 :
30  REM  INPUT TEXT VIA USR
40  PRINT : PRINT
50  PRINT "ENTER TEXT:"
60  PRINT
70 D =  USR (X)
75 :
80  REM  CODE SELECTION
90  PRINT : PRINT
100  PRINT "ENTER C (CODE) OR D (DECODE), AND SEED:"
110  INPUT X$,S
115 S =  - 1 -  ABS (S)
120  PRINT : PRINT
125 :
130  REM  GET CHARACTER FROM BUFFER VIA USR
140 D =  USR (X)
145 :
150  REM  CHECK FOR END OF TEXT
160  IF D = 127 THEN 90
165 :
170  REM  CHECK FOR CARRIAGE RETURN
180  IF D = 13 THEN  PRINT : GOTO 140
185 :
190  REM  RANDOM FORMULA
200 S =  INT ( RND (S) * 20 + 1)
205 :
210  REM  GOTO CODE OR DECODE SUBROUTINE
220  IF X$ = "D" THEN  GOSUB 310
230  IF X$ = "C" THEN  GOSUB 410
235 :
240  REM  PRINT CHARACTER
250  PRINT  CHR$ (D);
255 :
260  REM  LOOP BACK TO GET NEXT CHARACTER
270  GOTO 140
275 :
300  REM  DECODE SUBROUTINE
310 D = D + S
320  IF D > 90 THEN D = D - 90:D = 31 + D
330  RETURN
335 :
400  REM  CODE SUBROUTINE
410 D = D - S
420  IF D < 32 THEN D = 32 - D:D = 91 - D
430  RETURN
```

Passing Values

USR(A) is used like RND(A) or ASC(A); i.e. B=RND(A), B=USR(A); or, PRINTRND(A), PRINTUSR(A). USR(A) is also like RND(A) or ASC(A) in that the value of the variable in the parentheses (after execution) is different from the value of the formula. The variable in the parentheses contains a value that can be picked up by your machine language program. But the formula USR(A) is assigned the value that you sent back to BASIC from your machine language program. How do you send values back and forth?

When BASIC executes the USR function, the value of the variable in the parentheses is put in the FPA (floating point accumulator) at $AC-$AF. Your machine language program can pick up the value there if you want to work with it in floating point format, otherwise you must do a JSR to INVAR at $AE05, which converts the floating point representation to a fixed point representation, and puts the result in $AE and $AF, where your program can pick it up.

If you wish to send a value back to BASIC, you can do that again by sticking a floating point number in $AC-$AF. But, again, if you don't want to work with floating point, just jump to OUTVAR at $AFC1 with your 16-bit value in A (high byte) and Y (low byte), and the value will be converted and put in the FPA, where BASIC will pick it up and assign its value to the expression USR(A).

Obviously the USR function can be used for much more than just inputting strings. You can use it for any subroutine you might find easier or faster to do in machine language than in BASIC.

The BASIC Program

I've written an encoding and decoding program which will provide an example of how the USR can be used. The program encodes and decodes the ASCII characters from decimal value 32-90; from blank to capital z. I imagine we've all tried to make a code at one time or another, although most of them were probably rather simple. It's pretty easy to make a sophisticated code using the RND function, a code which probably wouldn't

```
0800                ;CODE EDITOR SOURCE
0800                ; BY SHERWOOD HOYT (MICRO #34)
0800                ;
0800                INPUT  EQU $FFEB
0800                LF     EPZ $0A
0800                OUTPUT EQU $FFEE
0800                MEMPT  EPZ $D8
0800                MEMPT1 EPZ $D9
0800                BCKSP  EPZ $5F
0800                RUBOUT EPZ $7F
0800                RETURN EPZ $0D
0800                USRPT  EPZ $0B
0800                USRPT1 EPZ $0C
0800                OUTVAR EQU $AFC1
0800                ;
023C                       ORG $023C

023C                ;
023C 205D0                 JSR STPT          ;SET MEMORY POIN
                                             TER
023F 20EBF          GETCH  JSR INPUT         ;GET CHARACTER
0242 C90A                  CMP #LF           ;CHECK FOR LINE
                                             FEED
0244 F0F9                  BEQ GETCH
0246 20EEFF         DSPCH  JSR OUTPUT        ;DISPLAY CHARACT
                                             ER
0249 C90A                  CMP #LF           ;CHECK FOR LINE
                                             FEED
024B F0F2                  BEQ GETCH
024D A200           STORE  LDX #$00          ;STORE CHARACTER

024F 81D8                  STA (MEMPT,X)
0251 C97F                  CMP #RUBOUT       ;CHECK FOR END OF
                                             TEXT
0253 D011                  BNE BKSP
0255 A995                  LDA #$95
0257 850B                  STA USRPT         ;SET USR POINTER
                                             TO
0259 A902                  LDA #$02          ;2ND PART OF PRO
                                             GRAM
025B 850C                  STA USRPT1
025D A910           STPT   LDA #$10          ;SET MEMORY POIN
                                             TER
```

```
025F 85D9                  STA MEMPT1        ;TO $1000
0261 A900                  LDA #$00
0263 85D8                  STA MEMPT
0265 60                    RTS               ;DONE
0266                ;
0266 C95F           BKSP   CMP #BCKSP        ;CHECK FOR BACKS
                                             PACE
0268 D00D                  BNE NEXT
026A C6D8                  DEC MEMPT         ;DECREMENT
026C A9FF                  LDA #$FF          ;MEMORY POINTER
026E C5D8                  CMP MEMPT
0270 D01F                  BNE LOOP
0272 C6D9                  DEC MEMPT1
0274 4C3F02                JMP GETCH
0277 E6D8           NEXT   INC MEMPT         ;INCREMENT
0279 D002                  BNE CHRET         ;MEMORY POINTER
027B E6D9                  INC MEMPT1
027D C90D           CHRET  CMP #RETURN       ;CHECK FOR RETURN

027F D005                  BNE CHMEM
0281 A90A                  LDA #LF           ;GIVE LINE FEED
0283 4C4602                JMP DSPCH         ;TO DISPLAY
0286 A91F           CHMEM  LDA #$1F          ;CHECK FOR END
0288 C5D9                  CMP MEMPT1        ;OF MEMORY
028A D005                  BNE LOOP
028C A97F                  LDA #RUBOUT       ;STORE A
028E 4C4D02                JMP STORE         ;RUBOUT
0291 4C3F02         LOOP   JMP GETCH         ;GET CHARACTER
0294 00                    BRK
0295 A200           RETREV LDX #$00          ;PULL OUT CHARAC
                                             TER
0297 A1D8                  LDA (MEMPT,X)     ;FROM MEMORY
0299 A8                    TAY
029A 8A                    TXA
029B C07F                  CPY #RUBOUT       ;CHECK FOR END OF
                                             TEXT
029D F008                  BEQ INIPT
029F E6D8                  INC MEMPT         ;INCREMENT POINT
                                             ER
02A1 D007                  BNE RTN
02A3 E6D9                  INC MEMPT1
02A5 D003                  BNE RTN
02A7 205D02         INIPT  JSR STPT          ;SET POINTER BACK

02AA 4CC1AF         RTN    JMP OUTVAR        ;RETURN CHARACTER
```

give any clues for decoding even if you wrote a book on it, and even those who know of the RND function won't know which one of the indefinitely many ways you used it to write your code.

This program needs some method of storing text, and I wanted to get around the problem of having to use quotes when inputting commas and colons. I used the USR function to go to my machine language program, which stores the text in memory and pulls it out when it's needed.

A few things about the BASIC and machine language programs need to be explained. Line 60 of the BASIC program sets the USR pointer to the start of the machine language program. Line 100 asks for "D" for decode and "C" for code. If you want to print the text as it is, just type in any letter besides D or C.

Line 100 also asks for a seed. The seed must be a negative number: −1, −2, etc. It provides the basis for the random numbers, and allows you to have a standard key to encode or decode text. In other words the RND function isn't really random, otherwise you couldn't use it for decoding. In line 200 you may wonder how I'm getting different random numbers, since the same negative argument, RND(−A), always produces the result. I start the random sequence with a negative number, S, as the seed. After the first random number is produced, S is assigned this random value, which is a positive number and is then used as the seed to generate the next random value.

In the decode and encode subroutines at 310 and 410 I add or subtact the random formula number; a number from 1-20, to or from the ASCII value for the particular character I'm working on. This can be visualized as a circle of characters, one of which is moved out of position in a clockwise direction or a counter-clockwise direction, and takes the value of the character it lands on. The alphabet is shifted by a new random value for each character of text. In line 70 we go via the USR to the machine language program and input text, and in line 140 we use the USR again to go to the machine language program. But this time the USR address is changed to point to the second part of the program which pulls characters of text out of memory and sends them to BASIC. Let's take a look at the machine language program.

The Machine Language Program

At $023C, the first line, we jump to a subroutine that sets the memory pointer to $1000 where text is stored. $023F jumps to the INPUT routine to pick up a character. That character is displayed by jumping to the OUTPUT routine, $0246. Line-feeds are ignored by jumping back to the INPUT line at $023F, so that the return character can be used both as a return and line-feed. Next the character is stored in memory: $024D and $024F. Then we check to see if the character is a rubout character signaling the end of text. If it is, the pointers are set to their proper values,from $0255 to $0265, and we return to BASIC. Then we check for a back-space character and if it is one, we decrement the memory pointer, and jump back for another character; $0266 to $0274. If the character wasn't a back-space character then we increment the memory pointer; $0277 to $027B. From $027D to $0291 we check for a return. If it is a return we go back to display a line-feed, if not we check for the end of memory, which I've put at $1F00. If we've run out of room, we store a rubout at the end of memory, set the pointers, and return to BASIC. Otherwise we loop back to get another character.

The second part of the program pulls out characters from memory one-by-one and sends them back to BASIC. $0295 to $0297 pulls a character out. $0299 to $029A transfers the character to Y and transfers $00 to A, to prepare for sending the character to OUTVAR; the low byte must be in Y. In $029D we check for the end of text. If it is the end of text we do a JSR to set the pointers back, if not we increment the memory pointer, and jump to OUTVAR.

RUN

Now what does this code look like when printed? About as senseless as a monkey typing on a typewriter. Here's an example. Type this in:

```
HELLO,  READER::
THIS IS COLON TOWN
::::::::::::::::::
AND COMA TOWN,,,,,
```

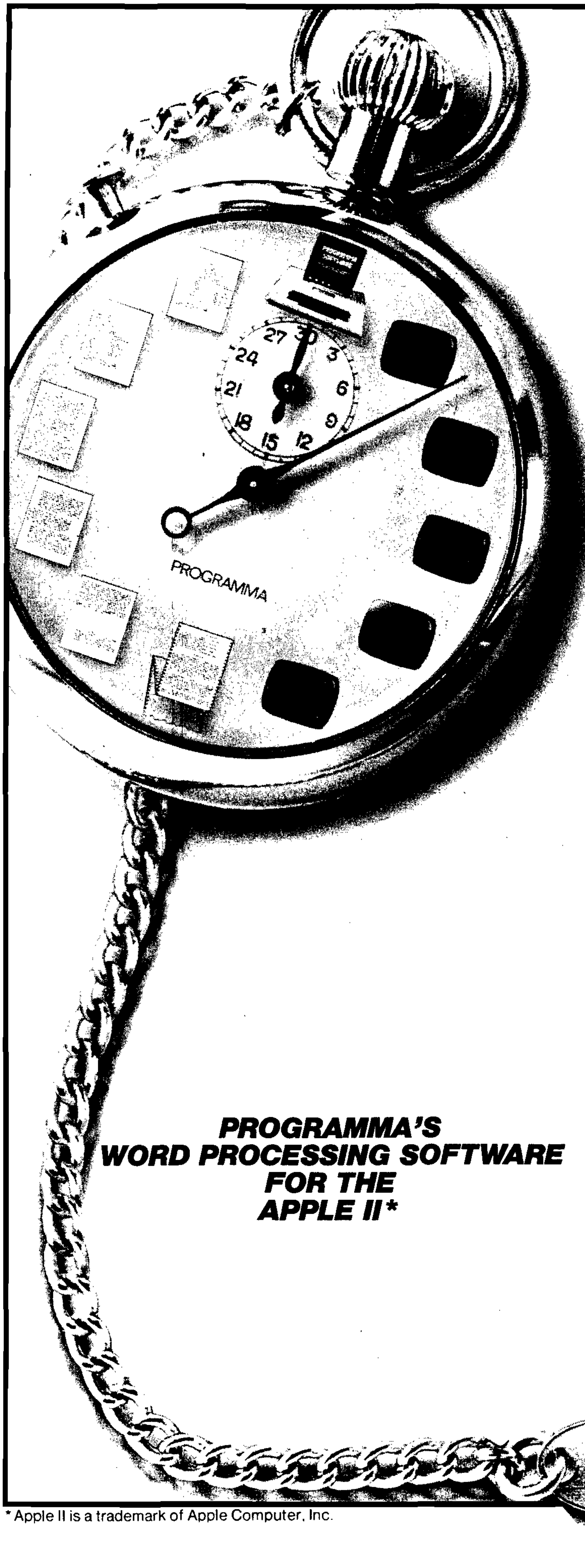
And tell the program to encode it with a seed of −1.

```
GA@:H#IQC4;42I3.
J5EDHFHX1@8CMTOAOL
/5)+24(06,9-*'-&O3
8;CS8??:QCEJ;"U+*'
```

If you wish to decode it, just type the monkey characters in, and tell the program to decode it with a seed of −1.

MICRO

REAL TIME SAVER



Minutes are money. So, when it's possible to easily handle many times your workload, with better results, you're on to something.

And, that something is Programma's versatile and powerful Word Processing System. Because it's designed from the user's viewpoint, it's easy to work with. You can start right out doing basic word processing. And, since each step logically leads to the next, you'll be handling even complex work problems in a very short time.

Our WPS consists of two superior programs, Apple PIE (Programma Improved Editor) and FORMAT. PIE is a free-form, live-screen-oriented editor, for creating and editing text for processing. The commands let you enter or alter anything, anywhere on your screen. You can search and replace, add, delete, move, or insert, by character, word, line or paragraph. And you know what's happening, as you see it right on the screen. PIE also allows use of a lower case adapter, and can even be used as a software development tool.

The other half of our real time saver is FORMAT. It uses simple, embedded codes to set formats for letters, manuals, scripts, documents or catalogs to your specifications.

You want this centered?...that underlined?... those areas indented?...something paragraphed? You make the decisions, FORMAT does the work immediately, accurately. Then, if before printing, you want to revise copy, or change a document's final appearance, you can do it with ease. FORMAT is very forgiving!

To make the most of your time, order today. Our WPS for the 32K Apple II, on mini-floppies, with thorough documentation, is now just $129.95. At your local computer store. Or direct from Programma, 2908 No. Naomi Street, Burbank, CA 91504. (213) 954-0240.

PROGRAMMA'S WORD PROCESSING SOFTWARE FOR THE APPLE II*

* Apple II is a trademark of Apple Computer, Inc.

Automatic Keyboard

Making changes in a BASIC program while running it—that's what the Automatic Keyboard is all about.

Theo Schijf
Noorddammerweg 50
1187 ZT AMSTELVEEN
The Netherlands

The automatic keyboard can be used for program-writing programs—for instance, a program that plots mathematical functions. In time it can be taught how to plot more and more different functions. All functions will be part of the BASIC program, but still the person who takes care of the input, does not have to be familiar with BASIC.

It can also be used for computer-aided programming. Anyone can write his own compiler (in BASIC, to BASIC). Also, BASIC could easily be translated into any language with no increase in execution time, as the internal representation would be the same.

You may also want to use the automatic keyboard to create programs that will update or change existing programs. For instance, an accounting program-writing program could give anyone the capability of writing his own accounting program. One would begin with this accounting program-writing program and no knowledge of BASIC, and the result would be an accounting program specified to one's own needs.

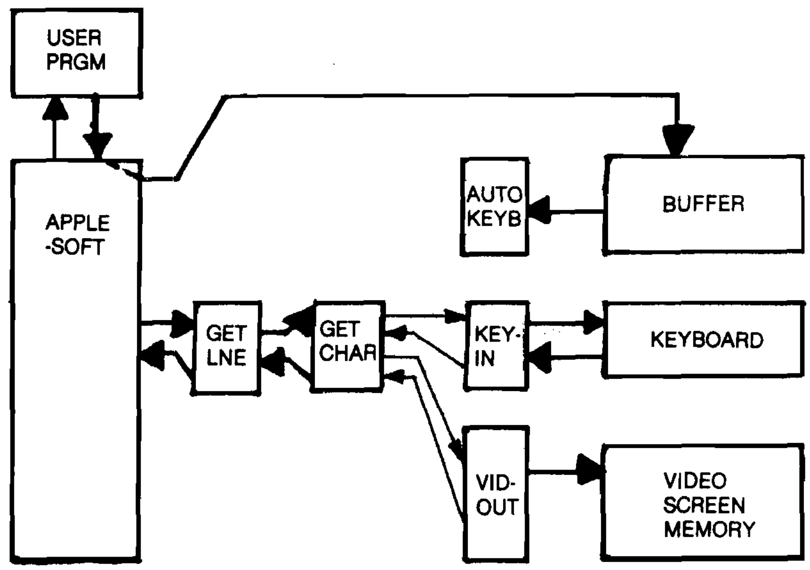


Figure 1: **Normal Mode**

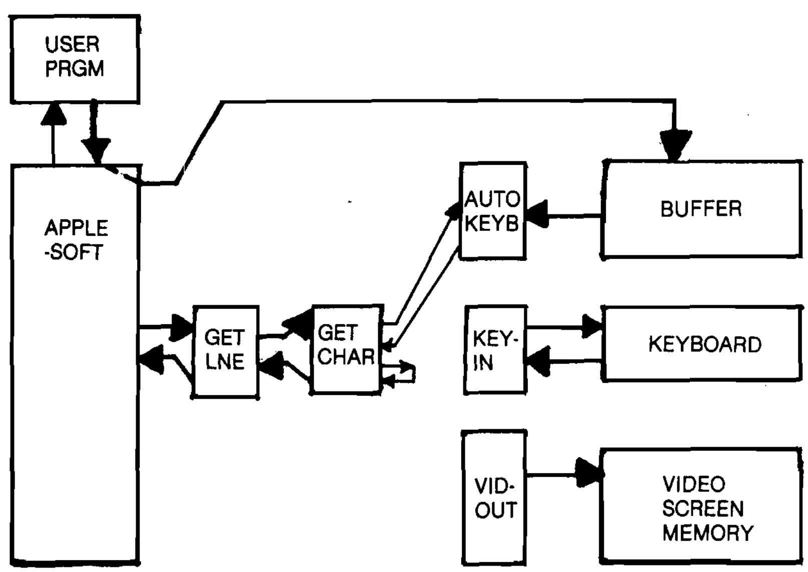


Figure 2: **AK Switched On**

A BASIC program could be transferred easily from one personal computer to another. One would only have to take care of the 'clear from the outside' differences. The complex operation of the different BASIC interpreters, of different types of personal computers, would not have to be considered.

How?

Let's consider AB (Applesoft floating point BASIC). AB prompts a ']' and waits for input. It continues as soon as it gets a <RETURN>. Then it does whatever it is asked to do and it prompts the ']' again. The KEYIN routine (and thus the GETCHAR ROUTINE) loops until a keyentry is detected, the GETLINE routine loops until a <RETURN> is received from GETCHAR. (See figure 1.)

The AK-INIT routine (see listing Automatic Keyboard) replaces the keyin pointer, so that any input request will be dealt with by the AK routine (keyin pointer: zero page: 56,57). (See figure 2.)

From this moment on, the GETCHAR routine gets its characters from AK, and AK gets them from a buffer. Data can be entered into this buffer from a BASIC program, also the switching on and the switching off can be done from a BASIC program. The AK-OFF routine switches the KEYIN pointer back to normal.

Subroutine 900 loads the AK into memory. AK uses locations 768-862, while the buffer uses 512-767. No HIMEM or LOMEM adjustments are necessary; the DATA statements should be the last ones in the program.

Subroutine 500 puts A$ into the buffer. It is important that:

1. Characters like 'return', '' '' etc. are included in A$.
2. The last part of A$ should be:
   - ''RUN'', ''RUN 110'', etc. or
   - ''GOTO 110'' or
   - ''CALL 842''
3. After 'GOSUB 500' there must be a 'CALL 795' to switch on AK and an 'END' to get back to programming mode.

```
100  REM
101  REM  CHANGE NAME (TS800415)
102  REM
105  GOSUB 900: REM  LOAD AK
107  GOTO 120: REM  DON'T SWITCH OFF BEFORE SWITCH ON
110  CALL 842: REM  AK-OFF
120 NM$ = "NONAME"
125  HOME : PRINT "MY NAME IS ";NM$
130  PRINT : LIST 100,125: PRINT
140  PRINT : INPUT "NEW NAME PLEASE ";NM$
150  IF  LEN (NM$) = 0 OR  LEN (NM$) > 28 THEN 140
160 A$ = "120 NM$=" +  CHR$ (34) + NM$ +  CHR$ (34) +  CHR$ (141)
170 A$ = A$ + "RUN 110" +  CHR$ (141): REM  TO RESTART PROGRAM
180  GOSUB 500: REM  TO PUT A$ INTO BUFFER
190  CALL 795: REM  AK-INIT
200  END : REM  FROM NOW ON KEY-INPUT FROM BUFFER, NO OUTPUT TO VIDEO
500  REM
501  REM A$ INTO BUFFER
502  REM
510  FOR N = 1 TO  LEN (A$)
520 ADS = 511 + N: POKE ADS, ASC ( MID$ (A$,N,1))
530  NEXT N
540  RETURN
900  REM
901  REM  LOAD AK
902  REM
910  RESTORE
920  READ H$: IF H$ <  > "*AK*" THEN 920
930  FOR  N = 768 TO 862
940  READ H: POKE N,H
950  NEXT N
960  RETURN
965  DATA  *AK*
970  DATA  173,0,2,72,173,1,3,24,105,1,141,1,3,173,2,3,105,0,141,2,3,104,
     96
980  DATA  0,0,0,0
985  DATA  165,54,141,23,3,165,55,141,24,3,165,56,141,25,3,165,57,141,26,
     3,169,2,141,2,3,169,0,141,1,3,169,3,133,57,169,0,133,56,169,3,133,55
     ,169,22,133,54,96
990  DATA  173,23,3,133,54,173,24,3,133,55,173,25,3,133,56,173,26,3,133,5
     7,96
```

```
0800                 ;******************************
0800                 ;*                            *
0800                 ;*    AUTOMATIC KEYBOARD      *
0800                 ;*                            *
0800                 ;*      BY THEO SCHIJF        *
0800                 ;*                            *
0800                 ;*    MICRO #34-MARCH 1981    *
0800                 ;*                            *
0800                 ;******************************
0800                 ;*
0800                 ;*
0800                 BUFFER EQU $200
0300                        ORG $300
0300                        OBJ $800
0300 AD0002          ENTRY  LDA BUFFER        ;GET CHARACTER FROM BUFFER
0303 48                     PHA
0304 AD0103                 LDA ENTRY+1       ;LOW-BYTE BUFFER ADDRESS
0307 18                     CLC
0308 6901                   ADC #$01
030A 8D0103                 STA ENTRY+1
030D AD0203                 LDA ENTRY+2       ;HIG-BYTE BUFFER ADDRESS
0310 6900                   ADC #$00
0312 8D0203                 STA ENTRY+2
0315 68                     PLA
0316 60              NOPR   RTS               ;RETURN FROM ENTRY
0317 17              R1     BYT *
```

```
0318 18         R2     BYT *
0319 19         R3     BYT *
031A 1A         R4     BYT *
031B A536       INIT   LDA $36
031D 8D1703            STA R1
0320 A537              LDA $37
0322 8D1803            STA R2
0325 A538              LDA $38
0327 8D1903            STA R3
032A A539              LDA $39
032C 8D1A03            STA R4         ;POINTERS SAVED
032F A902              LDA /BUFFER
0331 8D0203            STA ENTRY+2    ;RESET HIGH-BYTE BUFFER ADDRESS
0334 A900              LDA #BUFFER
0336 8D0103            STA ENTRY+1
0339 A903              LDA /ENTRY
033B 8539              STA $39        ;SET HIGH-BYTE NEW KEYIN POINTER
033D A900              LDA #ENTRY
033F 8538              STA $38        ;SET LOW-BYTE NEW KEYIN POINTER
0341 A903              LDA /NOPR
0343 8537              STA $37        ;SET HIGH-BYTE NEW VIDOUT POINTER
0345 A916              LDA #NOPR
0347 8536              STA $36        ;SET LOW-BYTE NEW VIDOUT POINTER
0349 60                RTS            ;RETURN FROM INIT
034A AD1703     OFF    LDA R1
034D 8536              STA $36
034F AD1803            LDA R2
0352 8537              STA $37
0354 AD1903            LDA R3
0357 8538              STA $38
0359 AD1A03            LDA R4
035C 8539              STA $39        ;ALL POINTERS RESTORED
035E 60                RTS            ;RETURN FROM OFF
```

AK should *never* be switched *off* before it has been switched *on*. If AK is switched on, there will be no video output at all, and also if an 'INPUT' or a 'GET' statement is encountered, it will receive a lot of nonsense from the buffer and then anything can happen.

Subroutines 500 and 900 can be copied (and renumbered) and used in any other program. Note that problems might occur when 'renumber' is used —line numbers between quotes (see line 160) are not affected.

During execution of the AUTOMATIC KEYBOARD program, the video output is suppressed in order not to disturb BASIC program execution.

Theo Schijf is currently studying electronics in Delft. He has interfaced his Apple to an IBM Selectric via the game paddle interface. He has also written some software programs in Dutch.

MICRO™

MICRO Club Circuit

Mike Rowe
Club Circuit
P.O. Box 6502
Chelmsford, MA 01824

The following club announcements are presented in zip code order.

Amateur Computer Group
This New Jersey personal computer users group has a membership of approximately 1,300 with Sol Libes as President/Secretary. They hold three meetings per month on the first, second and fourth Friday of each month. Contact:

Amateur Computer Group
UCTI
1776 Raritan Road
Scotch Plains, New Jersey
07076

Apple Group — N.J.
This club meets the 4th Friday of each month at 7:00 p.m. at U.C.T.I., 1776 Raritan Road, Scotch Plains, N.J. The approximate number of members is 100-150. The aim of the club is to exchange information, help beginners, and distribute user contributed software. Contact:

Steve Toth, President
1411 Greenwood Drive
Piscataway, New Jersey 08854

SARDEGNA Computers
The purpose of this club is to introduce microcomputers to the Italian Culture. President is Blake Etem. Meetings are held each Wednesday at 11 a.m. at Decimomannu AFB in Caguari-Sardinia, Italy. For more information, contact:

Dave Kaufman
Box 3, DET 4 40th TACG
APO New York, New York
09161

Apfelsaft
This group of approximately 20 members consists of servicemen and locals in Germany. They meet the third Tuesday of each month to exchange programming ideas, programs and information about the Apple. For more information contact:

SP6 David Lee Powell
270th Sig. Co., Box 1157
APO New York, New York
09189

OSIO
OSIO is a nonprofit organization of owners of Ohio Scientific computers with over 350 members, some in countries other than the U.S. There is a monthly Newsletter and exchange of nonproprietary software. Objectives: "To study, advance, and promote the application of computers; to sponsor conferences, workshops, symposia, demonstrations, and publications..." Central group, with about 120 members in Virginia, Maryland and the District of Columbia, meets at 7:30 p.m. on the first Tuesday of each month, usually at the Walter Johnson High School in Rockville, Maryland. Contact:

Wallace Kendall
9002 Dunloggin Rd.
Ellicott City, Maryland 21043

Carolina Apple Core, Inc.
Joe Budge is president of this club with the purpose of education on applications for Apple computers. Meetings are held in Durham, NC in odd numbered months, and in Raleigh, NC in even numbered months. This club supports approximately 78 members. Frank Barden is their newsletter editor. Contact:

Carolina Apple Core, Inc.
P.O. Box 31424
Raleigh, North Carolina 27622

Behavioral Sciences AIM-65 Users Group
Workers in the behavioral and biological sciences who are currently using, or are interested in using the AIM-65, are invited to participate in a users group now forming. Areas of interest include hardware and software for experimental control, data acquisition, statistical analyses, and other applications. If interested, please write, outlining areas of interest, current and planned projects, etc., to:

Dr. J.W. Moore, Jr.
Box 539
MTSU
Murfreesboro, Tennessee 37132

Louisville Apple Users Group
Mike Finn is president of this group which meets the 1st Sunday and 3rd Tuesday of each month. The 80+ members exchange knowledge about the Apple so all users may get the maximum benefit from their machines. The club is willing to exchange newsletters with other clubs. For further information contact:

Patrick J. Connelly
3127 Kayelawn Drive
Louisville, Kentucky 40220

Central Ohio Apple Computer Hobbyists
Meets the 3rd Saturday of each month. Thomas Mimlitch is president. There are 35 members. The purpose of this group is self education and to promote interest in home computers. For more information, contact:

David Reinoehl
1357 Bernhard Rd.
Columbus, Ohio 43227

Apple-Sider's Club of Cincinnati
Meets the 2nd Tuesday of each month at 7:30 p.m. at the University of Cincinnati Medical Center, Cincinnati. "POKE-APPLE" is published the first week of each month. The purpose of the group is to share Apple experiences. President is Gary Johnston. Membership is 178. For more information, contact:

W.M. Fowee
1074 Brooke Ave.
Cincinnati, Ohio 45230

The CCC1P'ers
Located in Upper Michigan, Copper Country, this group's aim is experimentation, programming hardware modifications, amateur radio. Meetings are held at the Dollar Bay High School in Dollar Bay the 1st and 3rd Tuesday of each month at 7:30 p.m. Membership is at 12 with Scott Anderson as president. Please contact:

Gregory S. Anderson
119 South Iroquois
Laurium, Michigan 49913

Iowa City Apple Users Group
The purpose of this group is to provide mutual support, information and encouragement to Apple users in the Eastern Iowa area. Meetings are held at Westinghouse Data Score Systems, I-80 and Iowa Rt. 1 in Iowa City the third Tuesday of the month. (Executive Committee meets the second Tuesday.) There are approximately 35 members. For more information, contact the president evenings.

David B. Thomas
134 Ravencrest Drive
Iowa City, Iowa 52240
319/351-3368

Chicago Area Computer Hobbyist Exchange
CACHE meets monthly on the third Sunday at 11:00 a.m. at the DeVry Technical Institute, Belmont Avenue at Campbell in Chicago. There are 50 members. Jeff Fisher is president. CACHE also has special interest groups for other types of processors (total membership is over 400). The purpose

of the group is to advance knowledge of the Apple computer, available hardware and software, etc. There is also an extensive program library. For further information contact:

Timothy Clark (Librarian)
18w145 Belair Court
Darien, Illinois 60559

Central Illinois Apple
The 50 members of this club meet the 2nd Tuesday of each month at 6:30 p.m. at the Peoria Public Library, Peoria. Mike Still is president of the club which puts out a monthly newsletter. The purpose of the group is to share experiences, information and software among new and old Apple users in the business, personal and educational fields. For information contact:

David M. Crull, Secretary
1824 Hoover Drive
Normal, Illinois 61761

Micro & Personal Computer Club of St. Louis
Rick Connoly is president of this group which meets to gather and diffuse knowledge of software development by members and commercials. There are approximately 25 members. For more information contact:

Kunihiro Tanaka, Secretary
3268 Watson Road
St. Louis, Missouri 63139

Santa Barbara Apple User's Group
This group, formed to exchange information and provide user support, has a membership of 60. Ed Adams is club President, Maisie Cohen, Secretary. Their meetings are held on the first Wednesday of each month, 7:30 p.m., at The Computer Shop. For further information, contact:

Santa Barbara Apple User's Group
2007 State Street
Santa Barbara, California 93105

Forth Interest Group
Meets on the fourth Saturday at Noon. Membership is over 1200. The club puts out a publication called "Forth Dimensions." For more information, contact:

Jim Flournay Ancon
17370 Hawking Lane
Morgan Hill, California 95037

Apple Sac
This club consists of approximately 160 members with meetings on the first Tuesday and third Wednesday of each month. Ken Gray is President, Jim Henry, Secretary. Their purpose is to provide a common meeting place to exchange ideas and techniques, and to provide guest speakers. A newsletter and program library are provided with membership. For more information contact:

Apple Sac
8074 Ruthwood Way
Orangevale, California 95662

Salem Area Computer Club
This club, with 70 members, meets at the Salem Academy Library the first Monday of each month. Kenneth Ernst is President, Leonard Oswald, Secretary. The purpose of the club is fellowship, enrichment, and assistance. For further information contact:

Salem Area Computer Club
P.O. Box 7715
Salem, Oregon 97303
393-1173

NW PET User Group
This club of approximately 45 members meets the 2nd Tuesday of the month at 7:30 p.m. at the U of W Academic Computer Center, 3737 Brooklyn, Seattle. President is Richard Ball. The purpose of the group is to share and aid members with knowledge of programming, hardware and all PET/CBM uses. For more information write:

NW PET User Group
2564 Dexter N. #203
Seattle, Washington 98101

The Apple Cobblers
This user group is comprised mostly of educators, and is an associate member in the International Apple Corps. Presently they meet in Lacey, Washington at the North Thurston High School, 600 Sleater Kinney Road, NE. This group began with approximately ten members and is steadily growing, sharing their ideas on uses of the Apple in the classroom. For more information, contact:

Rodney Taylor
3920 56th Avenue NE
Olympia, Washington 98506

London Apple Corps
Members meet the first Tuesday of the month at 7:30 p.m. at Lyons Logic, 296 Horton St., London, Ontario. There are approximately 20 members who meet for the purpose of making more beneficial use of personal computers through idea sharing and improved programming. Contact:

John Forristal (President)
296 Horton St.
London, Ontario
Canada N6B 1L4

CEBUSTACK
The Central Bureau of Statistics Computer Klub is located in Voorburg, the Netherlands. Hardware meetings are on the 2nd and 4th Thursday of the month. Other meetings are once a month at 7:30 p.m., CBS, pr. Beatrixloan 428, Voorburg. Publication: Stackpointer. For further information, contact:

J. de Jong
Hordykstr. 111
25g3 HC Den Haag
The Netherlands

OSI Users Group (Auckland)
This group of 53 members meets the 3rd Tuesday of each month at 7:30 p.m., V.H.F. Clubrooms, Mt. Roskill, Auckland. The purpose of the group is hobby and general applications of OSI microcomputers, and general and technical information exchange. For more information, contact:

Brian Wilson
88 Stanley Road
Glenfield, Auckland,
New Zealand
Ph. 4443123

PTC Update
Membership consists of 120 people from all over Germany, Austria, and Switzerland. There are no general meetings, but many members hold local gatherings for their own interests. Their club newsletter, the "PTC-info" is published quarterly, in German. Their PET/CBM program library contains nearly 500 programs. For more information, contact:

Dietmar Severitt
Gutenbergstr 20
D-6052
Muhlheim, West Germany

Computer Programs for Investment Management
If you're an investor, how do you obtain professional-quality investment management programs? One way is to belong to a professional, nonprofit group called the MicroComputer Investors Association (MCIA). Since 1977 the Association has published a journal, *The MicroComputer Investor.* Their journal contains a wealth of information and programs for investors who use microcomputers. If you would like to obtain a membership application and an index of all programs and articles published to date, send $3.00 for an information packet to:

Jack Williams, MCIA
902 Anderson Drive
Fredericksburg, Virginia 22401

MICRO Challenges

By Paul Geffen

This column will appear regularly and contain news of interest to the community of Ohio Scientific users.

The M/A-COM Connection

On December 5, Ohio Scientific, Inc. was bought by M/A-COM of Burlington, Mass. Since its founding in 1975, OSI has grown into a $30 million-a-year operation employing about 250 people at five sites.

M/A-COM is a holding company, formed in 1978 when Microwave Associates bought Data Communications Corporation. M/A-COM has since bought six other companies and has agreed to buy three more in the near future. These companies include: Lawrence Laboratories, a casting house; Omni Spectra, a maker of microwave connectors; LINKABIT, a producer of digital signal processors; Valtec, a supplier of coaxial cable to the CATV industry; Prodelin, a maker of antennas and waveguides; and now Ohio Scientific, a leading producer of microcomputers and systems. Future acquisitions may include Microwave Power Devices, Alanthus Data Communications and Power Hybrids, Inc.

M/A-COM is pursuing a policy of vertical integration, buying out suppliers in the field of microwave communications. M/A-COM's interests also include cable TV, satellite ground stations and, with the addition of Ohio Scientific, an office of the future with distant locations tied together by satellite. Other M/A-COM acquisitions have been more closely related to its original field of interest.

OSI has done some work in data communications; its C2 and C3 business systems are designed to allow networks of multiple processors and work stations, but these are limited to telephone lines and other forms of hard wire. Nevertheless, this was seen by M/A-COM as a step in the right direction. The OSI purchase is a step into a new market for M/A-COM.

As of the first of January, 1981, OSI has a new president named Harvey P. White. Mr. White comes to OSI from LINKABIT where he was Executive Vice President. OSI is currently building new facilities, including offices, and I expect that new management posts will be created and filled this year. In addition, OSI is looking forward to an influx of new ideas from the engineering departments of other M/A-COM companies, presumably along the lines of networks and communication. OSI has been testing systems in Ohio, Virginia and elsewhere, with a home computer and cable TV hookup for home comparison shopping, stock market reports, etc.

The purchase also provides OSI with new opportunities for growth by supplying a source of capital for research and development. OSI is now part of a publicly-held corporation whose annual sales total about $322 million.

I called OSI because I was concerned about the future of the home computers they have pioneered. I was assured that no changes in product line or marketing strategy would result from the take-over. OSI's founder, Mike Cheiky, is now Chairman of the Board, Director of Marketing and Director of Research and Development. It has been M/A-COM's practice in past acquisitions to leave top management in place at these companies, and OSI is no exception. In this case, the policy makes especially good sense since OSI is more consumer-oriented than any other M/A-COM division. OSI aims at three markets: small systems, educational, and business/-scientific, whereas M/A-COM sells to industry and government.

In sum, the M/A-COM connection does not mean bad news for any user or potential user of Ohio Scientific small systems. In fact, it is probably good for everyone, but especially for the high end or business market. I expect no dramatic improvement in low end support-like documentation or application notes as M/A-COM expertise does not lie in that direction. Low end application support does come from journals like MICRO and from local user groups.

More Connection

In another example of vertical integration, last November Ohio Scientific bought, for an undisclosed sum, the Hard Disk Division of Okidata. This means that OSI now owns the California plant which produces its CD-74 hard disk systems. The same plant will soon produce a CD-37 system, which has half the memory and sells for $8,500—a little more than half the price of the CD-74.

The main reason for the Okidata Hard Disk Division purchase was to ensure a supply of these systems. Also, most of the plant's output was going to OSI and most of OSI's disks were coming from that plant.

User Groups

One of the most active OSI user groups is in Maryland. It is known as OSIO and produces two separate monthly publications, PEEK(65) and the OSIO newsletter. PEEK(65) is the younger journal and has at least one staffer in common with the newsletter, OSIO president Wallace Kendall.

PEEK(65) ran sixteen pages last November, including six pages of ads. Article topics included: better random numbers, Superboard expansion, C1P display format modification, 65U file header explained, and password file protection. Many letters from users nationwide provided answers to those annoying little questions.

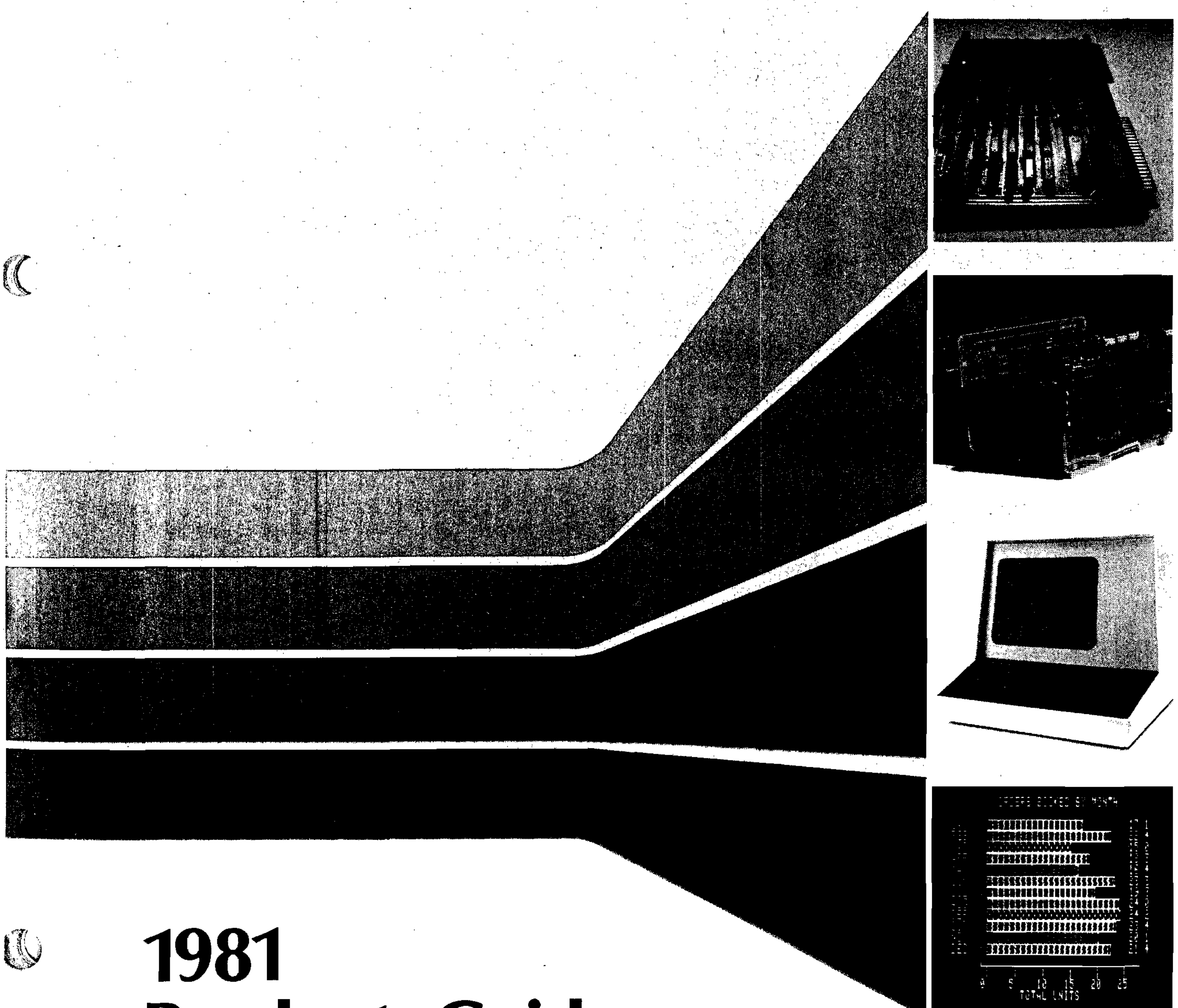
OSIO Newsletter carries no advertising and is oriented more toward software than PEEK(65). Topics include: telephone communications, real time clock, line printer driver, modem software, Hi-Res graphics, indirect ASCII files and so on. OS-65D notes are a regular feature. Subscriptions are $12 for a year (12 issues) of PEEK(65). Membership in OSIO, which includes a subscription to the Newsletter, is $15 a year. Last fall OSIO offered both subscriptions for $25, but this may no longer be available. Back issues are available and form a good, if unorganized, reference library.

PEEK(65), The Unofficial OSI User's Journal
1819 Bay Ridge Ave., Suite 220
Annapolis, Maryland 21403

OSIO
9002 Dunloggin Rd.
Ellicott City, Maryland 21043

I want to encourage OSI user groups to let us know about their activities, especially foreign groups. I urge all OSI groups interested to register with the MICRO Club Circuit, and to feel free to send newsletters, etc. to my attention.

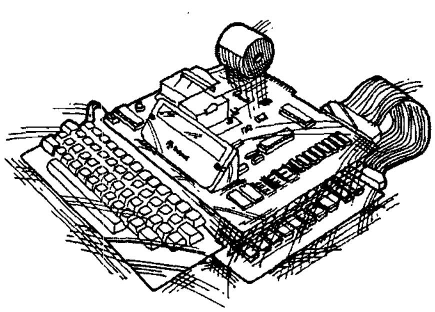
THE COMPUTERIST®



1981
Product Guide

The Capability Company...

The Computerist offers a complete line of integrated microcomputer products, from expansion boards to single-board microcomputers to complete systems. Our extremely versatile boards are well designed and documented in order to allow easy expansion and customization. They provide a convenient and reasonably priced solution to your microcomputer system needs.



The Computerist has been providing support for 6502-based systems since 1976. Our milestones include:

- First commercial software package for the KIM-1.
- First power supply custom-designed for the KIM-1.
- First multi-purpose expansion board for AIM, SYM and KIM with RAM, EPROM, EPROM Programmer and I/O on a single board.
- First AIM enclosure with built-in power supply.
- First high-quality, 6502-reference journal — MICRO — now published by Micro Ink, Inc., a sister corporation.

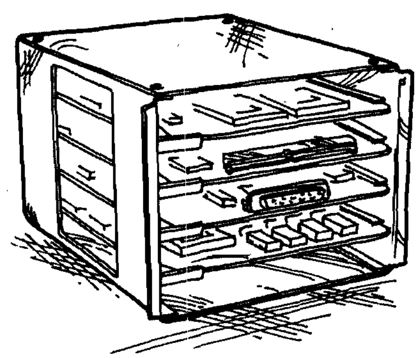
The Computerist has developed a complete line of integrated sub-systems designed around the 6502 microprocessor. These modules form the basis of our current product line. Each board is a second-generation product incorporating years of development and testing. Thousands of our boards are currently working in a wide range of environments and applications.

Each of **The Computerist**'s boards — Dram Plus, Video Plus and Proto Plus — may be used as an expansion board for ASK family (AIM, SYM and KIM) microcomputers. We have adapted our popular Dram Plus to run with the PET/CBM and Ohio Scientific Superboard as well.

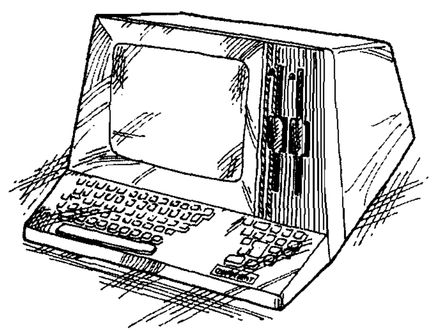
All of **The Computerist**'s boards include provision for customization and configuration in the field. Our Mother Plus allows multi-board systems to be neatly and economically packaged.

The Computerist supports customers by providing all the necessary software to run its boards; and each program runs on any ASK family micro.

With the addition to our product line of Micro Plus, a complete single-board microcomputer, **The Computerist**'s boards may now be used to create complete microcomputer systems tailored to user specifications.



The Computerist will bring out a number of additional products in 1981, including the Floppy Plus disk controller, additional software packages for Micro Plus, as well as complete systems. The complete systems will include a basic terminal package, a group of intelligent terminal emulators, a word processing system, an editing/assembly/teaching system and business systems. These will be built around the Micro Plus board with other modules from our standard product line.



The Computerist can build unique systems from the standard products listed in this guide plus a virtually unlimited selection of enclosures, monitors, keyboards, disk drives, and so forth. This versatility permits a user to obtain exactly the system he requires. A number of special packages are available for particular interests.

OEM/Systems Integration Houses — Customization support, extended warranties and direct end-user service plans.

Education — Systems with teaching-oriented software and student-oriented features.

Terminals — Terminal emulators and customized terminals.

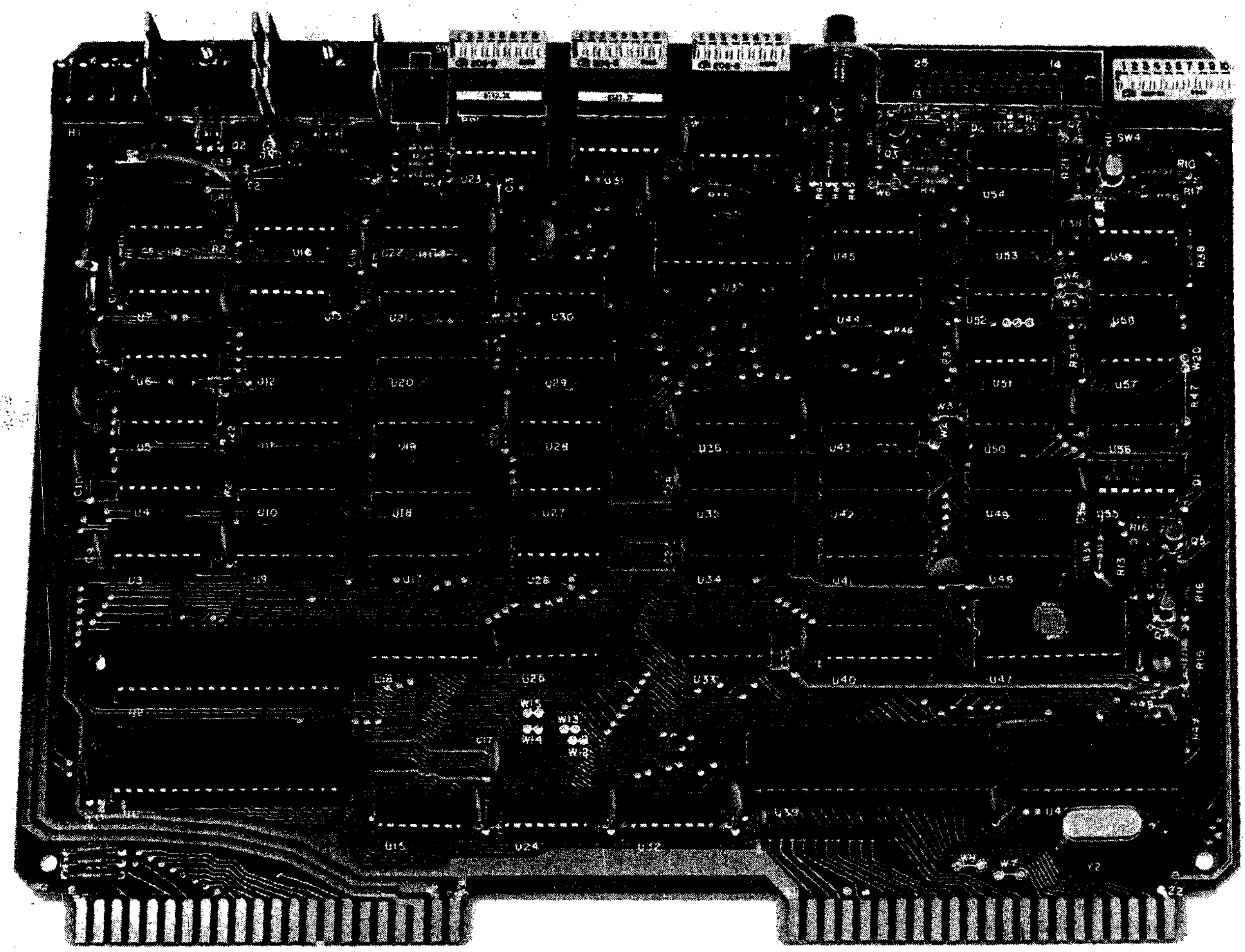
Turnkey Systems — Complete systems to service specialized applications.

If you have an interest or application requirement in one of these areas, please contact us for additional information.

THE COMPUTERIST®

34 Chelmsford St., Chelmsford, MA. 01824 617/256-3649

MICRO PLUS™



A Microcomputer with Sophisticated Video and Communications Capabilities

Complete Enough to be Used Alone in Some Applications
Flexible Enough to be Used as a Building Block for Custom Systems

The Micro Plus TCB-111 is a 6502-based microcomputer with built-in controllers for video, communication and keyboards. The microcomputer contains a 6502 microprocessor, RAM, EPROM and I/O capability. The video controller supports programmable screen format, user-defined character sets, cursor control and other functions. The communications controller supports RS232 and 20-mA current loop service, programmable baud rates, and other communication functions. The keyboard controller services most ASCII keyboards. A choice of software monitors in EPROM allows system customization.

features

- 6502-based microcomputer system: Up to 7K RAM and 4K EPROM; 6522 VIA for basic I/O.
- 6845-based video controller includes: Programmable screen format up to 132 columns by 30 lines; EPROM character generator with standard upper/lower case ASCII or any other user-defined 8-by-16 character set; Cursor control and editing functions; Reverse video, blanking, and special effects; RAM character generator (see Video Plus for details).
- 6551-based communications controller with: RS232 with modem controls; 20 mA current loop; Programmable baud rates to 19.2K; Parity generation/testing.
- 6522-based keyboard controller: Supports ASCII keyboards with pos/neg strobe and data.
- Several monitors to choose from: MicroMon 1: Video editing, simple communications and basic debugging. AIM, KIM, SYM expansion support.

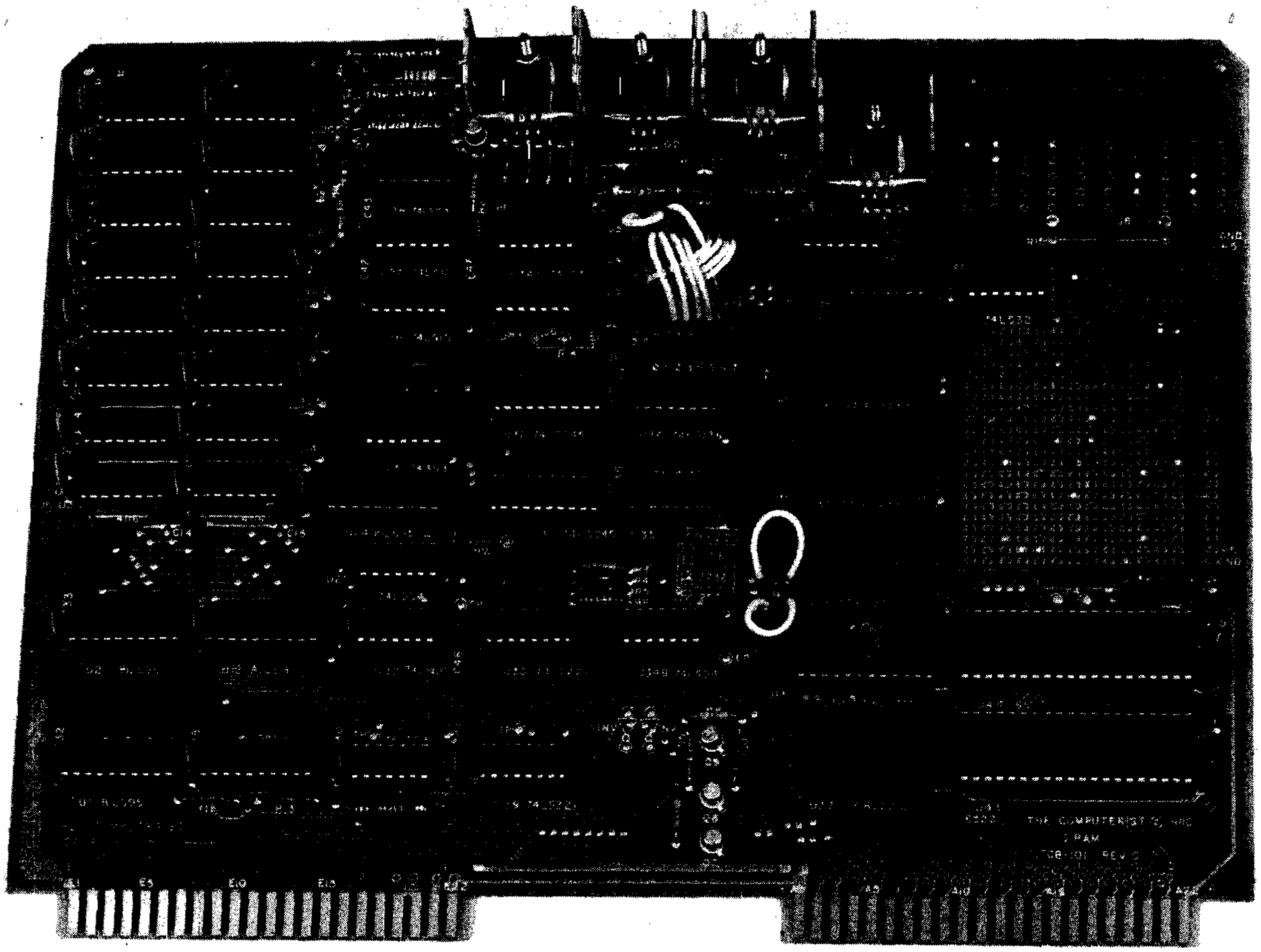
  MicroMon 2: Enhanced editing, modem communications and advanced debugging.

  MicroMon 3: Full source editor and assembler plus all MicroMon 2 features.
- Easy to Expand: Dram Plus adds 16/32K RAM, 16K EPROM, I/O; Floppy Plus adds 5¼-and 8-inch disks.

specifications

Micro Plus hardware is identical to Video Plus with the addition of the 6502 microprocessor (see Video Plus).

DRAM PLUS™



The Dram Plus TCB-101 system expansion board includes 16K or 32K dynamic RAM, 16K EPROM or ROM and EPROM programmer on a single board. 2K EPROMs may be addressed on 2K boundaries, allowing efficient utilization of available address space. Two VIA devices provide four 8-bit parallel I/O ports or 40 independently programmable I/O lines. Prototyping area allows additional address decoding or memory management circuitry to be added. Dram Plus interfaces to ASK computers through simple expansion cable or Mother Plus board.

features

- 16K or 32K dynamic RAM addressable in 4K segments
- Transparent refresh
- Up to 16K ROM or EPROM
- EPROM programmer
- 2716, 2532 and 2732 EPROMs and 2332 ROM
- Two versatile interface adapters
- Prototyping area for custom circuits
- On-board voltage regulators
- Flexible address selection
- Adaptable to PET (1st Mate) and OSI Superboard (Super Mate)
- All IC's are socketted

specifications

Power Requirements

+5 volts @ 1 amp
+12 volts @ 150 milliamps
+27 volts @ 50 milliamps for EPROM programming

All voltages may be regulated on board from higher supply voltage

Read/Write Access Time: 375 nS

Bus Signals

Address Bus: Inputs are fully buffered presenting one TTL/LS load
Data Bus: Three-state, TTL-compatible, buffered inputs/outputs
Control Signals: All signals and clocks are buffered to present one TTL/LS load

Operating Temperatures

0° to 70° C

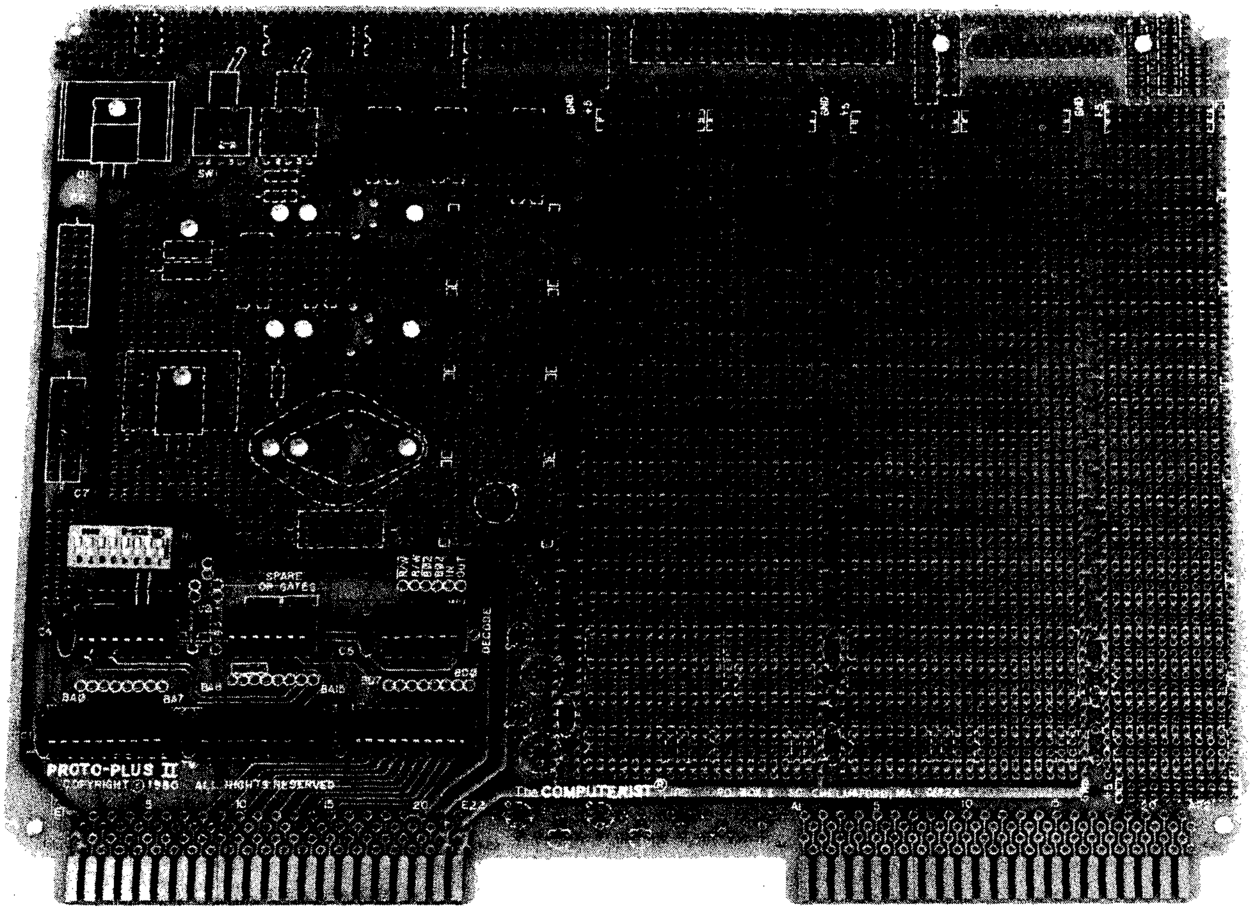
Physical Characteristics

Length: 7.875 in.
Width: 10.75 in.
Board thickness: .0625 in.

Connectors

Edge Connectors:
Dual 22/44 .156 centers
(Winchester HCA 2250 or equivalent)

PROTO PLUS™



The Proto Plus TCB-115 simplifies custom system expansion and prototype construction. It includes etched patterns for address, data, and control-line buffers, as well as address decoding circuitry required by most system expansion boards. Proto Plus is available as a bare board or assembled and tested.

features

- Room for over 60 16-pin DIPs
- Universal grid pattern
- Power and ground bus with locations for decoupling capacitors
- Buffer and decoder circuit provided
- Special patterns for many discrete components
- Pattern supports wire-wrap and solder connections
- Double-sided etch
- Silk-screened
- Gold-plated fingers
- Plated-through holes

specifications

Length: 7.875 in.
Width: 10.75 in.
Board thickness: .0625 in.

Edge connectors:
Dual 22/44, .156 centers
(Winchester HCA 2250 or equivalent);
Identical to AIM, SYM, or KIM edge connector.

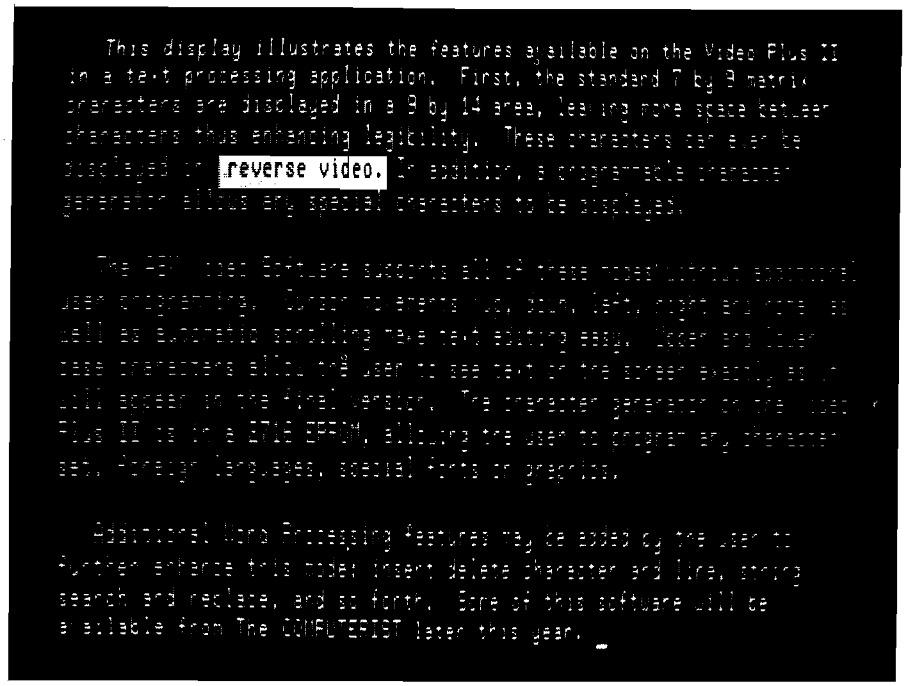
VIDEO PLUS™

The Video Plus TCB-112 provides intelligent video, keyboard and communication capabilities for 6502-based systems. Its video features may be customized for specific applications and include programmable character sets, programmable screen formats, reverse video, and more. It interfaces to most ASCII keyboards.

The Video Plus may be used as direct expansion for the ASK microcomputer family. It may be interfaced directly via a cable and software is provided which will instantly support the monitors, editors, assemblers and BASICs of these systems.

The Video Plus includes many configuration options which allow an OEM or end user to easily adapt the board to his specific requirements. The display format is determined by a hardware controller driven by software parameters and these parameters may be changed at any time. Some operating modes are switch selectable including low-bandwidth (TV) output, reverse video, PCG characters, keyboard data and strobe polarities. Additional unassigned control switches, which may be read by the software, allow further customization and configuration control.

The communications option provides full RS232 support with programmable baud rate, parity generation and checking, and more.



Word Processing — 80 Columns by 20 Lines
[7 × 9 Character Set on 9 × 14 Display Grid]

features

Video Features

- MCM6845 programmable CRT controller
- Programmable screen format up to 132 characters by 30 lines
- Complete cursor control
- Television and CRT monitor modes
- Composite or separated video outputs
- Reverse video on character-by-character basis
- EPROM character set for user-definable characters
- RAM character set for dynamically changing characters under program control
- Light pen input
- Programmable character width
- Up to 4K display memory

Communication Features

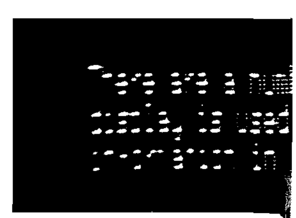
- SY6551 ACIA asynchronous communications interface adapter
- Programmable baud rates from 50 to 19.2K baud
- Parity generation and checking
- Programmable word length and stop bits
- Data set and modem control signals provided at TTL levels
- 5-, 6-, 7-, 8- and 9-bit transmission
- Full-duplex or half-duplex operation
- Both RS232C and 20-milliamp current loop interfaces provided
- Standard RS232C-D connector

Keyboard Interface

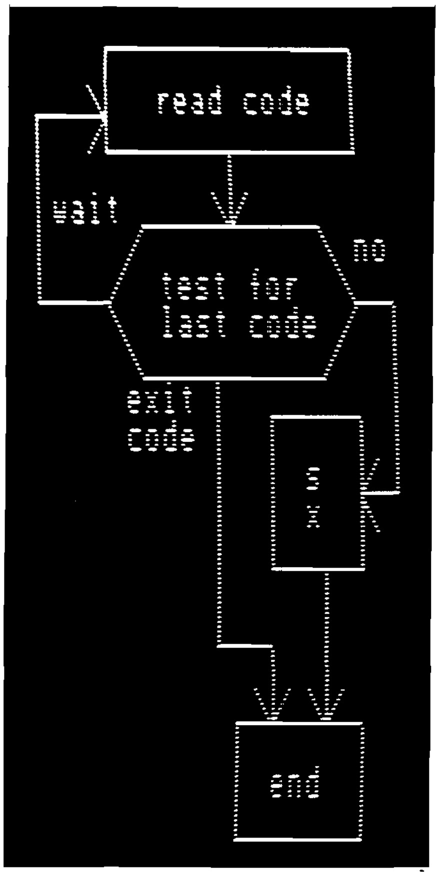
- Direct interface for most ASCII keyboards
- Switch selectable strobe polarity
- Switch selectable data polarity
- 8-bit input including parity

MicroMon 1 Software Features

- Resident in 2K EPROM
- Fully supports AIM and SYM software: monitors, editors, assemblers and BASICs
- Supports AIM keyboard in upper and lower case in BASIC
- Page zero use transparent to host microcomputer programs
- Automatically relocates in memory
- Automatically determines type of host microcomputer
- Supports basic ACIA communications
- Screen editing features: cursor control, screen erase, upper case mode, scrolling, delete, etc.



Dense Mode — 132 Columns by 30 Lines
[5 × 7 Character Set on 6 × 9 Display Grid]



specifications

Power Requirements

+5 VDC @ 1.2 amps (maximum). On- board regulators permit use of unregulated +8 to +12 VDC in place of regulated +5 VDC.

Bus Signals

Address bus: Inputs are fully buffered presenting one TTL/LS load
Data bus: Three-state, TTL-compatible, buffered inputs/outputs
Control signals: All signals and clocks are buffered to present one TTL/LS load

Simple Line Graphics [8 × 8 Character Set on 8 × 8 Display Grid]

Operating Temperatures

0° to 70° C

Physical Characteristics

Length: 7.875 in.
Width: 10.75 in.
Board thickness: .0625 in.

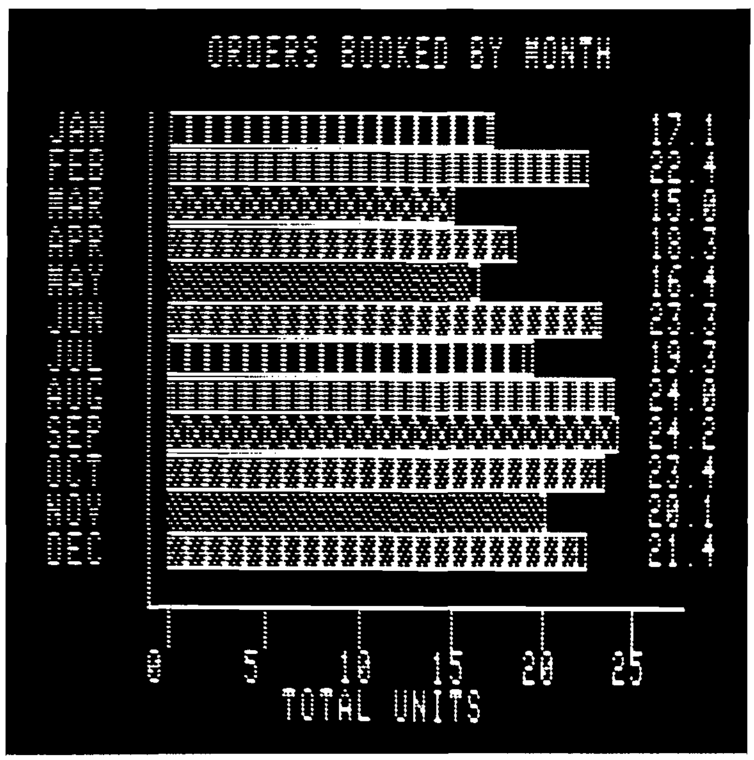
Connectors

Edge connectors:
Dual 22/44, .156 centers
(Winchester HCA 2250 or equivalent);
Identical to AIM, SYM or KIM connector

RS232C-D with communications option
(Cinch Jones DB25s or equivalent)

12 posts with .10 spacing
(Waldom 22-01-2121 or equivalent) for keyboard

Graphic Characters in a Business Application [8 × 8 Characters on 8 × 8 Display Grid]

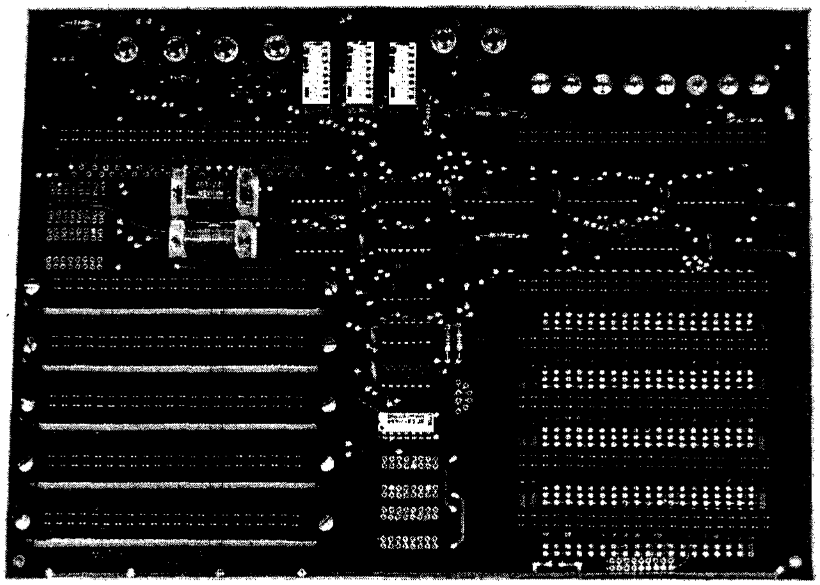


System Features

- Up to 7K RAM: 4K display RAM, 2K programmable character generator RAM (which may be used for program RAM), 1K program RAM
- Up to 4K EPROM: Normally used with 2K 2716; may be easily upgraded to use 4K 2532 EPROM
- Provision for addition of 6502 for complete stand-alone system (See Micro Plus)
- DIP switches for selecting major options
- Jumper blocks for selecting minor options
- All IC's socketted
- Gold-plated edge connectors

of applications which can make good use of this very dense display mode. Examples are any system in which the format data to be output to a wide, normally 132 character, line printer. The capability of having the same same format on the screen as on the line printer can greatly simplify the entire job.

MOTHER PLUS™



The Mother Plus TCB-113 provides practical support for AIM, SYM and KIM system expansion. It includes buffered address, data and control lines, configuration switches, power and I/O connections. In conjunction with Card Cage Plus, it allows peripheral boards to be added to form a complete system in a small package.

features

- Room for five expansion boards
- Works with AIM, SYM and KIM
- Fully buffered address, data and control lines
- Switches select expansion board addresses
- Convenient terminal strip power connections
- Phono jacks for TTY and cassette connections
- Cassette control relays
- Cassette input monitor LED
- Application connector solder eyelets
- Standard KIM-4 bus expansion connections
- Compact vertical orientation
- Generates decode signal for KIM
- Host ports A and B brought to DIP socket

specifications

Power: +5 volts at 100 milliamps

Dimensions:
Length: 8 in.
Width: 11 in.
Board thickness: .0625 in.

Microcomputer connections follow KIM-1 standard

Expansion connections follow KIM-4 standard

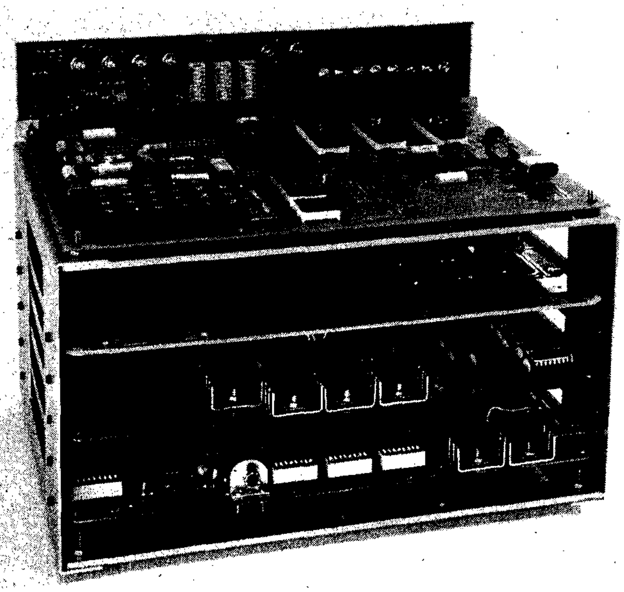
Spacing between host and first expansion board: 2 in.

Spacing between expansion boards: .875 in.

CARD CAGE PLUS™

The Card Cage Plus TCE-303 is designed to support an ASK microcomputer and from one to five expansion boards in a compact package.

Simple assembly consists of bolting to Mother Plus, bolting on two cross members, snapping the card guides into place, and adding stick-on insulation and feet.



specifications

Length: 8.25 in.

Width: 11 in.

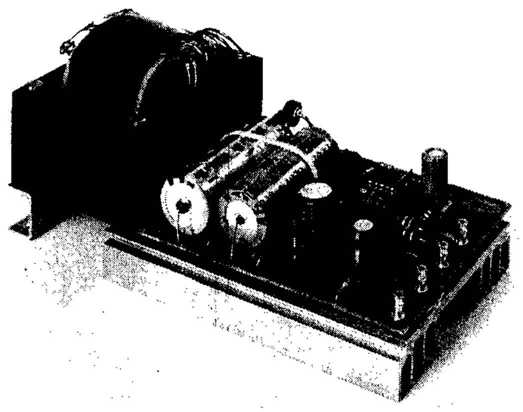
Height: 6 in.

Weight: 1 lb.

Composition: Gold anodized aluminum

POWER PLUS™

AIM PLUS™



Triple Voltage Microcomputer Power Supply

An Enclosure with Built-in Power Supply

Input:
110 or 220 VAC @ 50 or 60 Hz

Output:
+5 VDC @ 5.0 amps with protection
+12 VDC @ 0.5 amps
+24 VDC @ 1.0 amp/1.5 amp surge
−5 VDC @ 0.5 amps (optional)

Compact:
Length: 7 in.
Width: 3.25 in.
Height: 3 in.
Weight: 3 lb. 10 oz.

Complete: Includes a fuse holder, ON/OFF switch and a heavy-duty, three-wire grounded power cord.

features

- **Functional packaging:** This unit protects the AIM while providing access to all switches, the LED display, keyboard and printer.
- **Easy assembly:** All fasteners are provided; absolutely no alteration of the AIM is required, and the non-conductive, thermoformed plastic can be cut with a knife if any special alterations are desired. The edge connectors are accessible from the rear.
- **System expansion:** The enclosure has room in the bottom for one expansion board, such as a Video Plus or Dram Plus, with direct access to the edge connectors.
- **Power supply:** The Power Plus supply, built into the enclosure, provides +5 VDC and +24 VDC to run the AIM, and enough power to spare for an expansion board. The line cord and fuse holder are positioned at the rear of the case. The ON/OFF switch is mounted to the right of the LED display on the front of the case. Louvers provide air flow for cooling the power supply which is bolted to the top of the case. For specifications, see Power Plus specifications below.

specifications

Dimensions: Length 16.25 in.; width 14 in.; height 6.25 in.
Weight: 6 pounds, including power supply
Color: grey and black

Power Plus Input Specifications

105-125 VAC, 50 - 60 Hz, 1.5 Amperes or 210-250 VAC, 50 - 60 Hz, 0.8 Amperes

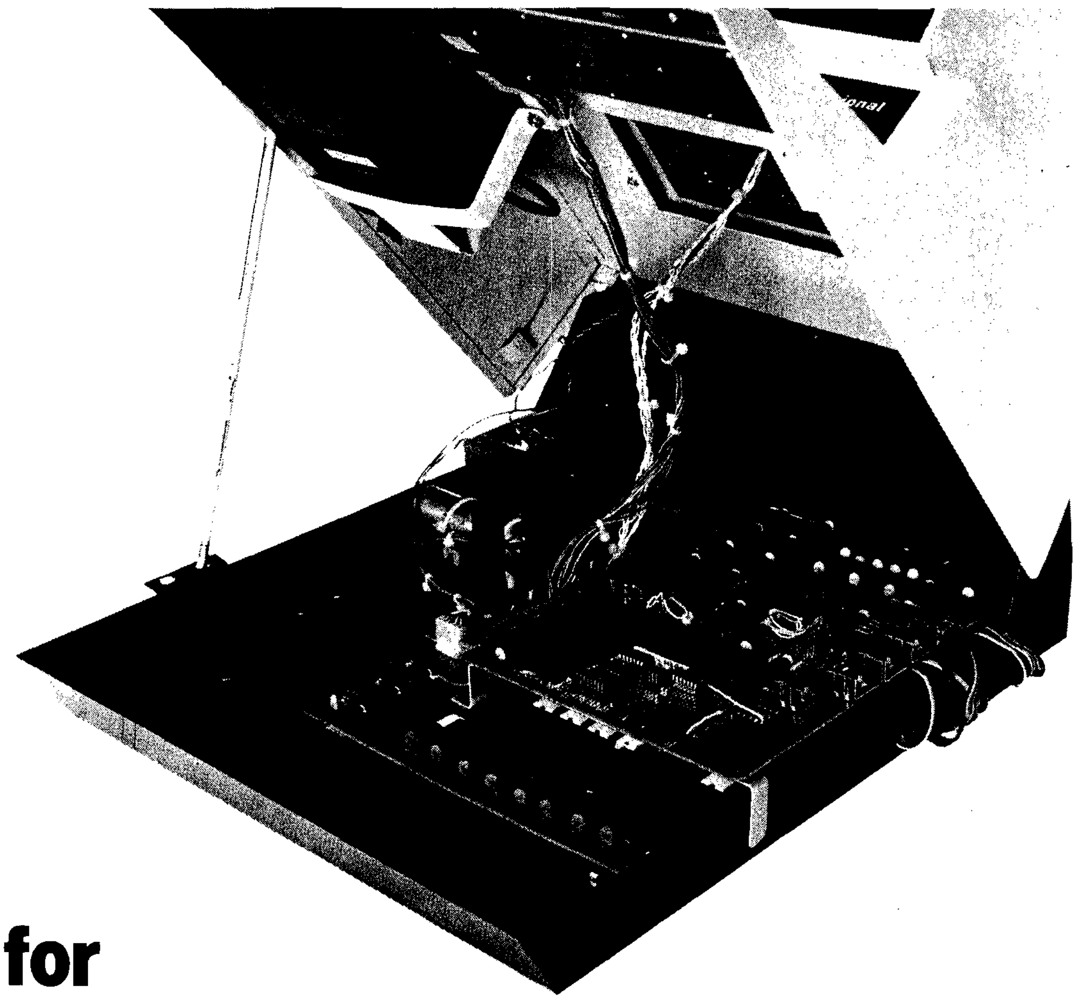
Power Plus Output Specifications

Output	Ripple and Noise	Regulation
+5 VDC @ 5 Amps Overvoltage protection Current foldback short circuit protection Reverse voltage protection	10 mv max. @ 0.0 to 4.5 Amps 25 mv max. @ 4.5 to 5 Amps	(Voltage fixed to within 1%) ±0.1% line and load with ± 10% line change or 50% load change in 0.0 to 4.5 Amp load range
+12 VDC @ 0.5 Amp (−5 VDC @ 0.5 Amps Optional) Thermal shut down overload protection	15 mv max. @ 0.0 to 0.5 Amp	(Voltage fixed to within 5%) ± 1%, line and load with ±10% line change or 50% load change in 0.0 to 0.5 Amp load range
+24 VDC @ 1.0 Amps* (1.5 Amp surge) Thermal shutdown overload protection Reverse voltage protection.	24 mv max. @ 0.0 to 1.0 Amp	±2% line or load reg. 0 to 1 Amp

*If 12 volt output is 0.5 Amp. continuous, derate 24 volt output to 0.5 Amps

Thermal Specifications (Derate linearly) 100% output @ 80° C (176°F) heat sink temp./50% output @ 90° C (203°F) heat sink temp.

1ST MATE™



for PET/CBM

1st Mate , TCB-110, is a memory and I/O expansion board for all PET and CBM computers. It includes 16K or 32K dynamic RAM and up to 16K ROM or EPROM. RAM is independently addressable in 4K segments. A programmable address controller allows different RAM and/or ROM resources to be switched to the same address space. Two 6522 VIAs provide forty programmable I/O lines, shift registers and timers. EPROM programmer includes BASIC software.

features

- 16K or 32K dynamic RAM
- Up to 16K EPROM/ROM
- Complete EPROM programmer
- Programmable address controller
- Two versatile interface adapters
- Two-button reset support
- Mounts inside PET/CBM
- Includes connecting cable
- May draw power from PET supply

specifications

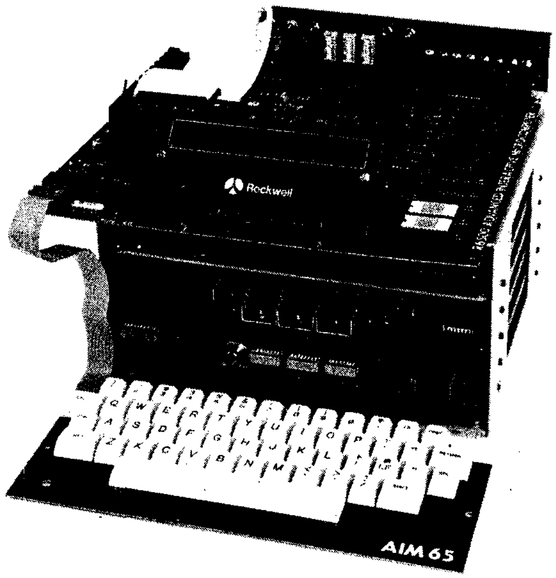
Length: 7.875 in.
Width: 10.75 in.
Board thickness: .0625 in.

Edge connectors:
Dual 22/44, .156 centers
(Winchester HCA 2250 or equivalent);
Identical to AIM, SYM, or KIM edge connector.

(For complete specifications, see Dram Plus)

AIM 65 by Rockwell International

AIM 65
TCS-403



AIM 65 in Card Cage with Mother Plus and several expansion boards.

- The full-size, typewriter-style keyboard makes it easy to enter data, programs, edit files, and so forth
- A twenty-character LED display can display all normal alphanumeric characters in an easy-to-read format
- A twenty-column thermal printer provides hardcopy
- The 8K ROM monitor includes a mini assembler, disassembler, and editor, plus other important support functions
- Expandable on-board to 20K ROM and 4K RAM
- BASIC and an assembler/editor are available in ROM to plug directly into the basic system
- Expandable with Dram Plus, Video Plus, Proto Plus, Mother Plus
- Works with Power Plus and AIM Plus

Floppy Plus™

FLOPPY PLUS
TCB-108

The Floppy Plus TCB-108 is a controller for 8-inch and 5¼-inch floppy disk drives. It will handle one to four Shugart-compatible drives and IBM and other formats. Floppy Plus will include on-board support software in EPROM, on-board RAM, 6522 VIAs and a cassette controller. It may be connected directly to Micro Plus or to ASK family computers. With the addition of a 6502 processor, it will run as a stand-alone disk controller/formatter and interface to any computer through a standard I/O port.

features

- Based on WD 1791B-01 Floppy Disk Controller Chip:
  - Handles variety of formats including IBM
  - Handles both 8-inch and 5¼-inch drives
  - Controls up to four drives
- On-board RAM
- On-board support software:
  - disk formatter
  - read/write/allocate routines
  - file management system
- Additional I/O devices including 6522 VIAs and cassette interface

Available Second Quarter 1981

Expansion Cable

The Computerist's Expansion Cable, TCX-922, may be used to directly connect an AIM, SYM or KIM to a Dram Plus, Video Plus or Proto Plus.

Warranty

The Computerist provides a one-year limited warranty on all of its products. AIM 65 is covered by Rockwell's 90-day warranty.

Literature

The Computerist provides extensive documentation for its major products. Documentation includes schematics, component layouts, parts lists, installation and operating instructions, program listings and all of the information required to effectively evaluate and use the boards. Documentation may be purchased separately in order to assist the evaluation process.

Documentation Packages:

TCB-101-D	Dram Plus	$10
TCB-108-D	Floppy Plus*	$10
TCB-109-D	SuperMate	$10
TCB-110-D	1st Mate	$10
TCB-111-D	Micro Plus	$10
TCB-112-D	Video Plus	$10
TCB-113-D	Mother Plus	$ 5
TCB-115-D	Proto Plus	$ 5

**(2nd Quarter)*

Price includes shipping in US only.
Foreign: add $2.00 surface postage.
Air mail — write for rates!

Ordering Information and Price List

Part Number	Description	List Price (US Dollars)	Shipping Weight Lbs.
	System Boards		
TCB-101-16	**Dram Plus** with 16K RAM for AIM/SYM/KIM	$325.00	3
TCB-101-32	**Dram Plus** with 32K RAM for AIM/SYM/KIM	395.00	3
TCB-108	**Floppy Plus** (Available second quarter)	*	3
TCB-109-16	**SuperMate** with 16K RAM for OSI superboard	375.00	3
TCB-109-32	**SuperMate** with 32K RAM for OSI Superboard	445.00	3
TCB-110-16	**1st Mate** with 16K RAM for PET/CBM	395.00	3
TCB-110-32	**1st Mate** with 32K RAM for PET/CBM	475.00	3
TCB-111	**Micro Plus** with 3K RAM	375.00	3
TCB-112	**Video Plus** with 3K RAM	325.00	3
TCB-113	**Mother Plus**	125.00	2
TCB-115-A	**Proto Plus** assembled and tested	75.00	1
TCB-115-B	**Proto Plus** bare board	50.00	1
	Options		
TCX-910	**4K Memory Option** for Micro/Video Plus	50.00	1
TCX-911	**Communications Option** for Micro/Video Plus	50.00	1
TCX-922	**Expansion Cable** for AIM/SYM/KIM	20.00	1
	Power Supplies and Enclosures		
TCP-212	**Power Plus** Triple Voltage Supply	85.00	4
TCP-213	**AIM Plus** Power Supply and Enclosure	150.00	8
TCE-302	**AIM Enclosure**	50.00	3
TCE-303	**Card Cage Plus** for use with Mother Plus	25.00	2
	Software EPROMs		
TCM-601	**MicroMon 1** basic Micro Plus	35.00	½
TCM-602	**MicroMon 2** advanced Micro Plus	35.00	½
TCM-603	**MicroMon 3** editor/assembler	*	½
TCM-620	**ASK Video Software** for old Video Plus	35.00	½
TCM-621	**ASK Video Software** for new Video Plus	35.00	½
TCM-622	**Character Generator** for Micro/Video Plus	35.00	½
	Documentation (included with purchase of boards)		
TCB-101-D	**Dram Plus**	10.00	½
TCB-108-D	**Floppy Plus***	10.00	½
TCB-109-D	**SuperMate,** Dram for OSI Superboard	10.00	½
TCB-110-D	**1st Mate,** Dram for PET/CBM	10.00	½
TCB-111-D	**Micro Plus**	10.00	½
TCB-112-D	**Video Plus**	10.00	½
TCB-113-D	**Mother Plus**	5.00	½
TCB-115-D	**Proto Plus**	5.00	½
	AIM 65 by Rockwell		
TCS-403-1	**AIM 65** with 1K RAM	425.00	5
TCS-403-4	**AIM 65** with 4K RAM	475.00	5

*Note: Floppy Plus and MicroMon 3 will be available second quarter 1981.

The prices in this catalog are effective 1 March 1981 and supersede any and all other prices and are subject to change without notice. The price listed is for US/Canada only. All other countries must add 10% to the total price to cover costs incurred in processing overseas orders.

All products, with the exception of the AIM 65, carry a limited one-year warranty.

Quantity discounts available on all products except the AIM 65.

Dealer inquiries invited.

Shipping Table

Country	First Pound	Additional Pounds
United States	$ 2.00	$ .40
Canada	$ 4.00	$1.00
Europe & South America	$10.00	$4.00
Other Countries	$10.00	$6.00

Any overpayment in excess of $5.00 will be refunded.

For documentation packages, prices include shipping in the U.S. For foreign documentation shipments, add $2.00 surface postage. Air mail — write for rates!

Order these fine products directly from:

THE COMPUTERIST®
34 Chelmsford Street
Chelmsford, MA 01824
617/256-3649

or from your local dealer.

MICRO

PET Vet

By Loren Wright

Commodore Shows

Commodore's idea for public shows seems to be a great success. As I write this in early February, there has already been one in Philadelphia, December 13-14, 1980. By the time this issue reaches you there will have been another in Boston, February 7-8, 1981. A third is planned for New York, but a definite date has not yet been set.

While Commodore obviously sponsors these shows to attract new customers, particularly business customers, the shows can be very useful to Commodore's old customers. New products, such as the VIC 20, OZZ, and Wordcraft 80 are on display.

Many Commodore-supporting companies, particularly those from the show's local area, have displays. In Philadelphia, there were hourly drawings for Commodore watches, calculators, thermostats, and even a PET. A magician entertained the children, while they weren't playing computer games. Philadelphia Phillies' shortstop Larry Bowa made an appearance, signing autographs and participating in one of the drawings. I was particularly impressed with the great number of computers Commodore provided for its co-exhibitors and for public use.

PET and the GPIB

The PET continues to be one of the least expensive IEEE-488 bus controllers available, if not *the* least expensive. Most PET owners are unaware of its full capability. Devices compatible with the bus (which is also called GPIB and HPIB) continue to increase in number and decrease in cost. Hewlett-Packard, Tektronix, and Fluke, are some of the better-known companies that manufacture GPIB devices. These include meters, frequency counters, timers, plotters, spectrum analyzers, and many others.

Communication with GPIB devices is as easy as using the Commodore cassettes, printer, and disk units. The BASIC commands are essentially the same. If the PET's machine language subroutines are used properly, bus transactions can be faster and more efficient.

Commodore BASIC Version 4.0: User's Reference Manual, published by Commodore, is a worthwhile addition to your library, if only for its "Appendix H", covering the various GPIB machine language routines and how to use them. *PET and the IEEE-488 Bus (GPIB)*, mentioned in an earlier column, is probably a better starting point, though.

Readers who have had experience using the PET with GPIB devices are encouraged to write about their particular applications and experiences. Shorter contributions will be combined in a future PET Vet column, while longer ones will be considered for publication as articles.

Other topics I'm considering for future columns are:

1. Memory expansion—what is the potential and what is commercially available?
2. Character set substitution—yes, it is possible, there are commercial substitutes available, and you can design and make your own!
3. Fixing an Old PET—they're great machines, once you fix a few things, and more and more are becoming available used at attractive prices.

I would like reader comments on these topics and suggestions for others. Please address correspondence to:

Loren Wright
MICRO, The 6502 Journal
P.O. Box 6502
Chelmsford, MA 01824

8032 Screen Functions

At least three articles have been published describing the various features of the CBM 8032: "Butterfield Reports: The 8032" by Jim Butterfield, *Compute* #5, July/August, 1980; "New Additions to the Commodore Line" by Robert W. Baker, *Kilobaud Microcomputing*, July, 1980; and "Programmer's Notes for the CBM 8032" by Roy Busdiecker, *Compute* #7, November/December, 1980. I have been using an 8032 for the last three months (thanks to Commodore) and have found the new screen features very useful. However, I find it difficult to remember which characters go with which functions.

There are three ways to implement the 8032 screen functions:

1. Print the appropriate ASCII characters using PRINT CHR$(xx).
2. Include the appropriate reverse field character in a string by preceding it with the ESC and RVS keys.
3. Some of the functions can be accessed directly from the keyboard by pressing the right three or four keys simultaneously.

Function	ASCII	Reverse Field Character	Keyboard Combination
BELL	7	g	
DELETE LINE	21	u	ESC, RVS, K
ERASE to BEGINNING of line	150	V	LS, ←, 3
ERASE to END of line	22	v	←, Q, 4
GRAPHICS screen	142	N	LS, ←, 3
INSERT line	149	U	SH, ESC, RVS, K
SCROLL DOWN	153	Y	LS, ESC, K
SCROLL UP	25	y	
SET BOTTOM	143	O	SH, Z, A, L
SET TOP	15	o	Z, A, L
SET TAB/CLEAR TAB	137	I	
TAB	9	i	
TEXT screen	14	n	

SH = either shift
LS = left shift
All digits are on the main keyboard, not the numeric keypad.

MICRO Microbes and Updates

Mike Rowe
Microbes & Updates
P.O. Box 6502
Chelmsford, MA 01824

Robert Babcock of McKeesport, Pennsylvania tells us: In the "Microbes and Updates" section of the September 1980 issue of MICRO, page 48, Mr. Wendall Malpass suggested variations to two of the programs, "Clear" and "Mover" which were contained in my article "Sharpen Your Aim" (19:37). The suggested variations can cause unwanted additions to the prompting messages in the operation of the programs.

The prompt generation routines from "Mover" shown below are taken from the AIM monitor and depend upon the most significant bit of the last character in a message having a 1 value to terminate the message.

```
02B8 B9 LDA 02C6,Y
02BB 48 PHA
02BC 29 AND #7F
02BE 20 JSR E97A
02C1 C8 INY
02C2 68 PLA
02C3 10 BPL 02B8
02C5 60 RTS
(M)=02C6 4F 4C 44 A0
(  ) 02CA 4E 45 D7 00
```

In the sequence starting at 02CA (4E 45 D7) (N E W), the D7 signifies both the letter W and also the fact that it is the last character in the string.

Operation of the routine to generate the word "NEW" would be as follows. First, the value of the Y register would be set at 04 prior to calling the subroutine starting at 02B8. The first entry (4E) is obtained and placed on the stack. The MSB is stripped by the AND 7F, the character is displayed and Y incremented. The original entry is then pulled from the stack and tested by the BPL 02B8 step. Since the MSB is 0, then the program returns to load the next value 45. The process is repeated and returns to obtain the third entry D7. Stripping the MSB yields 57 to generate the correct display character, but this time when pulled from the stack and tested the 1 in the MSB causes a drop out of the loop and consequently a termination of the message.

Changing the D7 to a 57 as suggested leaves the end of the message at the mercy of following entries until an entry containing a 1 in the MSB is found.

This is a compact method of generating messages since no additional locations are needed to indicate end of message and no counting of steps for that purpose is required.

From Baldwin L. Troutman of Bedford, Massachusetts: Refer to Mr. Charles W. Hall's letter in the December 1980 issue of MICRO; he is quite correct in saying that Applesoft will accept commas in strings if they are inclosed in quotation marks. However, if you write these strings into a disk file, when you try to read them back using the input statement, you get the familiar "extra ignored" message, and Applesoft does just that, and doesn't input anything after the comma into memory from the disk. So if one is inputting strings into a disk file, the quotation marks won't work with commas.

Phil Burcher of Alexandria Virginia sent this update to his article: Your sharp-eyed readers have noted some minor omissions in my article "Biorhythm, An AIM BASIC Programming Exercise", in MICRO Issue No. 29, October 1980.

In the short program to check the days between dates calculation on page 29:52, line 585 was omitted. Here's the way it should have been.

```
LIST585-610
 585 PRINT!"THERE AR
E";53
 590 PRINT!"DAYS BET
WEEN",M1;"/";D1"/";
Y1;"AND"
 600 PRINT!M2;"/ 1/"
;Y2
 610 GOTO10
```

In lines 1252 and 1251, the program checks for zero crossings, "critical days". It does this by checking for dots (periods). Unfortunately, the dots were lost in reproduction. Using the program as printed makes every day a critical day printing "C" rather than the date. Lines 1251 and 1252 should read:

```
LIST1251-1252
 1251 IFI$=".."THENI
$=STR$(J):GOTO1290
 1252 IFRIGHT$(I$,1)
="."THENI$="C":LC=-1
:GOTO1260
```

Louis K. Bell of Augusta, Georgia sent this update: R.M. Mottola's program in your August issue (27:53) was too good to modify—but I did!

My modifications (mostly) are lines 182-199 (which I added). I "discovered" CALL-418, which disassembles 20 instructions in Applesoft.

You may wish to share with your readers my modification and my use of CALL-418.

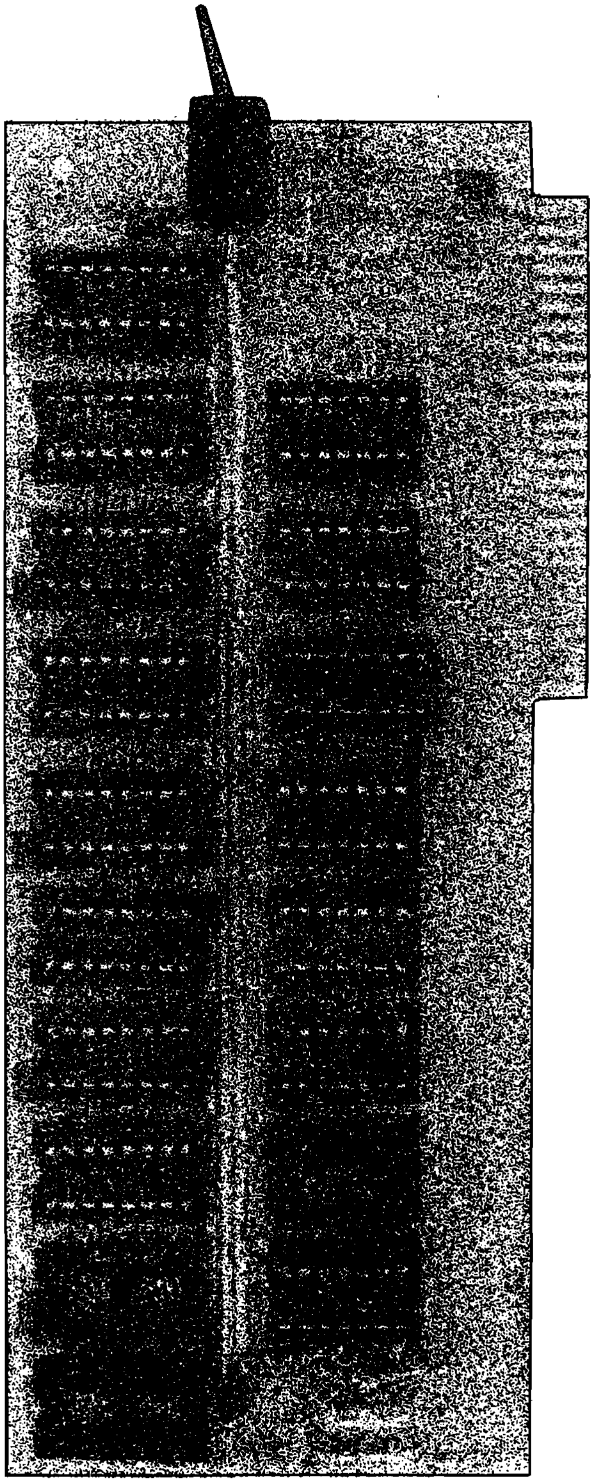
```
]LIST 182,199

182 POKE 58,0: REM LO
183 POKE 59,3: REM HI
184 HOME
185 CALL -418: REM
    DISASSEMBLER
186 VTAB 14: CALL -958: REM
    CLEAR LINES AFTER RTS
187 HTAB 10: VTAB 16: IN-
    VERSE : PRINT "CONVER-
    SION ROUTINE": NORMAL

189 HTAB 1: VTAB 23: GET R$
190 POKE 58,249: REM LO
191 POKE 59,234: REM HI
192 HOME : CALL -418
193 PRINT : HTAB 12: INVERSE
    : PRINT "MONITOR
    ROUTINE": NORMAL :
    HTAB 1: GET R$
199 GOSUB 600

]LIST 600

600 HOME : PRINT : PRINT
    TAB( 7
    )"FLOATING POINT CON-
    VERSIONS
    ": RETURN
```



MICRO

Microprocessors in Medicine: The 6502

By Jerry W. Froelich, M.D.

Information Transfer—Introduction

One of the principle reasons for the length, and thus the cost, of hospitalization is the time it takes to diagnose. The vast quantity of laboratory and radiological data acquired must be rapidly relayed to the managing physician before the data can be utilized to make the diagnosis. This column and the one to follow will describe how microprocessors rapidly distribute radiological data.

Rapid Telephone Access System

Rapid Telephone Access System (RTAS) was developed by Sudbury Systems Incorporated of Sudbury, Massachusetts. The system resulted from studies of the critical delays which occurred in obtaining radiological reports. RTAS provides the physician with rapid access to a dictated radiological report from any location via the public telephone network.

The report is dictated in the usual manner, permitting the radiologist to concentrate on films with minimal mechanical distractions (figure 1). Each report is stored on its own recorpak, a modular tape unit (figure 2 and figure 3, *R*), and is available for the referring physician to hear immediately after dictation. The physician simply dials the system and hears the actual dictation instantly; thus, the delay between dictation and transcription/distribution is minimized. A typist can also access the record to produce a permanent copy of the report for the patient's chart.

Description of RTAS

Hardware—Two versions of RTAS exist. The initial system is based on the 6502 microprocessor. The newest version, being installed now, will be discussed at the end of this article.

Like any information system, RTAS coordinates three basic operations: input, output, and storage.

Dictation—The radiologist dictates the report in the usual manner using a dictation handset with direct hardwire connections to the computer. The radiologist's dictation unit can record, play back, stop, and review, as in a standard dictation machine. A keyboard is used to enter the patient's identification code and to specify the mode of operation (e.g., dictation or recall).

Storage—Reports are stored in a central memory consisting of recorpaks: separate, independently controlled, miniature tape recorders which

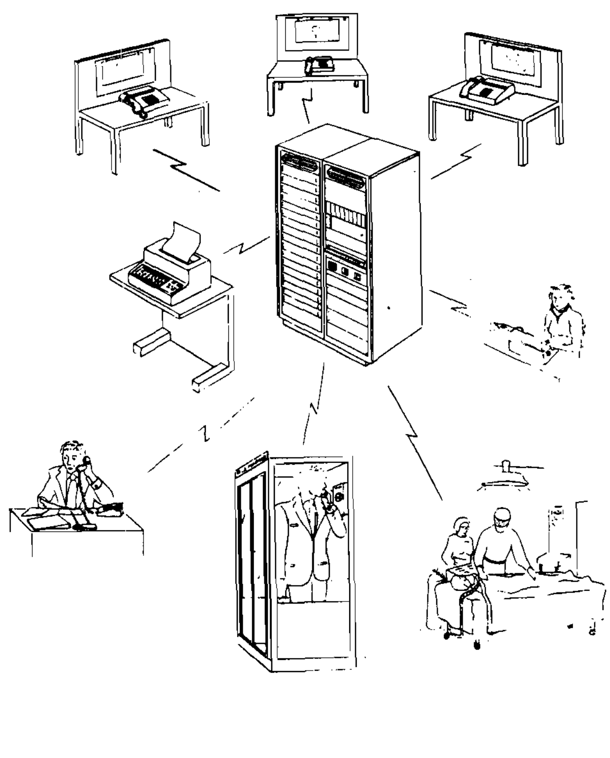


Figure 1: **RTAS — Radiologists dictate using a standard dictation handset. Reports are stored in a central computer-controlled Recorak. Reports can be heard by the referring physician from any telephone. A typist or an automatic typewriter (for negative reports) produces a written copy for the patient chart. Housekeeper produces management statistics and report status.** *(This figure used with permission of CRC Critical Reviews in Clinical Radiology and Nuclear Medicine, CRC Press of Boca Raton, Florida.)*

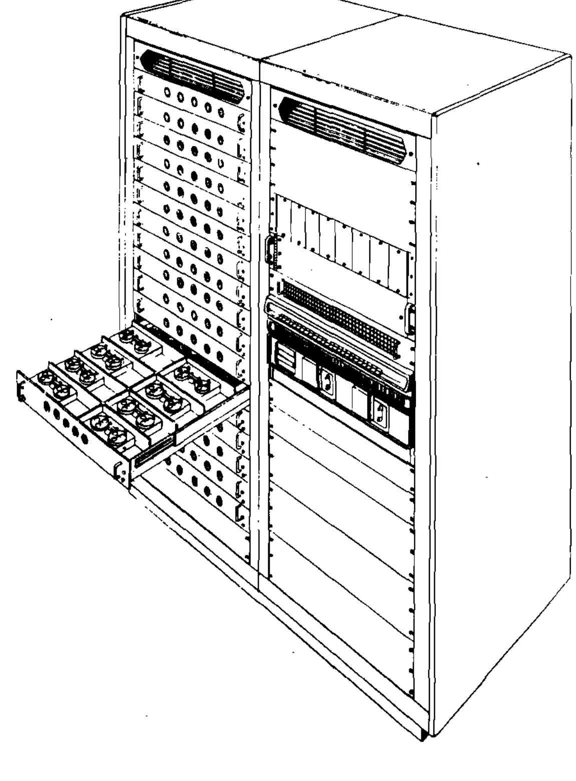


Figure 2: **System Recorak on left and Compurak on right. Recorak consists of multiple independently controlled audio tape Recorpaks with a separate report on each Recorpak. This allows for simultaneous dictation and telephone access for multiple reports. Compurak contains computer controller, interfaces, and associated electronic circuits.** *(This figure used with permission of CRC Critical Reviews in Clinical Radiology and Nuclear Medicine, CRC Press of Boca Raton, Florida.)*

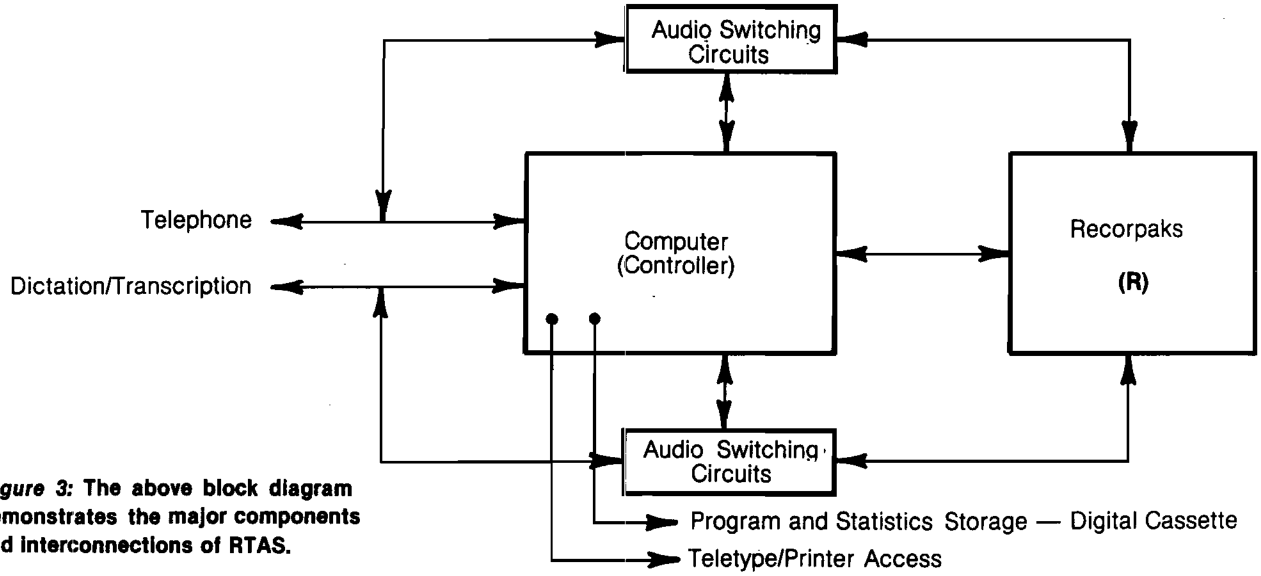


***Figure 3:* The above block diagram demonstrates the major components and interconnections of RTAS.**

are capable of holding five or more minutes of recorded information. Each report is contained on its own recorpak, thus allowing simultaneous and multiple access. Sets of eight recorpaks are assembled in a single, slideout drawer, as shown in figure 2. A full unit contains 16 of these drawers or 128 recorpaks.

Access—The referring physician can access RTAS through a standard touchtone phone. Once the phone is connected to the computer, the patient identification number is entered via the phone pad and the proper recorpak is connected to the phone line, via an audio switching unit.

A transcriptionist retrieves reports for typing through a dictation unit equipped with a conventional footpad and earphone. RTAS keeps track of every report with three parameters: (1) access by transcription unit, (2) an end-of-dictation marker placed on the tape at the end of the dictated report, and (3) termination notice (to computer) when transcription is complete.

Control—The input, output, and storage of RTAS is monitored by a controller, comprised of an audio switching unit and a computer, the 6502 microprocessor.

The switching unit channels the signals it receives from one of the access units (including telephones) to the appropriate recorpak (figure 3). The microprocessor directs operations and stores the information needed to locate the reports. The program is stored on a magnetic tape cassette. This allows reloading the program in case of power failure or mechanical problems.

When the input unit signals RTAS that the radiologist is about to enter a new report, the computer assigns an empty recorpak to the patient's identification code. As long as a report is stored in the system, a file is maintained within the computer.

The computer keeps track of the date and time of a report and its dictation status. If the supply of empty recorpaks becomes short, the system will erase old reports, replenishing the reservoir of recorpaks.

When an output signal is received by the controller, it first searches its file of stored data for the recorpak location. If several reports are listed under the same ID code, the computer arranges their access in chronological order, the most recent first and then plays them sequentially. Only one person can access any particular recorpak at one time.

Future of RTAS

A new version of RTAS is being dveloped. The new RTAS will digitize (A/D conversion) and compress the voice/report and store it on an 80-megabyte disk. A typical system will have four 80-megabyte disks with 360 minutes of report on each disk.

When the reports are replayed, the voice is decompressed and converted to an analog signal (D to A conversion). This allows a high signal-to-noise ratio and eliminates wow, flutter, and tape hiss. By digitizing the voice and storing it as digital data, any report can be accessed at any time by any number of users. The new RTAS no longer needs switching devices and mechanical recorders to record or play back the reports.

Summary

RTAS provides rapid access to dictated reports. Although its use to date has primarily been in radiology, it is by no means limited to this field. Other potential applications include: pathology, cardiology, hospital administration, business information retrieval, and flight information for airline companies.

Correspondence

Please send correspondence to me c/o 9 Brown Place, Woburn, MA 01801.

Bibliography and Acknowledgement

The author is grateful to Gerald Kolodny and Sudbury Systems Incorporated (Sudbury Massachusetts), for providing the above information.

1. Kolodny, Gerald M., RTAS—A cost-effective commercial radiology reporting system. Submitted for publication (1980).

2. Kolodny, Gerald M., A new system of radiology reporting. *CRC Critical Reviews in Clinical Radiology and Nuclear Medicine* 7:187-197, December 1975.

3. Weintraub, H.D., Worcester, J., Resnic, A., et. al. Clinical evaluation of the rapid telephone access system for radiology reporting. *Radiology* 121:349-352, November 1976.

4. Kolodny, Gerald M., Access of radiology reports. *Radiology* 111:597-601, June 1974.

The 6502 Dream Machine

The new generation of 16-bit microprocessors includes upgraded versions of currently popular processors with one important exception. Here is a proposal for a super-6502 as imagined by a software expert.

Randy Hyde
Lazer Systems
Box 55518
Riverside, California 92517

My first experience with a microcomputer was the Zilog Z-80. After all the hype surrounding the chip I was quite disappointed in it. My problem, you see, was that I was used to a somewhat larger computer, such as a PDP-11. I then took a job which required me to work on an Apple II microcomputer. The Z-80 was bad enough, but now I was restricted to 8-bit registers! Needless to say, the powerful 6502 zero page addressing modes more than made up for the lack of 16-bit registers. Still I couldn't help but think, "It sure would be nice to have an HL register pair on this thing." Compromise seems to be the mainstay of chip manufacturers—each chip has its own features, both good and bad.

Lately, however, semiconductor manufacturers have begun listening to the users of their products. The results have been quite encouraging. Intel with their 8086/8088/IAPX286/IAPX432 microprocessors, Motorola with the 6809 and 68000 microprocessors, and Zilog with the Z8000 are definitely paving the way for the new breed of microcomputers. These chips are designed with software development costs in mind. This means the new processors are easier to program and lend themselves to code generation by compilers and other systems rather well.

One nice thing about these new processors is the fact that they are loosely based on their 8-bit brothers which preceded them. The user of an Intel 8080 can view the 8086 as an extremely upgraded 8080 with considerable power. The same holds true for the users of Motorola's 6800 and Zilog's Z-80. It's a real shame that the 6502 user cannot look forward to the 6516 or a "65000" but will have to adapt to the 6809 or 68000 instead. Even though the 6800 family and the 6500 family are quite similar to one another, the change over is not trivial. For example, the zero page indexed addressing modes which give the 6502 much of its power are not available on the Motorola parts. Obviously a 16-bit or pseudo 16-bit 6500-type machine would be highly desirable.

Simple expansion of the 6502's instruction set can be done without going to a new chip. It is possible to add instructions to the 6502 by trapping out invalid opcodes using external decoding ROMs and microcontrollers. In fact, Apple Computer has done exactly that with their new Apple III computer. But adding instructions to the instruction set isn't nearly as useful as adding addressing modes, as well as an orthogonal instruction set. An orthogonal instruction set means that all of the instructions (if applicable) can use all of the available addressing modes. Most manufacturers now realize this and support an orthogonal instruction set on their newer processors. By far the best improvement which can be made to the 6502 is providing it with an orthogonal instruction set.

To begin the discussion of a "dream machine" one must first start with the processor model. My processor model is loosely modeled after the 6516 processor model described in tech notes #34 and #36 from Synertek. My processor model appears in figure one. This model contains ten registers, a 16-bit accumulator which can be split into two 8-bit registers (AH and AL), a 16-bit accumulator extension (AX), three general purpose/index registers, a hardware stack pointer (SP), the program counter (PC), a direct page register (DP), a mode register (Q) and the processor status word (PSW). Additionally each bit in the PSW can be treated as a 1-bit register. The X, Y, Z, SP, and PC registers are all sixteen bits long, the DP, Q, and PSW are eight bits long, and the accumulator can be treated as two 8-bit registers, one 16-bit register, or one 32-bit register (including the accumulator extension).

The X, Y, Z, SP, and PC registers can be used as index registers. In fact this is the primary purpose of the X, Y, and Z registers. SP is the hardware stack pointer where return addresses, etc. are kept. PC is the omnipresent program counter. The accumulator can be broken up into 5 registers. AL is an 8-bit register corresponding to the low order eight bits of the accumulator. AH is an 8-bit register corresponding to bits 8-15. AC is the 16-bit register comprising both AL and AH. AX is the accumulator extension which is a 16-bit register. ACX, or extended accumulator, is the 32 bits made up of AC and AX. AL corresponds to the accumulator in the 6502 register set.

DP is the direct page register. Remember zero page addressing? It only takes two bytes for instructions using this addressing mode. The only problem is that the zero page addressing mode can only refer to 256 bytes. In the 6516 the term "zero page addressing" was changed to "direct page addressing." The direct page register is used to specify which of the 256 pages in memory are to be used for the "direct page addressing mode." This means that all locations in memory can be referred to as a zero page location if the direct page register is properly set up.

The "Q" register is a collection of bits, much like the PSW, which controls the mode of some of the registers. Since compatability with the 6502 is highly desirable, some method of using the X, Y, and SP registers as 8-bit registers must be provided for (remember, some algorithms executing on the 6502 rely upon the 8-bit wrap-around effect). The accumulator can already be treated as an 8-bit register (e.g., AL) so it does not have to be handled specially. The Q register contains a bit for the X, Y, and SP registers as well as memory. If this bit is set, then the corresponding register will be treated as a 16-bit register. If the corresponding bit is reset, then the register will be treated as an 8-bit register. The memory bit is used for such instructions as INC, DEC, ASL, etc. In the 8-bit mode the high order bytes of the X and Y register are ignored.

The stack pointer's high order byte specifies the page in which the stack is to reside (similar to the direct page register for the zero page addressing mode). A fifth bit in the Q register specifies the direction used for auto-increment/decrement. This feature will be described later. The remaining three bits in the Q register are undefined. If a chip of this design ever hits the layout boards I will leave it to the designers to dream up applications for these bits.

The PSW is almost identical to the 6502's PSW except that the undefined bit in position five will be defined to be a user flag/boolean accumulator. In addition, the PSW can be treated as a normal 8-bit register: it can be loaded, stored, transferred, exchanged, etc. Additionally, each bit in the PSW can be treated as a 1-bit register. For example, if the accumulator were loaded from the carry flag, the current contents of the carry flag would appear in bit position zero and all other bits in the accumulator would be set to zero. Likewise, if the accumulator were stored in the carry flag, the current value in bit position zero would be transferred to the carry flag and all other bits would be ignored. This scheme would also allow any condition code flag to be transferred to any other. For example transferring the Z flag (ZF) to the decimal flag is a valid operation.

Now that the processor model is defined, the instruction set can be likewise defined. Basically the only way to get data out of the 6502 is with a store instruction. Data can be brought into the 6502 with a large variety of instructions. For example, LDA, ADC, AND, SBC, ORA, XOR, etc. All take data from main memory, bring it into the 6502, operate on it, and leave the modified data in the 6502. Since we want an orthogonal and consistent instruction set there should be just as many ways of storing data into main memory as there is to load data into the 6502.

This brings up the concept of providing both a destination operand and a source operand in an instruction. On the 6502 the destination (for a load type operation) is usually the accumulator. The new processor should be capable of loading and storing data for most operations. And that brings us to the first "new" instruction: MOV. MOV uses the syntax:

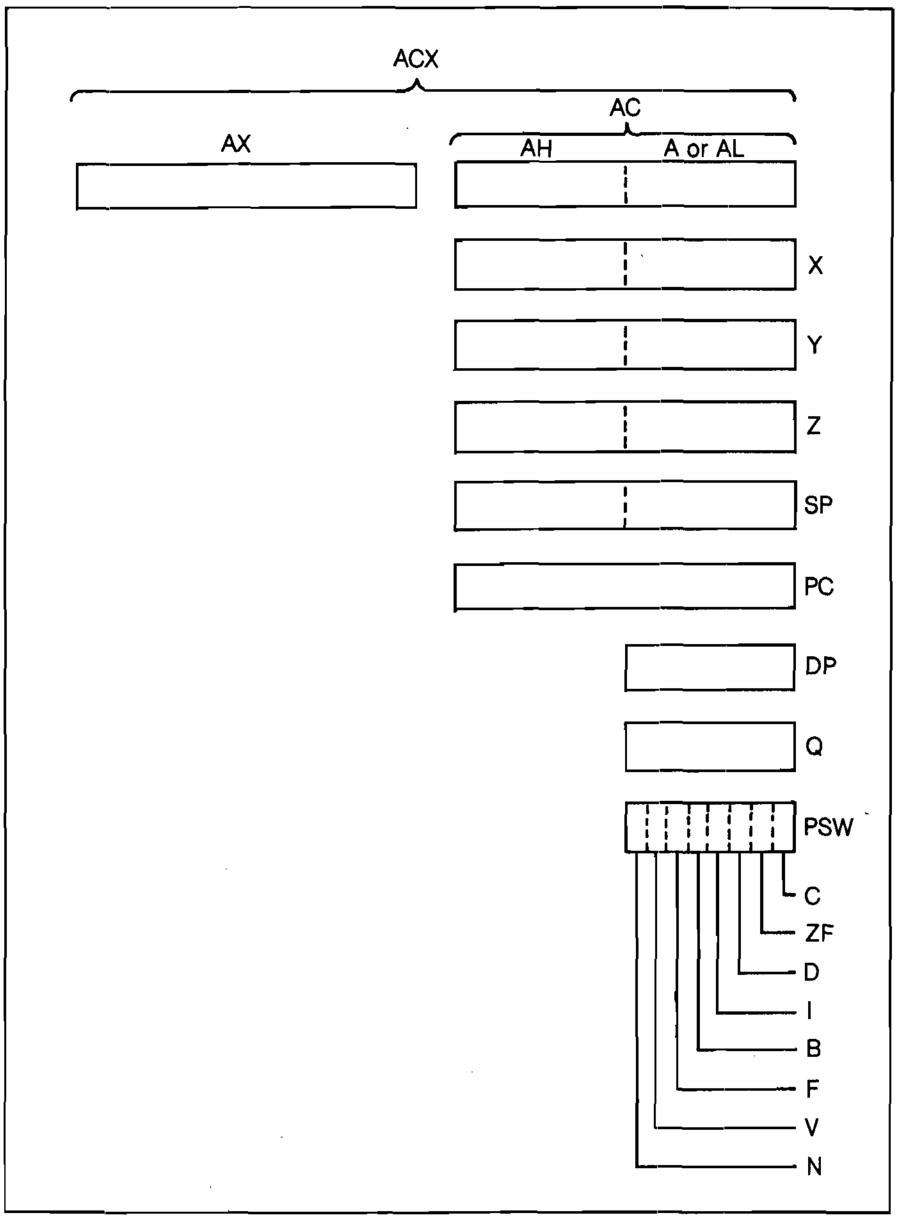
MOV < SOURCE > , < DESTINATION >

The source operand specifies where the data is coming from, and destination specifies where the data is going to. LDA and STA are easily simulated using this instruction.

MOV AL,MEM	STORES ACC (8-BITS) INTO MEMORY
MOV MEM,AL	LOADS THE ACC (8-BITS) FROM MEMORY

The 6502 transfer instructions can also be simulated as follows:

MOV A,X	SAME AS TAX
MOV X,A	SAME AS TXA
MOV A,Y	SAME AS TAY
MOV S,X	SAME AS TSX



The MOV instruction also replaces all of the set and clear flag instructions:

```
MOV #1,C   SEC
MOV #0,C   CLC
MOV #0,V   CLV
MOV #1,D   SED
MOV #1,I   SEI
```

Plus all kinds of combinations not possible on the 6502 are valid (see example 1):

Example 1

```
MOV #1,B        SET BRK FLAG
MOV #0,B        CLEAR BRK FLAG
MOV #1,F        SET USER FLAG/BOOLEAN ACC
MOV #1,V        SET OVERFLOW FLAG
MOV #80,PSW     SET N FLAG, CLEAR ALL OTHERS
MOV X,Y         TRANSFER X TO Y
MOV S,A         TRANSFER S TO ACC
MOV AC,Z        TRANSFER Z REGISTER TO AC
MOV C,A         TRANSFER CARRY TO LSB A, ZERO ALL OTHER BITS (1-7)
MOV ZF,C        TRANSFER CARRY FLAG TO ZERO FLAG
MOV AC,AX       TRANSFER AC TO AX
MOV PSW,A       TRANSFER PSW TO AC
MOV #$FDED,PC   JUMP TO LOCATION $FDED
ETC.            AD INFINITUM
```

Note that it is possible to load the PSW all at once. This will prove to be very handy. The MOV instruction can transfer any register to any other register (with a memory location being treated as a register). Obviously typing "MOV #$FDED,PC" is much more tedious than "JMP $FDED", but that's what macro expanders and extended mnemonics are for.

The source/destination concept applies to several instructions, not just MOV. These other instructions include: ADD (add w/o carry), ADC, AND, SUB (subtract w/o carry), SBC, OR, XOR/EOR, CMP, and EXC. Three new instructions are ADD, SUB, and EXC. ADD and SUB allow you to perform arithmetic without first setting or clearing the carry flag. EXC exchanges the data contained in the two operands (obviously which is the source and which is the destination is of no consequence here). A few examples will probably show the versatility of these instructions (see example 2).

The last example looks somewhat weird; how can you specify immediate data as the destination? Well, whenever immediate data is specified as the destination, the results obtained are discarded. Only the condition code flags are affected. The last example simulates the 6502 BIT instruction (sort of). When the condition code flags are specified, interesting things can happen:

```
OR  C,ZF    OR CARRY AND ZERO FLAGS,
            LEAVE RESULT IN THE ZERO
            FLAG
AND ZF,F    LOGICALLY AND Z AND F
XOR N,V     CHECK FOR SIGNED
            OVERFLOW
```

The last example is particularly useful because it allows you to simulate the signed branches available on the 6800 family devices.

At this point it would be wise to present the instruction set before considering addressing modes. The instruction set is divided into four groups. The first group requires two operands, the second group requires one operand, the third group is the relative branches, and the fourth group comprises several miscellaneous instructions.

Example 2

```
ADD MEM,A       EIGHT BIT ADD MEMORY TO ACC
ADD A,MEM       ADDS ACC TO MEMORY LOCATION, ACC IS LEFT
                UNCHANGED
ADD #2,A        ADD IMMEDIATE (8-BIT) TO ACC
ADD #2,MEM      ADD IMMEDIATE (n BIT, DEPENDS UPON MEMORY FLAG
                IN Q REGISTER) TO MEMORY
ADD A,X         ADDS A (8-BIT) TO X REG (16-BIT)
ADD AC,X        ADDS AC (16-BIT) TO X REG (16-BIT)
ADD X,AC        ADDS X REG (16-BIT) TO AC (16-BIT)
ADD X,A         L.O.  X ADDED TO A (8-BIT)
CMP MEM,A       COMPARE MEMORY TO ACCUMULATOR
CMP A,MEM       COMPARE ACCUMULATOR TO MEMORY
OR  #$80,A      OR ACC WITH #$80
OR  #$80,MEM    OR MEMORY WITH #$80 (I.E., BIT SET)
AND #$FE,MEM    AND MEMORY WITH #$FE (I.E., BIT CLR)
AND MEM,#$80    BIT TEST
```

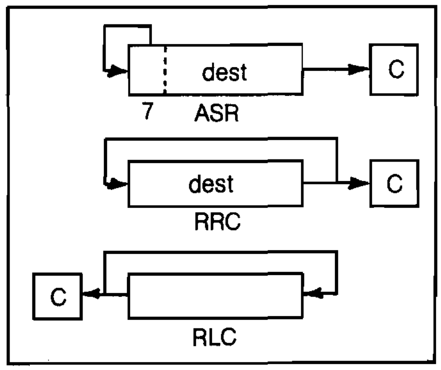
Group One: ADC ADD AND CMP EOR/XOR EXC OR SBC SUB

Group Two: ASL/LSL ASR LSR ROL ROR RRC RLC INC DEC

Group Three: BRA BCC/BLT BCS/BGE BEQ BNE BMI BPL BVS BVC BGT BLE BSB

Group Four: NOP BRK SW0 SW1 LOCK ESC SYNC

The instructions which have no 6502 counterparts are: ASR (arithmetic shift right), RRC (rotate right circular), RLC (rotate left circular), BRA (branch always), BGT (branch if greater than, i.e., C = 1, ZF = 0), BLE (branch if less than or equal, i.e., C = 0 or ZF = 1), BSB (branch to subroutine), SW0 (system routine zero), SW1 (system routine one), LOCK (bus lock), ESC (Coprocessor call), SYNC (Coprocessor synchronization). The last three instructions are useful for multi-tasking situations as well as future instruction set expansion. The effect of these instructions is shown in figure two.



Even if the phrase gets overused, one thing which I must reiterate is the fact that an orthogonal instruction set is all important. The 6502 is woefully lacking in this respect. Some instructions allow one addressing mode but not others, while other instructions allow different addressing modes but not the same as the former instruction. Obviously the first 6502 fix-up is to allow all applicable instructions (i.e., groups one and two) to use all available addressing modes. The next fix is to add several new and useful addressing modes. The addressing modes I propose are:

- REGISTER
- ABSOLUTE
- DIRECT PAGE
- REGISTER INDIRECT
- INDIRECT INDEXED
- INDEXED INDIRECT
- INDIRECT
- AUTO-INCREMENT
- AUTO-DECREMENT
- IMMEDIATE
- INDEXED
- IMPLIED
- RELATIVE

Many of these addressing modes could be combined to obtain hybrid addressing modes. For example ABSOLUTE and INDEXED could be combined to obtain the 6502 ABS,X addressing mode. Two special modes, auto-increment and auto-decrement must be used in conjunction with the indexed or register indirect modes. In the auto-increment mode, the specified index register is normally incremented after the index operation takes place. In the auto-decrement mode, the index register is decremented before the index operation takes place. If you're familiar with the way the 6502 stack works, you'll notice that this algorithm is identical. Which brings us to an interesting point.

I forgot push and pull instructions! Or did I? Since the stack pointer can be used as a normal index register and since the register indirect/auto-increment/auto-decrement modes can be used with any index register, the MOV instruction becomes our push and pull instruction. Simply tell the processor to MOV the contents of the accumulator to the location pointed at by the stack pointer, using the auto-increment mode which simulates a PLA instruction. The PHA instruction can be simulated by MOVing the data pointed at by the stack pointer after it has been decremented by one, using the register indirect/auto-decrement mode. The big advantage of going to all this trouble to push and pop the accumulator is the fact that you are no longer forced to push just the accumulator or PSW. Any register can be pushed, any memory location can be pushed, any immediate value can be pushed, and any single condition code can be pushed. Furthermore, any one of these values can be popped as well. More information on pushes and pops as the utlity of the addressing modes unravels.

The Q register contains a bit which affects how the auto-increment and auto-decrement feature works. If the bit is clear, then the auto-increment/auto-decrement feature works exactly as was just described. In this mode the hardware stack pointer is completely compatable with the 6502 stack pointer. If the auto-increment/auto-decrement bit in the Q register is set, however, the auto-increment mode causes the specified register to be incremented before the indexing operation. For the auto-decrement operation, the register is decremented after the indexing operation takes place.

One last detail on the auto-increment/decrement modes: since the processor works with both 16- and 8-bit quantities, a double increment and double decrement mode are provided.

Now that we have all of these wonderful addressing modes, how do we specify them in an assembly language program? The register addressing mode is obtained by specifying one or two registers. For example "MOV A,X" uses the register addressing mode, as does "ASL A". The defined registers follow.

8-bit operation specifiers:

A or AL	Specifies the L.O. 8-bit accumulator
AH	Specifies the H.O. 8-bit accumulator
P	Specifies direct page register
Q	Specifies Q register
PSW	Specifies program status word

16-bit operation specifiers:

AC	Specifies L.O. 16-bit accumulator
AX	Specifies accumulator extension
X	Specifies 16-bit X register
Y	Specifies 16-bit Y register
PC	Specifies 16-bit program counter
SP	Specifies 16-bit stack pointer

32-bit operation specifier:

ACX	Specifies 32-bit accumulator

The Immediate Addressing Mode

Instructions in this class would probably range from two to as many as six bytes in length. This addressing mode is best handled on an example basis (see example 3). If the X, Y, or SP register is programmed to act as an 8-bit register, the high order byte is still loaded with the immediate data. In the case of the X and Y index registers, the high order byte is ignored; in the case of the stack pointer, the high order byte specifies in which page the stack is to reside.

Example 3

MOV #$80,A	2-3 BYTE INSTRUCTION. LOADS $80 INTO BITS 0-7 OF THE ACCUMULATOR.
MOV #$80,AX	2-3 BYTE INSTRUCTION. LOADS $80 INTO BITS 8-15 OF THE ACCUMULATOR.
MOV #$800,AC	3-4 BYTE INSTRUCTION. LOADS $800 INTO BITS 0-15 OF THE ACCUMULATOR.
MOV #$800,ACX	4-5 BYTE INSTRUCTION. LOADS $800 INTO BITS 0-31 OF THE ACCUMULATOR.
MOV r,#n	CONDITION CODE FLAGS ARE SET ACCORDING TO DATA TRANSFER. DATA IS IGNORED. NOTE: r IS ANY SOURCE, n IS ANY VALUE.
MOV #$80,X	3-4 BYTE INSTRUCTION WHICH LOADS $0080 INTO BITS 0-15 OF THE X REGISTER.
MOV #$80,Y	3-4 BYTE INSTRUCTION WHICH LOADS $0080 INTO Y REGISTER.
MOV #1,C	2 BYTE INSTRUCTION WHICH LOADS $1 INTO THE CARRY FLAG.

AN ENTIRE STAR FLEET FOR $39.95. YOU'LL LOVE IT!

Like some future starship admiral hurtling through the vast void of hyperspace at speeds beyond comprehension, you are challenged to a battle for cosmic supremacy.

THE WARP FACTOR, the latest computer strategy game from SSI, is what every space war fan has been waiting for – the ability to command a star fleet in realistic battle simulation against alien vessels.

It is light years ahead of all other "space" games because it doesn't just fill your screen with pretty pictures and little substance. THE WARP FACTOR is a high-powered tactical simulation that places you squarely in the Captain's role, dealing with the critical parameters of interstellar battle such as sensor and scanner readings; energy allocation for weapons (phasers, disruptor bolts, photon and plasma torpedoes), shields, and warp engines; and battle damage.

THE STARSHIPS. With twelve different starship designs – ranging from dreadnoughts and fighters to star bases and base stations – representing five Galactic Empires, you can set up an astronomical variety of confrontations against another player or the computer.

Each class of vessels is awarded a point value to reflect its relative strength so you can assemble fleets of comparable power for a balanced game. Of course, your're free to play the intrepid hero against seemingly hopeless odds –perhaps mere fighters against a star base!

Employing up to ten ships, both sides can give individual or fleet orders, the latter allowing all your ships to execute your commands in unison.

THE COMPUTER, aside from being the game's perfect administrator and referee, also serves as your ever-ready, ever-capable nemesis in the multiple solitaire scenarios provided: The Reman Chase (replete with the Cloaking Device, Plasma Torpedoes, and Neutral Zone); Attack on Star Base; Attack on Base Station; and Dogfight.

THE TWO-PLAYER VERSION is essentially free-form. With each player choosing starships from a different Empire, you can create scenarios ranging from space skirmishes to a full-scale, all-out star war!

FOR $39.95, THE WARP FACTOR is undeniably the most complete and detailed simulation of tactical starship combat yet designed. It comes with the 5¼" program disc; a Starship Operating Manual; 3 Starship Data Cards; and a Game Selection Card – all of which will convert your computer into the gateway to galactic adventure.

THE WARP FACTOR™. The Universe Awaits Your Command.

Credit card holders, if you own an Apple®II 48K (Applesoft ROM) and a mini-floppy disc drive, call **800-227-1617 ext. 335 (toll free)** and charge your order to your VISA or MASTERCARD. In California, call 800-772-3545, ext. 335.

To order by mail, send your check to: Strategic Simulations Inc, Dept. M, 465 Fairchild Drive, Suite 108, Mountain View, CA 94043.

All our games carry a 14-day money-back guarantee.

While you're at it, you can also get our other games:

FOR YOUR APPLE®:

- ☐ **Computer Bismarck:** $59.95.
- ☐ **Computer Ambush** (a tactical simulation of man-to-man combat in WWII): $59.95.
- ☐ **Computer Napoleonics**, the Battle of Waterloo: $59.95.
- ☐ **Computer Quarterback** (a real-time strategy football game): $39.95.
- ☐ **Computer Conflict** (two modern-day tactical warfare simulations featuring *REBEL FORCE* and *RED ATTACK!*): $39.95
- ☐ **Computer Air Combat** (a simulation of air combat in WWII): $59.95.

FOR YOUR TRS-80®:

- ☐ **Computer Bismarck**, 48K Disc: $59.95. 32K Cassette: $49.95.

Apple is a registered trademark of Apple Computer Inc. *TRS-80 is a registered trademark of Tandy Corporation.*

The Indexed Addressing Mode

To specify the indexed addressing mode simply follow an address with the desired index register enclosed within the square brackets. For example:

```
MOV LBL [X],A    SAME AS "LDA LBL,X"
MOV A,LBL [X]    SAME AS "STA LBL,X"
ADC LBL [Y],A    SAME AS "ADC LBL,Y"
ETC.
```

To use the auto-increment mode simply specify "+" after the index register. To specify the double auto-increment mode use a "++" after the index register. The auto-decrement mode is specified by using "−" instead of "+". Examples:

```
MOV LBL [X+],A
MOV LBL [X++],A
MOV A,LBL [X+]
MOV A,LBL [X++]
MOV A,LBL [X-]
MOV A,LBL [X--]
ETC.
```

The indirect indexed and indexed indirect addressing modes would be specified:

```
MOV A,(LBL [X])
    (6502 = "STA (LBL,X)")
MOV A, (LBL)[Y]
    (6502 = "STA (LBL),Y")
MOV (LBL [Y]),A
MOV (LBL) [X],Z
MOV (LBL [Z]),A
MOV A,(LBL) [Z]
```

Note that there are no restrictions on the usage of the registers as with the 6502. In fact you could have the strange looking but perfectly valid:

```
MOV X,(LBL [X])
MOV (LBL) [X],X
MOV A,(LBL [SP])
MOV A,(LBL) [SP]
```

Also, both the direct page and absolute addressing modes should be supported.

A special addressing mode has been included to handle the special case "MOV A,$0 [X]". This addressing mode is known as "register indirect" addressing mode. It is specified as follows:

```
MOV A,[X]
MOV [X],A
MOV [Z],X
MOV [Y],[X]
ETC.
```

Note that both the source and destination can take advantage of all addressing modes at all times.

Any form of the indexed addressing mode can include the auto-increment or auto-decrement mode simply by specifying "+", "++", "−", or "−−" within the "[" and "]".

The Indirect Addressing Mode

At any time you can use the contents of two consecutive memory locations as an indirect address. Simply enclose the address of the first memory location in parentheses and the indirect addressing mode will be used. Examples:

```
MOV A,(ADDRS)
MOV (ADDRS),A
MOV (ADDRS),PC
    (6502 = JMP (IND))
ETC.
```

The indirect indexed and indexed indirect addressing modes have already been considered.

The Relative Addressing Mode

As with the 6502, the relative addressing mode is used with the branch instructions. The jump to subroutine (JSR) also has a relative addressing mode (for when the BSB is out of range). Many readers will groan because a 16-bit relative jump has not been provided. After all, it would be nice to have relocatable code. But fear not, a relocatable jump has been provided—you just probably did not realize it.

```
ADD #LABEL-*,PC
```

does just the trick. This adds the displacement to the address "LABEL" to the program counter, thus giving you a jump relative instruction. Combined with all the other addressing modes possible, you wind up with an incredibly powerful JMP relative instruction. This concept can also be used to load relative effective addresses into any of the registers.

The Implied Addressing Mode

The implied addressing mode is only used with seven instructions. They are: NOP, BRK, SW0, SW1, ESC, and LOCK. These instructions will be discussed next.

The New Instructions

The new instructions provided in this instruction set are MOV, EXC, LSL, ASR, RLC, RRC, INC, DEC, BRA, BGT, BLE, SW0, SW1, LOCK, ESC, and SYNC. MOV has already been beaten into the ground so it will not be discussed any further here.

The EXC (exchange) command allows you to exchange data between any two operands. Besides its obvious programming benefits, such as applications in sorting, etc., this instruction is absolutely required for operating system Semaphore operations. Any multi-tasking system with device arbitration requires an instruction such as EXC.

The LSL, ASR, RLC, and RRC instructions perform the operations shown in figure two. These instructions complete the 6502 shift and rotate group. These instructions are group two instructions, requiring only one operand.

INC and DEC are used to increment any of the registers or memory. If one of the sub-accumulator registers is specified, then that register (however large) gets incremented. If the Z register is specified, then a 16-bit increment always occurs. If memory, the X, Y, or SP register is specified, then an 8-bit increment/decrement is performed if the corresponding bit in the Q register is clear, and a 16-bit increment/decrement is performed if the corresponding bit is set. At this point, unfortunately, our orthogonal instruction set breaks down as the processor cannot increment or decrement the program counter, direct page register, Q register, or PSW. Fortunately (with the possible exception of the direct page register) there is no need for this type of instruction.

BRA, BGT, BLE, and BSB are additional branches added to the basic 6502 instruction set. BRA is a branch always instruction, BGT is a branch if greater than (C = 1 and ZF = 0), BLE is a branch if less than or equal (C = 0 or ZF = 1), and BSB is an 8-bit relative branch to subroutine.

SW0 and SW1 are system software calls. They are very similar in operation to the BRK instruction, except that the PSW is not pushed onto the stack automatically, and they vector through locations $FFF6 and $FFF8.

The LOCK instruction causes a pin on the processor to go low throughout the execution of the following instruction. Peripheral devices and other processors on the bus must check this line and not perform any DMA or interrupt operations until the pin goes high again. This allows the EXC instruction to function as a Semaphore set, and test instruction without fear of being voided by a DMA operation.

The ESC instruction is an instruction stolen directly from the Intel 8086/8088. This instruction is basically a NOP to the processor. But an external "Coprocessor" can recognize this opcode and provide a sequence of instructions external to the processor. This allows the expansion of the instruction set with outside hardware. Uses include a hardware floating point instruction set, the missing multiply and divide instructions, or any other user-definable instruction sequence. The SYNC instruction is executed to synchronize external processors.

Instruction Set Deficiencies and Other Problems

Despite the obvious advantage of an orthogonal instruction set, some problems do surface. Group two instructions (the read/modify/write instructions) cannot operate on the PC, DP, Q, and PSW registers. Also the ACX (32-bit accumulator) can only be accessed via the MOV instruction. Other operations are undefined for this register. Additional operations are not supported, due only to the current size of the instruction set. Adding 32-bit operations would double the size of the instruction set, making this computer impractical from a hardware point of view.

The 32-bit moves are quite useful since floating point operations are typically handled via "floating point accumulators" kept in page zero. The 32-bit move instructions allow the user to easily move data into the zero page floating point accumulators from external memory. The immediate addressing mode has not been fully defined in this paper because of the problems concerning 6502 compatability with the 8/16-bit X and Y registers. I leave that problem to the microprocessor designer to solve. Ultimately there will have to be an 8-bit MOV instruction for the X and Y registers, as there is for the accumulator. Users of the Zilog Z-80 chip will probably moan the absence of the block move and compare instructions, as well as the bit set, test, and clear instructions. Well, the set, test, and clear instructions are provided with the AND and OR instructions.

Simulating a bit set instruction:

```
OR #%011,MEM     SETS BIT TWO AND
                 THREE
```

Simulating a bit clear instruction:

```
AND #$FE,MEM     CLEARS BIT ZERO OF
                 MEM
```

Simulating a bit test instruction:

```
AND MEM,#$FE     TEST BIT ZERO OF
                 MEM
```

The block move and compare instructions can easily be simulated using the auto-increment/decrement addressing modes. While block move and compare instructions are very useful, they are specialized instructions which do not fit into an orthogonal instruction set. As such they are not included in my instruction set. They are, however, perfect candidates for inclusion in a coprocessor instruction set.

The floating point operations, as well as multiply and divide instructions, are not provided because their utility is not high enough to warrant their inclusion in the basic instruction set. Once again, though, these instructions are so useful that their inclusion in a coprocessor's instruction set is almost mandatory. Since these operations are not required in many applications (such as controllers, word processors, etc.) forcing all users to bear the cost of the added instructions is not practical. The users who require these instructions will gladly bear the additional cost of the coprocessor.

I can say that my instruction set is totally complete, because if you have some "pet" instruction which I have not implemented you can always create a coprocessor to implement the desired instruction. This gives this processor a totally universal appeal. In fact, the first coprocessor (hypothetically) designed for this processor would simply be a unit with a writable control store, allowing users to load in their own microcode, thereby defining the applications as their current needs dictate.

About all you would really need to make this processor perfect is 20 to 24 lines of address bus, a memory manager, and possibly a BASED addressing mode. But of course, with all these features you are talking about a gigantic chip which would be very expensive to manufacture.

Ah Yes! What About the Hardware?

Obviously a chip with the software power described in this paper is going to require more hardware sophistication than is provided on the 6502 chip. First of all the ESC, LOCK, and SYNC instructions require special pins on the processor. A pin is required to inform a coprocessor that its services are required. This pin, defined as an output signal, would become active whenever the processor determined that the ESC opcode had been encountered. At this point, the address bus would contain the address of the byte immediately following the ESC code, the R/W line would be in the READ state, and the data bus would contain the value of the byte immediately following the ESC instruction. The coprocessor(s) would look at the data on the data bus to determine a course of action.

Another pin, LOCK, would go active throughout an instruction execution if the LOCK instruction were the previously executed instruction. Peripherals and other processors on the bus would look at this signal. If active, DMA and interrupts would not be allowed. This allows Semaphore operations to occur with the proper arbitration. LOCK is an output only signal. Another pin, DMA, stops the processor whenever a bus cycle is requested from an external device. The two pins, LOCK and DMA, in conjunction with a DMA ACK signal, allow the processor to be used in a multi-processor configuration.

Another nice feature to have is the ability for external hardware to access all of the registers (eight bits at a time) on the processor chip. This would allow coprocessors to operate on the accumulator, use the indexed addressing modes, set program status, etc. A single pin, when active, would cause the processor to look at the address bus (which is now bi-directional). The low order four bits would contain a register number. If a read operation were requested, the value of the desired register would appear on the data bus; if a write operation were requested, the data on the address bus would be placed in the desired register.

High Level Language Support

All modern processors have been designed with the execution of high level languages in mind. My processor has been designed with making the task of assembly language programming much simpler. As a by-product, this processor that I have defined allows the efficient compilation of high level languages as well. The orthogonal instruction set, the auto--increment/decrement modes, and especially the ability to treat the condition codes as 1-bit registers are the main reasons high level languages are easy to support on this processor. Consider the Pascal segment:

```
If (M = N) and (L > = J) OR (R < >S)
THEN BEGIN
    L: = I + K;
    M: = 0;
END
ELSE M: = 1;
```

If this short segment were coded up using 6502 assembly language, quite a bit of code would be required. Using the processor I've just defined, the code sequence becomes (with macros to make it look more like the 6502):

```
JMP MACRO ADDRS
    MOV    #ADDRS,PC
    MEND

; IF (M = N) AND (L    = J) OR (R      S)
THEN BEGIN
;
    CMP N,M
    MOV ZF,F
    CMP J,L
    AND C,F
    CMP S,R
    EOR #1,ZF
    OR ZF,F
    BFC LBL

;
; L := I + K;
;
    MOV I,L
    ADD K,L
;
; M := 0;
;
    MOV #0,M
    JMP LBL0
;
; ELSE M := 1;
;
LBL MOV #1,M
LBL0:
```

Notice how the orthogonal instruction set improves the efficiency tremendously. Since all addressing modes are available for both the source and destination operands, memory-to-memory operations are possible. Thus register usage is avoided altogether here. I cannot give you a byte count for the above sequence since I have no idea how long the opcodes are for the above instructions.

The processor I've defined can simulate a register machine as well as a memory-to-memory machine. It can also simulate a stack machine such as the P-code machine used by the UCSD Pascal System. Since the auto-increment/decrement modes allow all of the index registers (X, Y, Z, SP, and PC) to be used as stack pointers, a very powerful push down automata can be simulated. In a stack machine there is usually a stack pointer which points to the evaluation stack. All operations on a stack processor usually affect only the top one, two, or three elements of the stack. For example, an ADD instruction on a stack machine would pop the two top elements off the top of the stack, add them together, and then push the result back onto the stack. Other binary operations (AND, OR, EOR, etc.) would function in an identical manner. Monadic functions (such as negate, the shifts and rotates, etc.) operate only on the element on the top of the stack.

Thanks to the auto-increment/decrement modes and the programmable bit in the Q register, it is very easy to simulate a stack machine on the processor I have defined. First, the bit in the Q register must be programmed so that the auto-increment mode performs the indexed operation, and then increments the index register. The auto-decrement mode first decrements the index register and then performs the indexed operation. This causes the stack pointer (or other index register) to always point at the element on the top of the stack, not to the next available element. Now, to simulate a stack machine ADD instruction, one would use the instruction "ADD [SP + +],[SP]". This instruction would take the element on the top of the stack, increment the stack pointer by two, and add the value to the new top of stack, leaving the result on the top of the stack. This example assumes that the memory bit in the Q register was programmed for 16 bits. If it were programmed for eight bits the "ADD [SP +],[SP]" instruction would be used. To push data onto the stack you would use the MOV instruction. For example: "MOV #$800,[SP – –]" pushes the 16-bit constant $800 onto the stack (assuming memory is programmed for 16 bits). Generally, one would use a register other than the hardware stack pointer for the evaluation stack. The Z register (since it has no equivalent on the 6502) is probably a good choice. Simply substitie "Z" for "SP" in the previous examples; e.g., "ADD [Z + +],[Z]".

Dreaming vs. Reality

I feel that I have defined quite a nifty little processor here. Unfortunately it has been simply a mental exercise, since there are no plans in the works for implementing the 6516 or a 65000 at this time. Remember, the 6502 was originally designed as a "powerful and inexpensive" microprocessor. For its time it filled that application rather well. The 6502 was released with a $20 price tag when the 6800 was going for $60-$80 and the 8080 was still $100 + . The reason the 6502 became so popular was due to its application in video games such as the Atari home video unit, etc. The 6502 "cult" which developed is probably responsible for less than 400,000 microprocessors (and that figure includes all Apples, PETs, KIMs, AIMs, SYMs, OSIs, etc.) whereas the game and low-cost controller market has been responsible for several million processors. The processor I have just defined is really overkill for the larger application, and the 500,000 or so "hobby" applications (assuming that everyone would switch to this processor—which is unlikely) simply isn't worth the effort. The semiconductor manufacturers are much better off supporting the existing line and making even more money on a sure bet.

Still it would be nice to have a couple of million dollars for development and Chuck P. working for me!

Randall Hyde is a software/hardware engineer and partial owner of Lazer Systems in Riverside, CA. He wrote the highly popular LISA interactive assembler for the Apple II and is the author of the book *Programming the Apple II Using 6502 Assembly Language*. Randy is also a consultant to Programma International, in charge of hardware development and production.

MICRO

MICRO Classified

Super Fast

Assembler for the Apple II. 1000 line program assembled in less than 1 second! A powerful editor included with the assembler for updating source program. Available in cassette tape or disk. Requires 48K. Disk requires 3.3 PROM. Send $25.00 to:

Thomas Wong
39-55 51st Street
Woodside, New York 11377

C1P Extended Monitor

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

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

OSI Software

High quality programs for OSI featuring Super Graphics, Fast Action, Free Shipping and Color and Sound (for C4) at no extra cost! Write for descriptive list.

Bob Retelle
2005 Whittaker Street
Ypsilanti, MI 48197

Accounts Receivable by SBCS

for the Apple II. This conversion of Osborne's Accounts Receivable software contains the same capabilities, plus many enhancements that increase your Apple's flexibility, speed, and performance. Use alone or integrate with existing General Ledger program. Retail price $180.

Small Business Computer Systems
4140 Greenwood
Lincoln, NE 68504

OSI Input, Output, Timer

Add inputs, outputs, and timers (6522 VIA) to your Superboard II or C1P. No wiring changes needed; it plugs into your expansion jack. Program it in BASIC or Assembly Language. Assembled & Tested $26, Bareboards $8, Shipping $2.

SB2 Accessories
RR1 Box 99
Greenfield, IN 46140

Programmer Fatigue?

SYM—BUG/MONEX adds 15 commands to SYM's repertoire including an interactive trace/debug. Cassette @ $0200 or $3800: $19.95. EPROM (2716-5v) @ $F000-$F7FF: $39.95. Commented source listing: $9.95. RAE-1(/2) FORMAT CASSETTE: $35 (requires 8K). Custom assembly add $2.00. Foreign add $2.00. SASE for more information.

Jeff Holtzman
6820 Delmar-203
St. Louis, Missouri 63130

AIM/KIM/SYM

NBS Computing gives you time! A battery backed-up clock-calendar board that runs on the application bus. The clock will run for months without power and can generate interrupts on SYM systems. $69.95 assembled, $34.95 bare board. Both include drivers.

NBS Computing
1674 E. M-36
Pinckney, Michigan 48169

PET Machine Language Guide

Comprehensive manual to aid machine language programmer. More than 30 routines are fully detailed so that the reader can put them to immediate use. OLD or NEW ROMS. $6.95 + .75 postage. VISA & Mastercharge accepted.

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

OHIO SCIENTIFIC

Animated Moon Lander game for C1 and C4. View a series of high resolution looking lunar vistas as you descend from 120 miles. Accurate instrument readings. A cartoon landing sequence rewards your successful landing. See what OSI graphics can do! Other programs available. $9.95 cassette 8K; $12.45 disk 24K.

Earthship
P.O. Box 489
Sussex, New Jersey 07461

OHIO SCIENTIFIC

Catchword, a multi-player, competitive word game uses the computer to generate letters and point values. Letters are up for grabs by any player. The idea is to make as many words as possible in crossword form on your board. The computer recognizes and penalizes you for all sorts of sloppy playing. C1 and C4. $9.95 cassette 8K; $12.45 disk 24K. Other programs available.

Earthship
P.O. Box 489
Sussex, New Jersey 07461

Spanish Hangman

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

George Earl
1302 South General McMullen
San Antonio, Texas 78237

Save Money — You Can!

By building your own computer interfaces. 80% savings. Send $3.95 for simple how-to package today.

ADS
Box 9770
Jacksonville, Florida 32208

The Newest In

Apple Fun

We've taken five of our most popular programs and combined them into one tremendous package full of fun and excitement. This disk-based package now offers you these great games:

Mimic—How good is your memory? Here's a chance to find out! Your Apple will display a sequence of figures on a 3×3 grid. You must respond with the exact same sequence, within the time limit.

There are five different, increasingly difficult versions of the game, including one that will keep going indefinitely. Mimic is exciting, fast paced and challenging—fun for all!

Air Flight Simulation—Your mission: Take off and land your aircraft without crashing. You're flying blind —on instruments only.

A full tank of fuel gives you a maximum range of about 50 miles. The computer will constantly display updates of your air speed, compass heading and altitude. Your most important instrument is the Angle of Ascent/Bank Indicator. It tells if the plane is climbing or descending, whether banking into a right or left turn.

After you've acquired a few hours of flying time, you can try flying a course against a map or doing aerobatic maneuvers. Get a little more flight time under your belt, the sky's the limit.

Colormaster—Test your powers of deduction as you try to guess the secret color code in this Mastermind-type game. There are two levels of difficulty, and three options of play to vary your games. Not only can you guess the computer's color code, but it will guess yours! It can also serve as referee in a game between two human opponents. Can you make and break the color code...?

Star Ship Attack—Your mission is to protect our orbiting food station satellites from destruction by an enemy star ship. You must capture, destroy or drive off the attacking ship. If you fail, our planet is doomed...

Trilogy—*This contest has its origins in the simple* game of tic-tac-toe. The object of the game is to place three of your colors, in a row, into the delta-like, multi-level display. The rows may be horizontal, vertical, diagonal and wrapped around, through the "third dimension". Your Apple will be trying to do the same. You can even have your Apple play against itself!

Minimum system requirements are an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive. Mimic requires Applesoft in ROM, all others run in RAM or ROM Applesoft.

Order No. 0161AD $19.95

Solar Energy For The Home

With the price of fossil fuels rising astronomically, solar space-heating systems are starting to become very attractive. But is solar heat cost-effective for you? This program can answer that question.

Just input this data for your home: location, size, interior details and amount of window space. It will then calculate your current heat loss and the amount of gain from any south facing windows. Then, enter the data for the contemplated solar heating installation. The program will compute the NET heating gain, the cost of conventional fuels vs. solar heat, and the calculated payback period—showing if the investment will save you money.

Solar Energy for the Home: It's a natural for architects, designers, contractors, homeowners...anyone *who wants to tap the limitless energy of our sun.*

Minimum system requirements are an Apple II or Apple II Plus with one disk drive and 28K of RAM. Includes AppleDOS 3.2.

Order No. 0235AD (disk-based version) $34.95

Math Fun

The Math Fun package uses the techniques of immediate feedback and positive reinforcement so that students can improve their math skills while playing these games:

Hanging—A little man is walking up the steps to the hangman's noose. But YOU can save him by answering the decimal math problems posed by the computer. Correct answers will move the man down the steps and cheat the hangman.

Spellbinder—You are a magician battling a computerized wizard. In order to cast death clouds, fireballs and other magic spells on him, you must correctly answer problems involving fractions.

Whole Space—Pilot your space craft to attack the enemy planet. Each time you give a correct answer to the whole number problems, you can move your ship or fire. But for every wrong answer, the enemy gets a chance to fire at you.

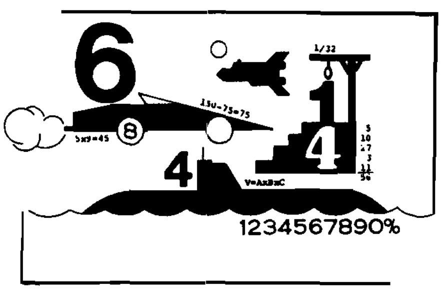
Car Jump—Make your stunt car jump the ramps. Each correct answer will increase the number of buses your car must jump over. These problems involve calculating the areas of different geometric figures.

Robot Duel—Fire your laser at the computer's robot. If you give the correct answer to problems on calculating volumes, your robot can shoot at his opponent. If you give the wrong answer, your shield power will be depleted and the computer's robot can shoot at yours.

Sub Attack—Practice using percentages as you maneuver your sub into the harbor. A correct answer lets you move your sub and fire at the enemy fleet.

All of these programs run in Applesoft BASIC, except Whole Space, which requires Integer BASIC.

Order No. 0160AD $19.95



Skybombers

Two nations, seperated by The Big Green Mountain, are in mortal combat! Because of the terrain, their's is an aerial war—a war of SKYBOMBERS!

In this two-player game, you and your opponent command opposing fleets of fighter-bombers armed with bombs and missiles. Your orders? Fly over the mountain and bomb the enemy blockhouse into dust!

Flying a bombing mission over that innocent looking mountain is no milk run. The opposition's aircraft can fire missiles at you or you may even be destroyed by the bombs as they drop. Desperate pilots may even ram your plane or plunge into your blockhouse, suicidally.

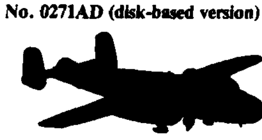
Flight personnel are sometimes forced to parachute from badly damaged aircraft. As they float helplessly to earth, they become targets for enemy missiles.

The greater the damage you deal to your enemy, the higher your score, which is constantly updated at the bottom of the display screen.

The sounds of battle, from exploding bombs to the pathetic screams from wounded parachutists, remind each micro-commander of his bounden duty. Press On, SKYBOMBERS—Press On!

Minimum system requirements: An Apple II or Apple II Plus, with 32K RAM, one disk drive and game paddles.

Order No. 0271AD (disk-based version) $19.95



Paddle Fun

This new Apple disk package requires a steady eye and a quick hand at the game paddles! It includes:

Invaders—You must destroy an invading fleet of 55 flying saucers while dodging the carpet of bombs they drop. Your bomb shelters will help you—for a while. Our version of a well known arcade game! Requires Applesoft in ROM.

Howitzer—This is a one or two person game in which you must fire upon another howitzer position. This program is written in HIGH-RESOLUTION graphics using different terrain and wind conditions each round to make this a demanding game. The difficulty level can be altered to suit the ability of the players. Requires Applesoft in ROM.

Space Wars—This program has three parts: (1) Two flying saucers meet in laser combat—for two players, (2) two saucers compete to see which can shoot out the most stars—for two players, and (3) one saucer shoots the stars in order to get a higher rank—for one player only. Requires Applesoft.

Golf—Whether you win or lose, you're bound to have fun on our 18 hole Apple golf course. Choose your club and your direction and hope to avoid the sandtraps. Losing too many strokes in the water hazards? You can always increase your handicap. Get off the tee and onto the green with Apple Golf. Requires Applesoft.

The minimum system requirement for this package is an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive.

Order No. 0163AD $19.95

Instant Software™

*A trademark of Apple Computer Inc.

PETERBOROUGH, N.H. 03458
603-924-7296

Why Magazine Subscription Rates Go UP, UP, UP!

Due to general inflationary pressures and increased mailing costs, MICRO must increase U.S. subscription rates from $15.00 to $18.00 a year, effective April 1, 1981. The cover price will remain unchanged, however, so that U.S. subscribers will be saving 25% over single-copy purchases.

International Politics Brings 90% Increase in Postal Rates

Small Countries Outvote U.S. and Other Major Publishing Countries

Due to international politics, MICRO is immediately forced to introduce large increases in some foreign subscription rates. MICRO anticipated increases, but was unprepared for the magnitude of the increases in international mailing rates effective January 1, 1981. The rates previously announced by MICRO in January (MICRO 32:47) are therefore superseded by the rates given below.

The gigantic rate increases for international mail result from the Universal Postal Union's vote to increase terminal dues, dues paid from one country to another when the two exchange unequal amounts of mail. The four largest mailers—the U.S., Britain, France, and Japan—opposed the increase but were outvoted by the smaller countries.

MICRO™ Works to Reduce Rates!

To save European subscribers from the full impact of the international rate increase, MICRO has engaged a European airline which will air freight the magazine to Europe and there post it by surface mail. Air Mail subscribers should receive their copies no later than the middle of each month.

MICRO will attempt to make similar arrangements for other geographic areas. If so, subscriptions made at the new rates will be extended by the amount of the savings achieved.

MICRO's New Annual Subscription Rates

(Effective Immediately)

	Current	New
U.S. (Effective 4/1/81)	$15.00	$18.00
International Surface Mail		
Anywhere outside the U.S.	18.00	21.00
International Air Mail		
Europe	33.00	36.00
Middle East	39.00	42.00
Africa		
North	39.00	42.00
Central	39.00	51.00
South	39.00	60.00
Mexico, Central America	27.00	39.00
South America	33.00	51.00
Far East, Australasia	39.00	60.00

MICRO New Publications

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

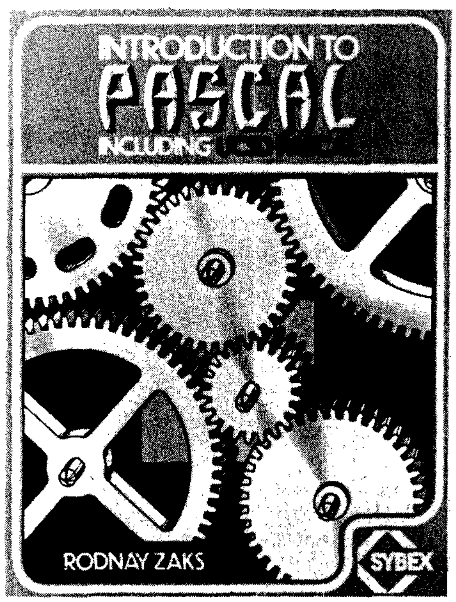
This column lists new publications received for review and also reports on pertinent publication announcements received from book and periodical publishers. Some works mentioned here may be reviewed by MICRO at a later date.

General Computer

An Introduction to Pascal (Including UCSD Pascal) by Rodnay Zaks. Sybex Inc. (2344 Sixth Street, Berkeley, California 94710), 1980, xviii, 422 pages + errata sheet, 131 illustrations, 7 × 9 inches, paperbound.
ISBN: 0-89588-050-4 $12.95

A comprehensive introduction and guide to standard and UCSD Pascals. Designed as a tutorial for both beginners and experienced programmers.

CONTENTS: *Basic Concepts*—Introduction; Computer Programming; Algorithms and Data Structures; Pascal; UCSD and Other Pascals; A Simple Pascal Program; A Second Program Example; Summary; Exercises. *Programming in Pascal*—Introduction; Writing A Pascal Program; The Syntax of Pascal; Format of a Pascal Program; Declarations; The Executable Program Body; Program Organization Summary; Formal Organization of a Program; The Symbols of Pascal; Reserved Symbols; Reserved Words; Standard Identifiers; UCSD Comments; UCSD Program Headings; UCSD Listings; Summary; Exercises. *Scalar Types and Operators*—Introduction; The Integer Type; The Real Type; The Character Type; Operators and Functions for Characters; The Boolean Type; User-Defined Types; Type Declaration; UCSD Long Integers; UCSD Standard Arithmetic Functions; Summary; Exercises. *Expressions and Statements*—Introduction; Expressions; Arithmetic Expressions; Using Standard Functions; Summary of Arithmetic Expressions; Boolean Expressions; Basic Rules of Boolean Algebra; Statements; Summary; Exercises. *Input and Output*—Introduction; Communicating With a File or the Terminal; READ and READLN; WRITE and WRITELN; UCSD Input/Output; Summary; Exercises. *Control Structures*—Sequential Execution; Repetition Statements; Repeat Statement; WHILE and REPEAT; Nested Loops; The Three Loop Statements—A Summary; Conditional Statements; Multiple Choice: Case Statement; Unconditional Branch: GOTO; UCSD Case Statement; Summary; Exercises. *Procedures and Functions*—Program Organization; Procedures; Block Structure and Scope Identifiers; Functions and Procedures as Parameters; Scope Revisited; Recursion Revisited; Recursion Example; Forward References; External Procedures; Restrictions on Parameters; UCSD Procedures and Functions as Parameters; UCSD Packed Variables as Parameters; UCSD EXIT; Summary; Exercises. *Data Types*—Types; Why Data Types?; General Rules for Data Types; Scalar Types; Summary; Exercises. *Arrays*—Data Structures; The Array; Referencing the Elements of an Array; Operating on an Array; Multi-Dimensional Arrays; Array of Characters; Case Study 1: Matrix Addition; Case Study 2: Quicksort; Packed Arrays; Array Valued Functions; UCSD Arrays; UCSD String Type; UCSD Pack and Unpack; Summary; Exercises. *Records and Variants*—Introduction; Record; Formal Definition; Operations on Records; The WITH Statement; Scope Identifiers; Case Study 1: Inventory Management; Case Study 2: Credit Card Number Validation; Variants; UCSD Packed Records; Summary; Exercises.



Files—Basic Definitions; Pascal Files; Formal Definition; Standard Files; Writing on a File; WRITE Summary; Reading a File; Case Study 1: Filemerge, Permanent and Temporary Files, Text Files, Text File Processing, The Input and Output Files; Case Study 2: Cipher Program; Case Study 3: Find Occurrences of a String; UCSD Files; Summary; Exercises. *Sets*—Sets in Pascal; Constructing a Set; Operations on Sets; Case Study: Identifying Characters; Summary; Exercises. *Pointers and Lists*—Introduction; Dynamic Data Structures; Lists; Creating a Dynamic Variable; Accessing an Element of a List; Adding and Removing an Element; Other List Structures; Case Study 1: A Librarian; Case Study 2: A Binary Tree; UCSD Dispose; Summary; Exercises. *UCSD and Other Pascals*—UCSD and Other Versions; Overview of UCSD Pascal; UCSD Units; UCSD Segment Procedure; System-Related Routines; Summary. *Program Development*—The Program Development Process; The Five Steps of Program Development; Writing a Pascal Program; Programming Style; Conclusion. *Appendices:* Pascal Operators; Reserved Words; Standard Functions and Procedures; Standard Identifiers; Operator Precedence; Syntax Diagrams; ASCII Code; UCSD Syntax Diagrams; Usual UCSD Limitations; UCSD Intrinsics; References; Answers to Selected Exercises. *Index.*

Computers and Education

Classroom Computer News is a bimonthly magazine founded in 1980 by Intentional Education (Allston, Massachusetts). It is written by and for teachers and administrators who use or intend to use computers in education, from kindergarten through graduate school. The publisher describes the magazine as covering "educationally significant computer applications, products, research, grants, and governmental programs." A one-year subscription (six issues) is $9.00 from *Classroom Computer News*, Box 266, Cambridge, Massachusetts 02138.

Educational Electronics is a newsletter which will appear monthly beginning March 1981. The publisher states that it will deal with the broad field of technology in education but "focus on the development of computer hardware and software for instructional and administrative purposes". Specific subjects to be covered: information retrieval systems, voice synthesis, speech control, audio-visual equipment, materials for training the handicapped, technology in school and library systems, grants, legislation, and government trends affecting technology. A one-year subscription (twelve issues) will be $50.00 through May 31, thereafter $60.00, from *Educational Electronics*, One Lincoln Plaza, New York, New York 10023.

General Microcomputer

Microcomputer Index (including abstracts) by Microcomputer Information Services (3070 Adams Way, Santa Clara, California 95051), 8½ × 11 inches, paperbound, issued quarterly. First issue published: January-March 1980. Annual subscription, $22.00.

A subject index, including abstracts, of some 20 microcomputer periodicals. According to the publisher, a single issue may employ over 300 index terms and contain as many as 1,000 citations, each with up to 4 descriptors, covering

(continued on page 12)

A Second Cassette for PET

This article contains a description of "how-to" modify a standard cassette recorder to function as a second cassette for the PET. Detailed pictures help in making the necessary modifications easier.

Jerry W. Froelich
9 Brown Place
Woburn, Massachusetts 01801

An inexpensive second tape cassette with a tape counter has been a dream of mine, but at $80 to $100 this was out of my reach. Since I have wondered about using my portable cassette recorder, I decided to interface it to my PET. The following is a description of such an interface and helpful hints for carrying out the implementation.

Interface Hardware

The second tape drive on the older PET is connected to the J3 connector. Although the specifications are in the PET manual I will review the pin connections.

- Pin 1 = Ground
- Pin 2 = +5 volts, on when the PET is turned on.
- Pin 3 = +6 volts, under software control to control cassette motor.
- Pin 4 = READ line from cassette recorder.
- Pin 5 = WRITE line to cassette recorder.
- Pin 6 = SENSE line which is normally open but goes to ground when any button is depressed on the cassette.

Pins 1 and 2 are self-descriptive. Pin 3 is the motor control line and is under software control. This line will not be activated unless the SENSE switch is closed (grounded).

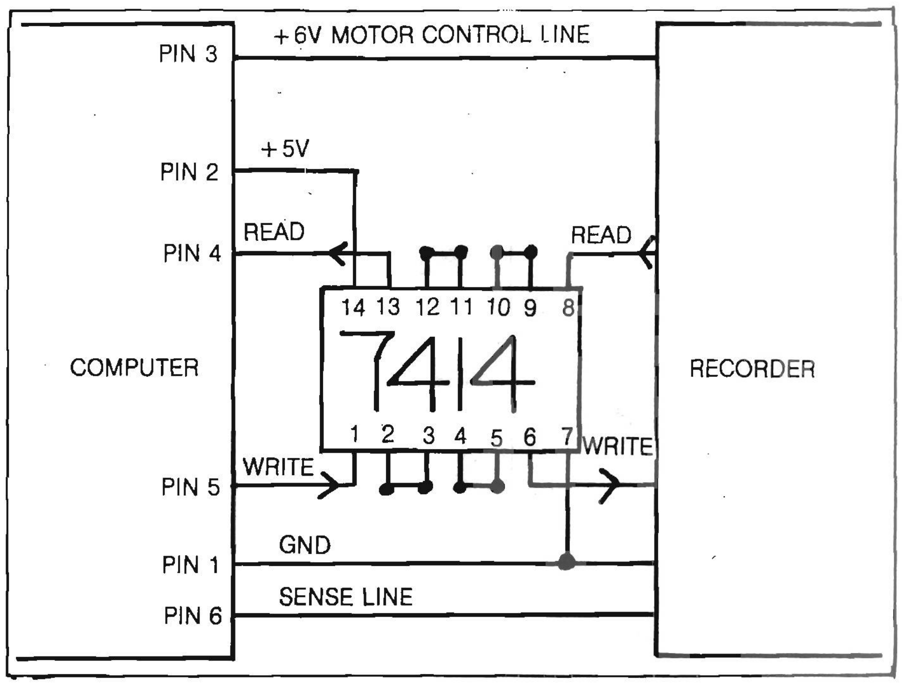


Figure 1: **Schematic representation of the interface connections.**

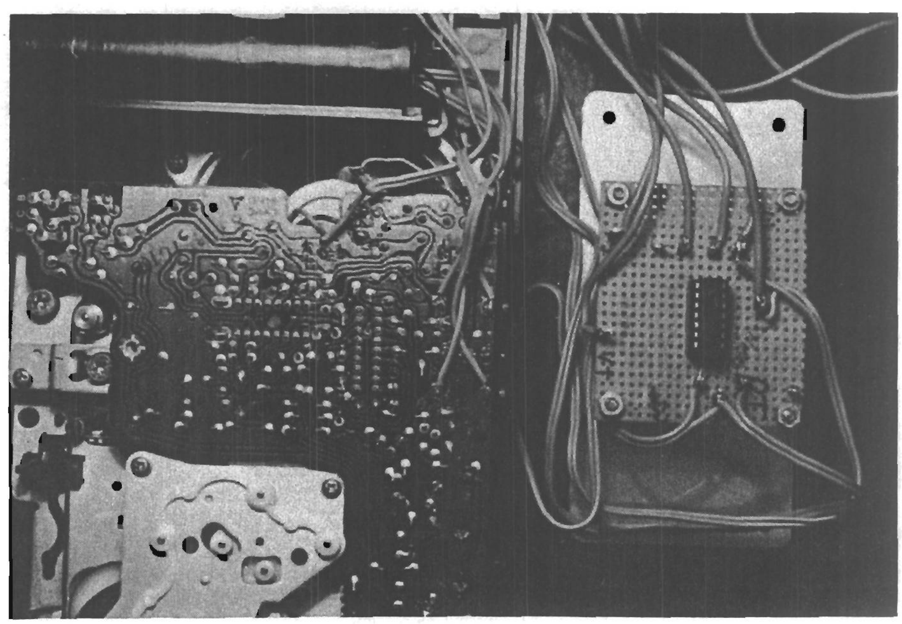


Figure 2: **Back side of the unmodified cassette recorder and the PERF board containing the 7414 IC.**

Pins 5 and 6 are the READ/WRITE lines. The signals on these lines are square waves. The write line comes directly from the 6522 VIA, inverted in a 7414 Hex-schmitt trigger inverter, and then written onto the tape. The signal coming from the recorder is a sine wave which is shaped and inverted by the second half of the 7414.

Pin 6 is the "SENSE" line. When any button is depressed on the cassette recorder, this closes an internal switch and drops this line to ground.

I chose the PANASONIC RQ2765 cassette recorder for interfacing for a variety of reasons: availability, reasonably priced ($39 to $49 in the Boston area), good electrical specifications, internally controlled by 6 volts (same as PET), has a built-in tape counter and has an internal switch which can be converted into the "SENSE" switch.

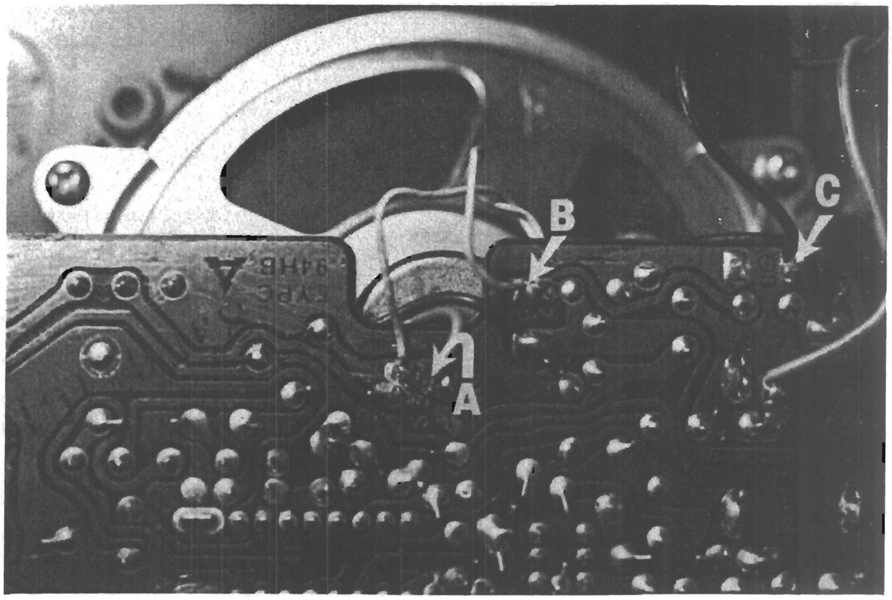


Figure 3: **Blown-up view of the unmodified cassette PC board with key points identified.**

Interfacing the recorder to the PET is relatively simple. First, remove the screws that hold the cassette case together. Carefully separate the halves of the case, but be aware that there are short wires connecting the printed circuit (PC) board to the speaker and battery pack. Figure 1 contains the schematic representation of the interface. Figure 2 shows the rear electronic on the PC board of the recorder and the 7414 chip. Figures 3 and 4 contain the pre- and post-wiring changes to the PC board and will be described in detail.

The following discussion refers to figure 3. On the unmodified PC board point *A* is the +6 volt connecting area for both the motor and the SENSE switch. Point *B* completes the circuit for the sense switch by turning the motor on. Point *C* is the ground terminal. Gently desolder the wire at the left-most aspect of point *A*. The wire at point *B* should be disconnected and reconnected at point *C*. The sense line from the computer should be attached to the free end of the wire which was removed from point *A*. Figure 4, point *A* illustrates the completed connection between the sense switch and the sense line from the computer.

The following description refers to figure 4. Points labeled *g* are the ground lines and should be run to the ground pin 1 of the computer. Point *B* is the +6 volt software controlled line from Point 3 of the computer. Points *C* and *D* are the READ/WRITE lines and are wired to the 7414 Hex-schmitt trigger inverters as drawn in figure 1. I disconnected the speaker wires which are located on the under-side of the PC board.

The 7414 was mounted on a small piece of PERF board and placed in a small external box, with wires connecting to both the computer and recorder. With redesigning the wiring and locating the 7414 on a small board inside the battery area the external box can be eliminated.

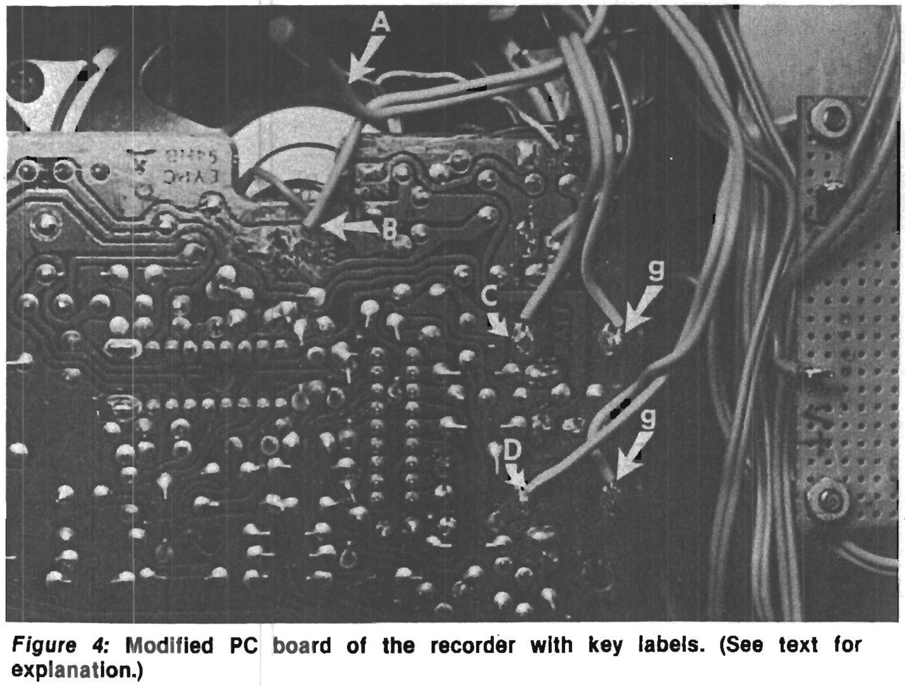


Figure 4: **Modified PC board of the recorder with key labels. (See text for explanation.)**

Using the Recorder

The converted recorder can be used like the standard PET cassette. The tone and volume controls should be set at "10". Although I did not have trouble with the head alignment between the standard cassette and the interfaced cassette, the potential still exists. To test the new cassette, first record and read on the interfaced cassette, and only after debugging the hardware should you read/write on the standard PET cassette.

Conclusion

The following is a description of how to modify an inexpensive cassette recorder to run on the PET computer. The advantages of this configuration are described above and are quite obvious. The hardware approach is definitely easier than the software approach for locating files on tapes.

Happy Recording!

MICRO

In this issue of the Ohio Scientific Small Systems Journal we conclude last month's discussion of Artificial Intelligence. Additionally, we are presenting a chart detailing the 6502 op-code structures and a method for implementing a BASIC trace function under OS-65D.

The Use of Microcomputers in Artificial Intelligence Research

Part Two: The OSI Language Processing Programs

In part one of this article (see reference II) we surveyed the general field of artificial intelligence and examined the problem of understanding written natural language in some detail. We conclude now with a description of experimental language processing work undertaken at Ohio Scientific.

Two experimental programs have recently been written that use some of the techniques described in the previous sections to achieve limited language-processing ability. The first program includes all of the dictionary and grammar features mentioned in part one. The basic dictionary includes about 450 root words, but functionally it is much larger because of several embellishments. First, the "snipping" algorithm of Winograd (see reference 8) has been expanded and implemented in the program. It now extracts root words from words with the following endings: *n't, 's, ', s, ly, ing, ed, en, er, est, 'll, and 've.* Of course, irregularities such as *"won't"* must be entered separately. However, the program is sophisticated enough to process the two distinct uses of the *"er"* ending. Thus, *"runner"* would be analyzed as *"one who runs"*, while *"faster"* is recognized as the comparative form of fast. The dictionary also augments its basic entry list by prompting the user for part of speech information when it encounters an unfamiliar word. The program determines four sentence types (two question types, a command, or a declarative sentence) and, for certain sentences, performs an augmented transition network analysis (see part one of this article) to extract actor, action, and object information. The result is a routine that provides a complete analysis of each word of the input sentence and begins to assemble important semantic information that is needed for response generation.

The second OSI experimental program uses a table-driven key word analysis to extract semantic information from an input sentence. This program then focuses on the response generation phase of language processing. Three elementary knowledge bases are implemented: a personal appointment calendar, a checking account record, and remote control of various home appliances. The techniques used result in a program similar to a data management system in which the query langauge is extremely flexible and "forgiving." This program was developed on OSI's OS-65U operating system. The CPU speed and disk access speed of the C3 systems have resulted in response generation that is approximately real time.

The program must find a key word to get into one of its "knowledge modes", or to change to another mode. Once there, the program will respond correctly to a variety of inputs. For example, "break", "cancel", "delete", "remove", and "forget" can all be used to delete an appointment in the personal calendar. The program recognizes the type of a sentence and responds properly in a variety of situations. For example, the sentence

Is there an appointment with John on Friday

would be recognized as a question (because of its structure; the program ignores punctuation) and the appointment file would be searched to answer correctly. On the other hand, the sentence

There is an appointment with John on Friday

which contains the exact same words as the previous sentence, would be properly recognized as a declarative sentence and the program would respond by adding the specified appointment.

Of course, the dependence on key word recognition limits the program's actual utility in its present form. The sentence

Make an appointment for John on Friday

would not be processed properly because the person's name must be preceded by the preposition "with" to be properly identified. In spite of such limitations, the development of this program still yielded a great deal of insight on many aspects of response generation in understanding natural language.

Obviously, the next step is to put the two experimental programs together into a comprehensive language processor. Much additional development must be done on the semantic analysis phase of the first program. Many of the necessary steps are cumbersome to implement in BASIC. The preferred language for this research, LISP, is presently unavailable.

The ultimate application of the comprehensive language processing capability comes when the computer system also has full real time voice recognition and generation capability. A computer with these communicating abilities brings to mind the famous computer, HAL, from the movie *2001: Space Odyssey.* OSI demonstrated an experimental computer at the AII conference at Stanford this past summer. This computer featured true parallel processing, Winchester Technology, Votrax-synthesized voice output and a real time speaker-dependent voice recognition system.

Although the work is very incomplete at this point, our efforts have convinced us that natural language processing can be done on microcomputers. Developmental work in this area need not be restricted to the enormously expensive mainframe system.

References

1. *AI Magazine*, American Association for Artificial Intelligence, 5147 Angeles Crest Highway, LaCanada, California 91011.
2. *Introduction to Artificial Intelligence,* by Phillip C. Jackson, Mason/Charter Publishers, 1974.
3. "Intelligent Machinery," by A.M. Turing (1947); reprinted in *Machine Intelligence 5,* edited by B. Meltzer and D. Michie, 1970.
4. "Programming a Digital Computer for Playing Chess," by C.E. Shannon, *Philosophy Magazine,* 41 (1950), pp. 356-375.
5. "GPS, A Program That Simulates Human Thought," in *Computers and Thought,* edited by Feigenbaum and Feldman, McGraw-Hill, 1963.
6. *Pattern Classification and Scene Analysis,* by R.O. Duda and P.E. Hart, Wiley, 1973.
7. *Computers and Thought,* edited by Feigenbaum and Feldman, McGraw-Hill, 1963.
8. *Understanding Natural Language,* by Terry Winograd, Academic Press, 1972.
9. *Psychology and Language: An Introduction to Psycholinguistics* by Clark and Clark, Harcourt, Brace and Jovanovich, 1977.
10. "Transition Network Grammars for Natural Language Analysis," *Communications* of the ACM, 1970, 13, pp. 591-606.
11. "The Use of Microcomputers in Artificial Intelligence Research," (Part One), MICRO, Feb. 1981, pp. 82-85.

6502 Op-Codes

Sooner or later, nearly every serious microcomputer experimenter begins to experiment with machine code—the fundamental building block of any microcomputer program. To aid in the understanding of the 6502's machine code we are presenting the following op-code structure chart. A brief study of the chart immediately reveals that this is not the typical op-code chart! This chart is set up as the 6502 evaluates its op-codes, a sort of "reverse" octal.

Rather than a laborious explanation of the chart, let's look at an example. Consider the op-code (hex) A9. Most people familiar with the 6502 will recognize this as the op-code for Load Accumulator, Immediate. When the 6502 "sees" this op-code, it sees the binary equivalent of A9—10101001. By regrouping the binary symbols, we may easily place this op-code in the structure chart.

Bit#	765	432	10
Code	101	010	01

Referring to the chart, we find that bits 1 and 0 define the op-code type, in this case Type 01. Bits 7, 6, 5, and bits 4, 3, 2 define the operation and the addressing modes, respectively. So, (hex) A9 equates to Type 01, Operation 5 (LDA) and Addressing Mode 2 (Immediate).

There are obviously several exceptions to the regularity of the chart. These op-codes are shown individually wherever they occur. An interesting note is the apparent absence of Type 3 (11) op-codes. Actually, there are several Type 3 op-codes that behave as a combination of Type 1 and Type 2 op-codes. An example of this is the binary "op-code" 101 001 11. This is a Load X *and* Load A operation in the Zero Page addressing mode.

Although it is interesting to experiment with these Type 3 op-codes, they should never be used in actual programs. They are "undocumentable" and may not exist in future revisions of the 6502 mask.

OHIO SCIENTIFIC'S

6502 OP-CODE STRUCTURE

7	6	5	4	3	2	1	0
OP			ADR MODE			TYPE	

TYPE 01

MODE / OP	(–,X) 0	ZP 1	IMM 2	ABS 3	(–),Y 4	ZP,X 5	ABS,Y 6	ABS,X 7
ORA 0	X	X	X	X	X	X	X	X
AND 1	X	X	X	X	X	X	X	X
EOR 2	X	X	X	X	X	X	X	X
ADC 3	X	X	X	X	X	X	X	X
STA 4	X	X	-	X	X	X	X	X
LDA 5	X	X	X	X	X	X	X	X
CMP 6	X	X	X	X	X	X	X	X
SBC 7	X	X	X	X	X	X	X	X

TYPE 10

MODE / OP	(–,X) 0	ZP 1	ACC 2	ABS 3	(–),Y 4	ZP,X 5	ABS,Y 6	ABS,X 7
ASL 0	-	X	X	X	-	X	-	X
ROL 1	-	X	X	X	-	X	-	X
LSR 2	-	X	X	X	-	X	-	X
ROR 3	-	X	X	X	-	X	-	X
STX 4	-	X	TXA	X	-	ZP,Y	TXS	-
LDX 5	IMM	X	TAX	X	-	ZP,Y	TSX	ABS,Y
DEC 6	-	X	DEX	X	-	X	-	X
INC 7	-	X	NOP	X	-	X	-	X

TYPE 00

MODE / OP	0	ZP 1	2	ABS 3	4	ZP,X 5	6	ABS,X 7
0	BRK	-	PHP	-	BPL	-	CLC	-
1	JSR	BIT	PLP	BIT	BMI	-	SEC	-
2	RTI	-	PHA	JMP	BVC	-	CLI	-
3	RTS	-	PLA	(JMP)	BVS	-	SEI	-
4	-	STY	DEY	STY	BCC	STY	TYA	-
5	LDY IMM	LDY	TAY	LDY	BCS	LDY	CLV	LDY
6	CPY IMM	CPY	INY	CPY	BNE	-	CLD	-
7	CPX IMM	CPX	INX	CPX	BEQ	-	SED	-

X = op-code valid
– = op-code not valid

CALL 1-800-321-6850 TOLL FREE

65D BASIC Trace

In this article we describe a routine that can be used for debugging and tracing a BASIC program written under the OS-65D operating system. This routine, when enabled, will monitor LET statements in a BASIC program. That is, when any statement of the form

LET Y = (formula)

or simply

Y = (formula)

is executed, the line number of the statement will be printed, followed by the value of the formula. Hence the programmer can follow the progress of a running program and obtain lots of output for debugging purposes. Only floating point variables will be monitored.

Lines 380-430 of the assembler program are only to give the output a nice appearance. Deleting these lines will not affect the function of the routine, although the output will be a little harder to read. Lines 190-230 are necessary to avoid a syntax error which would result if this code were used in a program with a FOR statement. When a FOR statement is executed, the code for LET is called, and the subroutines called here destroy some pointers used by FOR.

We have written the assembler program to be assembled to addresses $5000 through $5039. If the machine code is put at another place in memory, the only changes necessary are in the address portions of the two "JSR RESTOR" lines.

If each of lines 460 and 480 are replaced by lines 580 through 620, then the machine code can be placed anywhere in memory.

The following is an assembler listing and a sample BASIC program with the enabling and disabling POKEs.

```
 10 5000               *=$5000
 20                   ;
 30                   ; external addresses
 40                   ;
 50 00AE=                 FAC = $AE     ;the floating accumulator
 60                   ;                  used by BASIC
 70 00F0=              TMPFAC = $F0     ;a place for us to save the FAC
 80 00C7=              TXTPTR = $C7     ;this page zero address
 90                   ;                  contains a pointer into
100                   ;                  the line currently being
110                   ;                  executed
120 2343=               OUTCH = $2343   ;DOS routine to output the
130                   ;                  character currently in the
140                   ;                  accumulator
150                   ;
160                   ;first see if we are in the middle of a LET statemen
170                   ;if so, exit immediately
180                   ;
190 5000 A000                LDY #0
200 5002 B1C7                LDA (TXTPTR),Y
210 5004 C99D                CMP #157         is it the token for TO
220                   ;                       in a LET statement?
230 5006 F024                BEQ RETURN
240                   ;
250                   ;
260                   ;save the FAC, the subroutine called next will
270                   ;destroy it
280 5008 A205        SAVE    LDX #5
290 500A B5AE        LOOPSA  LDA FAC,X
300 500C 95F0                STA TMPFAC,X
310 500E CA                  DEX
320 500F 10F9                BPL LOOPSA
330                   ;
340 5011 20D81C PRTLIN  JSR $1CD8        call the routine to print
350                   ;                       the line number
360                   ;
370                   ;print a few characters
380 5014 A920                LDA #'
390 5016 204323              JSR OUTCH
400 5019 A93A                LDA #':
410 501B 204323              JSR OUTCH
420 501E A93D                LDA #'=
430 5020 204323              JSR OUTCH
440                   ;
450                   ;
460 5023 202F50              JSR RESTOR       bring back the FAC
470 5026 20510A              JSR $0A51        print the FAC
480 5029 202F50              JSR RESTOR       lost it again
490 502C 4CCB1A RETURN  JMP $1ACB        jump to the routine which
500                   ;                       stores the FAC. (This is
510                   ;                       line that was replaced by
520                   ;                       a jump to  this code.)
530                   ;
540                   ;
550                   ;routine to restore the FAC after it is destroyed
560                   ;by some called subroutines
570                   ;
580 502F A205        RESTOR  LDX #5
590 5031 B5F0        LOOPRE  LDA TMPFAC,X
600 5033 95AE                STA FAC,X
610 5035 CA                  DEX
620 5036 10F9                BPL LOOPRE
630 5038 60                  RTS


OK
LIST

 100 PI=3.14159
 200 DISK!"CA 5000=06,3          :REM we saved the machine code here
 300 L=2520
 400 POKE L,0: POKE L+1,80       :REM the enabling POKEs. The values
 500 REM                              POKEd are the two hales of
 600 REM                              entry address of the tracing
 700 REM                              routine.
 800 FOR N=1 TO 8
 900 Y=N*N-9*N+19
 1000 IF Y>0 THEN X=SQR(Y)
 1100 NEXT N
 1200 POKE L,203: POKE L+1,26    :REM disabling POKEs.

OK


RUN
 900 := 11
 1000 := 3.31662479
 900 := 5
 1000 := 2.23606798
 900 := 1
 1000 := 1
 900 :=-1
 900 :=-1
 900 := 1
 1000 := 1
 900 := 5
 1000 := 2.23606798
 900 := 11
 1000 := 3.31662479

OK
```

Reset Protection for the Apple II

Avoid the frustration of accidentally hitting your Apple's RESET with this easy-to-install override switch.

Joe Brady
1004 Breezewick Rd.
Towson, Maryland 21204

If you're like I am, you probably occasionally hit reset in the middle of a program. While your own BASIC programs can usually be re-entered by CTRL-C or 3D0G, what do you do if you blow a game of chess, or reset while entering a large data base? Well, if this has happened to you, read on and see what I did to protect myself.

I made my switch for less than $6.00. The parts needed are: (see photo 1)

1. A single pole, double throw, center-off, one side momentary switch. (What all this means is that down is on, center is off, up is on but returns to center when released.)
2. A 16 pin header package.
3. A 16 pin wire wrap socket.
4. A length of 3 conductor wire.

The idea of the switch is to disable the keyboard reset when the protect switch is in the center (off) position, enable it when the protect switch is in the locked down (on) position, and perform a reset when the protect switch is placed in the up (momentary on) position and released. All this is easier than it sounds, (see diagram 1).

What we are building is an intercept plug to be placed between the keyboard cable and the motherboard connector (see photo 2).

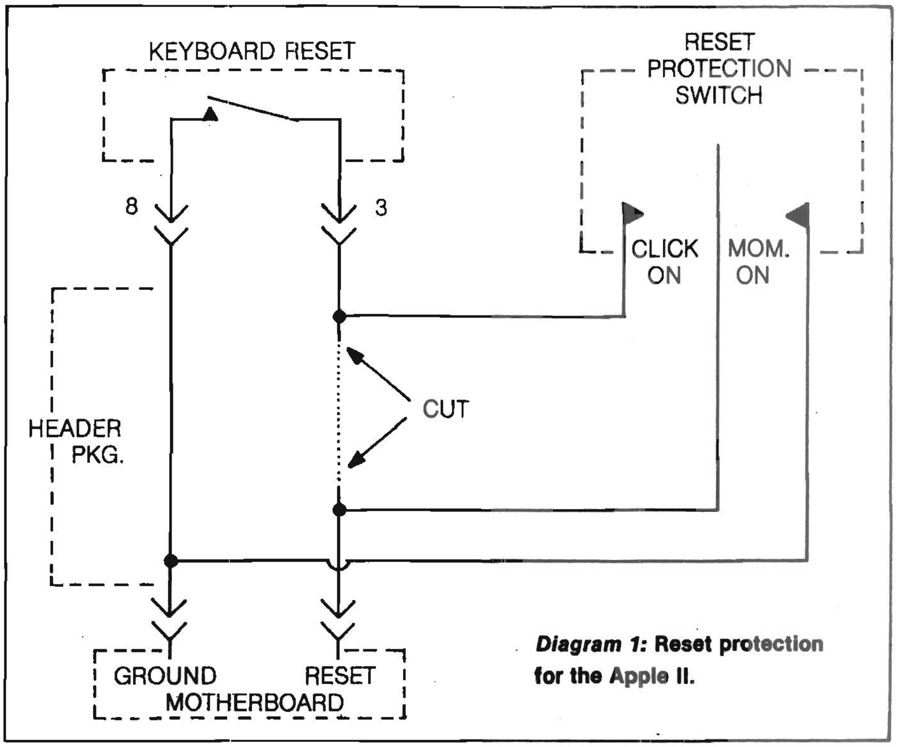
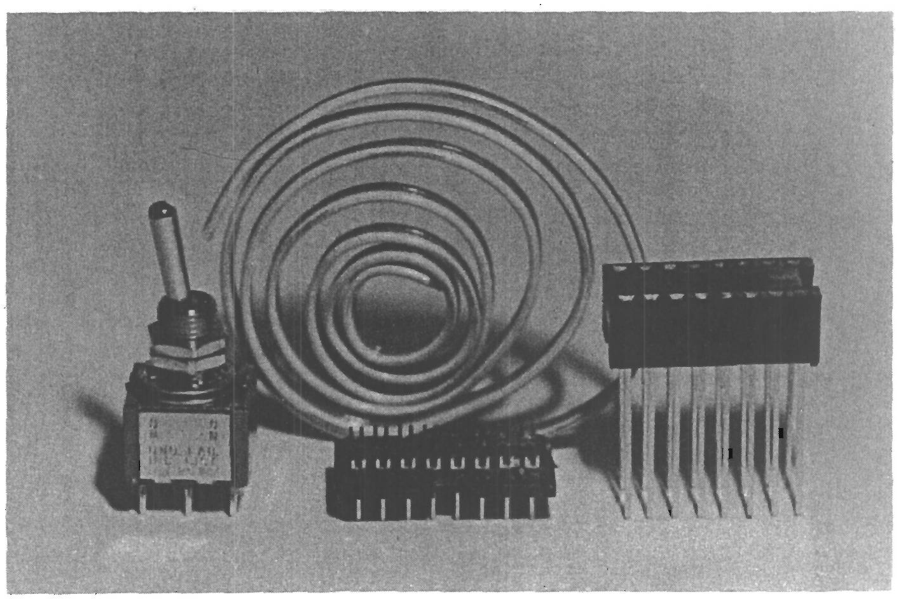


***Diagram 1:* Reset protection for the Apple II.**



To begin construction, the socket must be soldered to the header plug (photo 3). Due to the heat involved, it is recommended that the header plug be placed in another socket to act as a heat sink and keep the pins straight. After connecting the socket to the header, clip the leads connecting pins three, leaving enough lead to solder the multi-conductor wire. Connect two wires as shown in the photo, being careful not to form a solder bridge with adjacent pins. Connect the third wire to pin 8 (ground) This completes the plug end. Check for flaws and then construct the switch end.

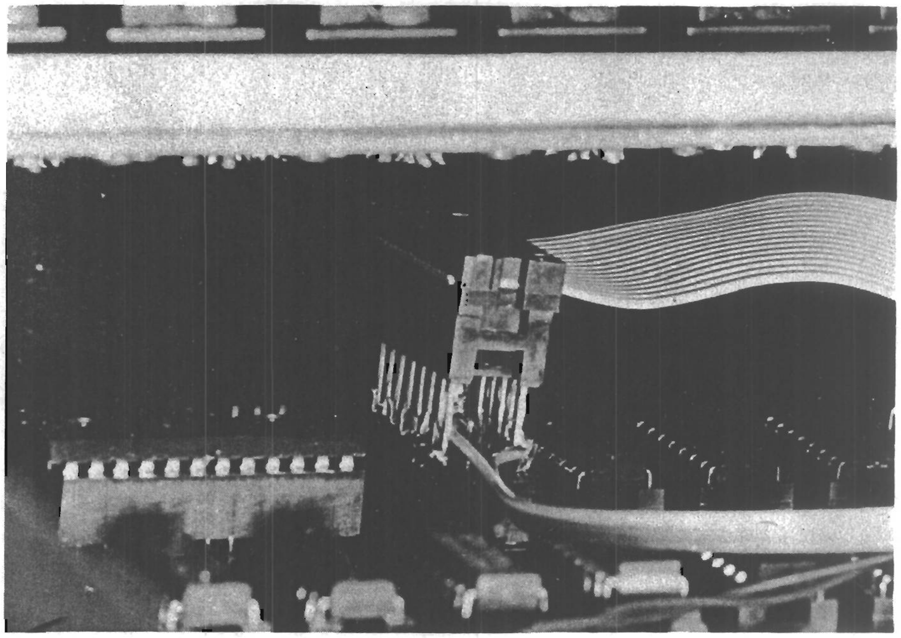


Photo 2

Leaving enough wire to reach from the motherboard connector to the back of the Apple, connect the wire from the motherboard side of pin 3 to the center connector of the switch. Connect the keyboard side of pin 3 to the 'click on' side of the switch and the wire from pin 8 (ground) to the 'momentary on' side of the switch.

NOTE, the switch connectors are in the opposite direction from the switch movement.

The switch is mounted in the rightmost cable slot in the Apple with no drilling required (photo 4). The momentary side of the switch faces up. With power off, mount the header/socket package between the cable and motherboard socket (photo 2).

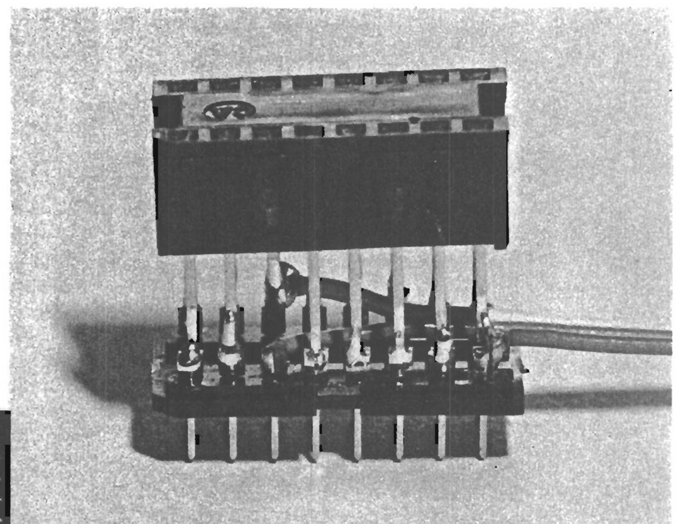


Photo 3

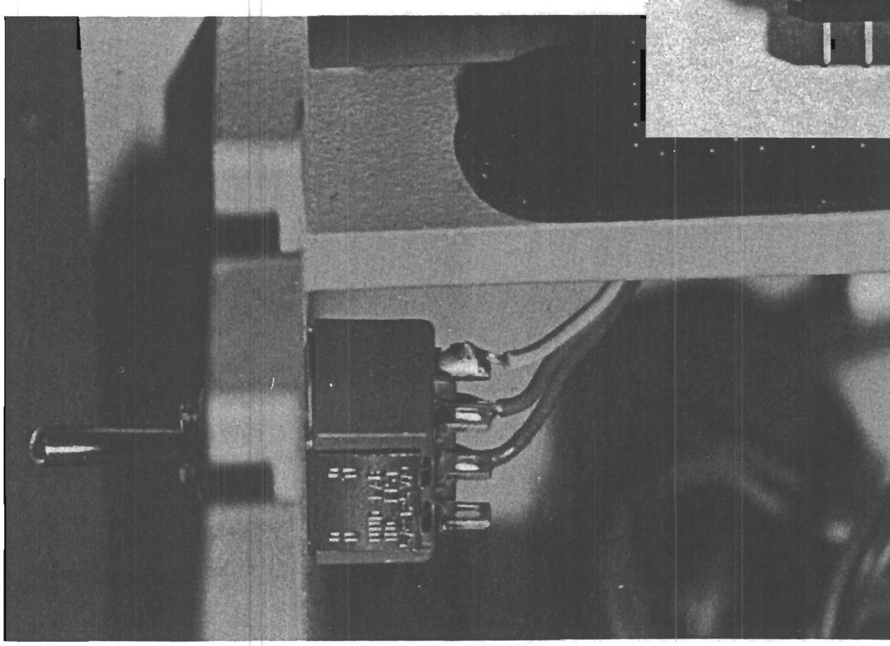


Photo 4

To use the keyboard reset normally, place the switch in the down position. To protect from accidental reset, place the switch in the center position and reset by pulling the switch up and releasing.

The nice thing about this addition is that it requires no modification of the Apple itself and can be removed at anytime.

MICRO

MICRO

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

Software Catalog: XXX

Name: **Lunar Zoo Keeper and Herder**
System: Apple II or ITT 2020
Memory: 48K
Language: Applesoft
Hardware: Apple II, Disk preferable
Description: Two excellent real time Hi-Res games: Lunar Zoo Keeper, catch the lunar creatures as you fly over the moon then drop them in the cage on earth: Herder—for 2 players—can you herd the bulls into your corral before your opponent? Will the bull gore you?
Copies: Just released
Price: Disk $18, listings $14
Author: **Kieron Leech**
Available: **Kieron Leech**
15 Chatsworth Ave.
Culcheth, Warrington, Cheshire, WA3 4LD, England, G.B.

Name: **6502 Assembler Package**
System: PET/CBM
Memory: 8K
Language: BASIC, Machine
Hardware: One tape or disk drive
Description: HESBAL is a full-featured Assembler that leaves over 1200 bytes free (8K) for your use. Several pseudo-opcodes and over 25 error messages. HESEDIT is a full-screen editor used to prepare input to HESBAL or maintain files like mailing lists, etc. All keys repeat and you can insert, delete and even duplicate lines.
Copies: 50
Price: Cassette—$23.95
Diskette—$26.95
Postage—$ 1.50
CA res. 6% sales tax
Includes 70 pages of documentation
Author: **Jay Balakrishnan**
Available: **Human Engineered Software**
3748 Inglewood Blvd., Room 11
Los Angeles, CA 90066
(213) 398-7259

Name: **Monster Combat**
System: SYM with BAS-1 or KIM 8K BASIC at 2000 H.
Memory: 8K
Language: BASIC
Hardware: Terminal using standard serial I/O ports on SYM or KIM
Description: An adventure game in which a journey is taken through a large forest. Various monsters are encountered and must be fought in order to obtain the treasure they guard. Object is to win as much treasure as possible and then get out of the forest alive with the treasure. Some "random happenings" are used to keep the game interesting. Please specify SYM or KIM version.
Copies: Just released
Price: $10.00 on cassette tape, ppd. in US only
Author: **Lee Chapel**
Available: **Lee Associates**
2349 Wiggins Ave.
Springfield, IL 62704

Name: **Super Decimals**
System: Apple II
Memory: 16K
Language: Applesoft
Hardware: Disk, printer option
Description: A serious CAI program for Teachers G1-9. Will do Whole Numbers or Decimals; You select size of the numbers, not LEVELS. Grades work, has traps, & tested in-house for 2 years at a Middle School. Two other programs on disk.
Copies: Just released
Price: $10
Author: **Pat Calabrese**
Available: **BIT'N PIECES SERIES**
P.O. Box 7035
Erie, PA 16510

Name: **Graph*Fit**
System: Apple II Plus or A-Soft ROM 48K
Hardware: Single Disk, Not available on tape.
Description: A Hi-Res graphing program that will produce four types of colorful Hi-Res graphs. 1-Bar Charts, 2-Pie Charts, 3-line graph with numeric x axis, 4-line graphs with monthly x axis. All entries made by the user may be easily changed and an automatic scale feature makes the software easy to use. All graphs may be saved on disk and the software is copyable, allowing the user to make backups.
Price: $25.00 Dealer inquiries invited
Author: **Phil Koopman, Jr.**
Available: **Micro-Ware Dist., Inc.**
439A Route 23
P.O. Box 113
Pompton Plains, NJ 07444

Name: **Planet Positions**
System: Apple II or II+
Memory: 32K
Language: ROM Applesoft or Language card. Some machine code. RAM Applesoft version on special order.
Description: This program plots the orbits of the six inner planets of the Solar System using High Resolution Graphics. Plots can be generated starting at any date between the years: 1980-1999. The distance from the earth is displayed plus the 'Right Ascension' values for locating the planets in the sky. Errors have been verified at less than 1.5%. Also included is a program that determines local time of 'Celestial Meridian' for a given Right Ascension, date & observer longitude. These programs are particularly useful in helping teach the motion of planets. An elementary knowledge of Astronomy is helpful in gaining maximum benefit of these programs.
Price: $21.95 includes DOS #3.3 Diskette, example, plus description.
Author: **Neil A. Robin**
Available: **TECH-DIGIT CO.**
21 Canter Lane
Sherwood, OR 97140

Name: **DQ Secretary**
System: ALL OSI
Memory: 24K (32K for 8" disk)
Language: Machine
Hardware: 5" or 8" disk
Description: An OS65D enhancement which replaces OSI's utilities for managing named files. Allows files to be renamed, created, and deleted without disturbing programs in the workspace. Files are created dynamically when programs or text are saved. Disk is repacked when necessary to free up extra space. Works with OS65D files and WP6502 word processing files.

Copies: Just released
Price: $50.00
Author: **Mike Cohen**
Available: **Dwo Quong Fok Lok Sow**
23 East 20th St.
New York, NY 10003

Name: **TREND-SPOTTER**
System: Apple II+
Memory: 48K
Language: Applesoft
Hardware: Apple II+, Disk Drive. (Second drive and printer optional)
Description: TREND-SPOTTER is an easy-to-use business graphics and analysis system. A friendly user interface, tailored to the needs of today's executive, provides the user with the ability to spot emerging trends through quick manipulation and graphing of business data. Generates color graphic displays, calculates and displays trend lines, performs mathematical and statistical computations, prints graphic and tabular data, and edits and updates data files. TREND-SPOTTER will both generate and read VisiCalc-compatible files.
Price: $275.00 includes manual and disk.
Available: **Software Resources, Inc.**
44 Brattle Street
Cambridge, MA 02138
(617) 491-6396

Name: **Apple Writer Mod for Control Codes & Lower Case**
System: Apple II or II Plus with 1 Disk
Memory: 48K
Description: A modification to the Apple Writer Word Processing program from Apple that allows you to enter control codes into the text, thereby utilizing the special print features of most new printers. Can be used to go from 10CPI to 16.7CPI, or from Normal to Expanded, etc. Requires Paymar LCA or similar. Mod must be made to an Apple Writer Original Disk, yours for $35, or a new Apple Writer for $100 with Mod. LCA available also for $60.
Available: **Turnkey Mini-Computers**
7372 NW 5th Street
Plantation, FL 33317
(305) 791-4578

Name: **Musical Computer One and Two**
System: Apple II, TRS-80 Level II and ATARI
Memory: 32K, 16K, & 32K, respectively.
Language: Integer BASIC
Description: A two-program cassette tape which explains the fundamentals of music — including musical symbols and language, note reading on both the trebel and bass clefs, telling time, note values and rests, piano keyboard, dynamic and tempo markings, signs and symbols and enough PRACTICE and TESTING opportunities for both the beginning and advanced student. Written by a M.A. educator with over 20 years of music experience. This is truly an alternative to music education, accompanied with colorful musical descriptions and musical sounds.
Copies: Many
Price: $34.95, $1.00 p&h (MI residents add 4 % sales tax.)
Author: **Myra Marshall**
Available: **Computer Applications Tomorrow**
P.O. Box 605
Birmingham, MI 48012

Name: **Super Draw and Write**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II, Disk II
Description: Contains many utility programs and two major programs allowing you to draw and type with ease. SUPERFONT allows you to type in 9 different scales (sizes) and with 8 different styles (72 combinations). You can save and retrieve, too! With Instant Graphics (Sound Option) you can draw anything: circles, ellipses, parts of figures, filled or unfilled, any color. Floating dot, cursor included. We have the only program available that arrays-saves drawings in 2-5 sector text files *and* "Hi-Res page one", saves in the usual 34 sector files. See it to believe it. Why pay 2-3 times this price and get less?
Copies: Many
Price: $15.95
Includes: Disk, drawing card
Available: **Avant-Garde Creations**
P.O. Box 30161MCC
Eugene, OR 97403

Name: **STOCKFILE**
System: Apple II or Apple II+ or Language System
Memory: 48K
Language: Applesoft/machine code
Hardware: Apple, two to four disks, printer
Description: Holds up to 9500 different parts, with maximum two second access. Disks can be added as needed. Deletes, adds, or changes will not slow up the system. Comes with a set of standard reports, but the user can easily design reports (NO programming!) to specify fields and their positions, any sorted order, page breaks, titles, how many "up", and complete choice of which parts are to be printed. Part numbers can be up to 20 characters long, any characters allowable. Fully indexed manual, plus a Tutorial Guide, and sample data. The user can alter part number length etc, and the system will convert the database. Has a Block Data entry facility to quickly add starting data. Many users, well field tested.
Price: Determined by Dealer
Author: **Softech International Corporation**
Available: **Softech International**
#6, 144 W. 15th St.
North Vancouver, B.C.
Canada V7M 1R5
(604) 984-0477

Name: **Astronomy Package**
System: ATARI or PET
Memory: 8K
Language: BASIC
Hardware: no extra
Description: After playing your thousanth game of Space Invaders, try some real astronomy software. With HORIZON and a star atlas, you can predict exactly where to point your camera or telescope for any celestial object, at any time, rising, setting, or high in the sky. When it's cloudy, STAR ENCOUNTERS can give you a provocative look at our dynamic universe by mathematically following the movement of the stars relative to the Earth. Will we ever collide?
Price: $6.00 ($8.00 ATARI) SASE for catalog.
Author: **Russell A. Grokett, Jr.**
Available: **KINETIC DESIGNS**
401 Monument Rd. #171
Jacksonville, FL 32211

Name: **ELF — Ecometrics, Linear Models and Forecasting System**
System: Apple II Plus
Memory: 48K
Language: Applesoft and 6502 Assembler
Hardware: 1 Disk
Description: Statistical and econometric programs; stepwise regression, factor analysis, correlation, discriminant analysis, univariate statistics, t-tests on means, cross tabs with Chi Square. Handles all Applesoft transformations. Can 'select if'. Standard statistics reported for each technique. Prompts user for answers. Data bank.
Copies: New release
Price: $150 includes program, manual (40+ pages), 1 year maintenance and updates
Author: **Eric Weiss, Ph.D.**
Available: **The Winchendon Group**
P.O. Box 10114
Alexandria, Virginia
22310

Name: **Accounts Receivable**
System: Apple II
Memory: 48K
Language: Applesoft or Language System
Hardware: Dual 5" drives, any 130 column printer
Description: A quality program, structured around the Osborne Accounts Receivable software, with several added enhancements. Can be used alone or integrated with existing General Ledger program. Features open invoicing, credit and debit memos, full or partial payments, progress billing, invoice aging, and printing of statements. System is available on DOS 3.2, DOS 3.3, or 8" drives, if additional capacities are required. A flexible, easy to use program with an exceptional combination of speed and high performance.
Price: $180.00 each package
Author: **David A. McFarling**
Available: **Small Business Computer Systems**
4140 Greenwood
Lincoln, Nebraska
68504

Name: **pns-FORTH**
System: Atari 400/800
Memory: 16K minimum
Language: Forth Interest Group Forth, with extensions
Hardware: 1 Disk, minimum
Description: Forth is an operating system, an interpreter and a compiler. You can define new command words to make full use of the hardware resources of the Atari. A full screen editor is provided; source code is edited from disk. Provision has been made for the implementation of floating point math. Future versions will include music and graphics editors.
Price: $50.00, ver. 1.2 on diskette — includes documentation and customization guide
Author: **Bob Gonsalves**
Available: **Pink Noise Studios**
1411 Center St.
Oakland, California
94607
(415) 465-1212

Name: **A-la. Stock Market Analysis**
System: PET
Memory: 8K
Language: BASIC
Hardware: PET/CBM
Description: Analyzes a time series, such as a stock price or market average giving equations for the long-term trend, major cycle and next shorter cycle. Prints individual curves or composite and/or tables projected to future times.
Price: $15.00 for cassette and documentation
Author: **Claud E. Cleeton**
Available: **Claud E. Cleeton**
122-109th Ave., S.E.
Bellevue, Washington
98004

Name: **DQ MAIL — I**
System: All OSI
Memory: 32K
Language: BASIC, Machine
Hardware: 8" Disk
Description: A system which selects data from DMS (OSI Data Base Management System) files and formats them for access by the WP6502 word processor. Any DMS file can be accessed such as Mailing List, General Ledger, Accounts Payable/Receivable, etc. The user may also specify up to four items of record selection criteria (amount due greater than $1000, etc.).
Copies: 100+
Price: $50
Author: **Hal Pollenz**
Available: **Dwo Quong Fok Lok Sow**
23 E. 20th St.
New York, New York
10003

Name: **STAT TUTOR**
System: Apple II Plus
Memory: 16K
Language: BASIC
Hardware: Applesoft in ROM
Description: Disk based (16 or 13 sector). Quiz mode tutorial - descriptive statistics - choose from mean - median -variance - standard deviation problems randomly generated, answers given and hints available—work is tabulated. Tape version also available (needs 4K plus).
Copies: New release
Price: $18.00
Author: **Bill Stanton**
Available: **STANTECH**
Box 19123
Cincinnati, Ohio
45219

Name: **Mighty Mite**
System: Apple II/Apple II+
Memory: 32+K
Language: Applesoft
Hardware: Disk drive, printer (optional)
Description: An affordable word processor simply commanded by control characters. You can insert lines and spaces, edit, command auto words, delineate errors plus access to any one of the 20 CRT pages with a single control character. It comes fully documented. Now you will affordably love letter writing, memo notation, file information and listings. And, of course, you can save and edit it for later use as well as run multiple copies.
Copies: As needed
Price: $24.95 shipped USA (specify 3.2 or 3.3)
Author: **BDI**
Available: **R. Sherman, PCSE**
52 Jackson Drive So.
Poughkeepsie,
New York 12603

MICRO

6502 Bibliography: Part XXX

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

861. Southeastern Software Newsletter, Issue 21 (September, 1980)

Staff, "DOS 3.3," pg. 1-4.
Interesting comments and experience on using the new DOS.

Staff, "Drawing a $BEAD on DOS," pg. 8-10.

862. Compute II, Issue 3 (August/September, 1980)

Zumchak, Gene, "Nuts and Volts No. 3: Address Decoding," pg. 15-16.
Address Decoding 6502 systems.

DeJong, Marvin L., "A Simple Interface For A Stepper Motor," pg. 18-19.
Hardware and listings to allow you to drive a stepper motor with your 6502-based micro.

Herman, Harvey B., "KIM-1 Tidbits," pg. 22-24.
Interrupt service routine for real-time clock.

MacKay, A.M., "SYM-1 Home Warning System," pg. 26-29.
Hardware and Program for SYM-1 to provide a home warning system.

DeJong, Marvin L., "A Digital Cardiotachometer Implemented With The AIM 65," pg. 32-35.
Hardware and listing.

Wells, George, "Saving Data Matrices with Your SYM-1," pg. 36-39.
A machine language program which enables BASIC data matrices to be saved on tape and loaded back into the SYM at a later time.

Stanford, Charles L., "Fast Graphics on the OSI C1P," pg. 42-46.
Machine language subroutine and tutorial for the C1P.

Clements, William C., Jr., "Modification and Relocation of FOCAL 65-E Into Erasible PROM," pg. 48-49.
Information relating to the use of the FOCAL 65-E with the KIM-1."

863. Creative Computing 6, No. 9 (September, 1980)

Douchant, Gary, "Waiting for Atari," pg. 10.
Graphing Polar Coordinates with the Atari 800.

Hansen, Chris, "Computer Countdown," pg. 98.
Experiences using the PET in working with young students.

Piele, Donald T., "How to Solve It—With the Computer," pg. 126-131.
A number of problems and computer solutions including a couple for Microsoft BASIC on the 6502 systems.

Blank, George, "Outpost: Atari," pg. 180-182.
Use of the Atari in education programs.

Yob, Gregory, "Personal Electronic Transactions," pg. 190-194.
The Stringy Floppy for the PET, computer assisted instruction, letter combinations for phone numbers, etc.

864. The Core (September, 1980)

Budge, Joe, "ONERR Messages," pg. 7-8.
Tutorial on ERR messages for the Apple II.

865. Apple Bits 2, No. 7 (September, 1980)

Kovalik, Dan, "Taking the Mystery and Magic out of Maxhine Language," pg. 7-10.
Passing parameters to subroutines in Apple machine language.

Koehler, John, "Basic BASICS," pg. 13.
How to improve the execution speed of your programs.

Anon., "IAC Application Note: Converting Integer BASIC Programs to Applesoft," pg. 15.
Procedure for conversion of Apple programs.

Anon., "Applesoft Random Numbers," pg. 15.
How to seed your Apple random number generator.

Anon., "IAC Application Note: Applesoft Array Eraser," pg. 18.
Routine for the Apple.

Anon., "IAC Application Note: Hi-Res SCRN Function Demo," pg. 18.
Demo to draw graphics in both Hi-Res and Lo-Res on the Apple.

866. MICRO No. 28 (September, 1980)

Cook, Peter A., "Creating Shape Tables, Improved!", pg. 7-12.
Ease the pain of making shape tables on the Apple.

Kolbe, Werner, "Auto-Run-Save, Y-t Plotter, Canary for the PET."
Potpourri of programs for the PET.

Gonzalez, Larry P., "Loading KIM-1 Tapes to AIM," pg. 19-22.
Routine for the AIM.

Bresson, Steve, "Compact," pg. 25-28.
A program for the AIM to strip REMarks from BASIC programs.

Taylor, William L., "A C1P and H14 System, Part 2," pg. 30-32.
Interfacing the OSI micro to a Heath printer.

Swank, Joel, "XREFER," pg. 34-39.
A cross-reference utility for 6502 systems permitting the output of an assembler to be sorted and cross-referenced.

Allen, David P., "A Versatile Hi-Res Function Plotter for the Apple II," pg. 49-54.
Plot various mathematical programs on the Apple. Also includes a version for the Atari as well.

Meinrath, Dr. Mark H., "MICROSCOPE—KIM-1 Venture," pg. 57.
A review of a tape program for the KIM-1. A fantasy game.

Gutekunst, Carl and Kollar, Larry, "Tiny Pilot for the AIM," pg. 59-65.
TINY PILOT is a compact programming language.

Mattola, R.M., "Mean 14: A Pseudo-Machine Floating Point Processor for the Apple II," pg. 67-71.
A floating point processor modelled after the Sweet 16 to provide five-byte floating point values.

Rowe, Mike (Staff), "The MICRO Software Catalog: XXIV."
Twelve software items for 6502 systems are reviewed.

Dial, Wm. R., "6502 Bibliography: Part XXIV," pg. 76-78.
Another 112 references to the 6502 literature.

867. Appleseed Newsletter 2, No. 3 (Aug./Sept., 1980)

Pump, Mark, "Apple II DOS Internals."
Information useful to Apple Disk owners.

Mikiten, Terry, "Memory Move Demonstration," pg. 3.
Shows how a block of memory can be moved using the Hi-Res graphics screen on the Apple.

Anon., "Cursor Changer," pg. 4.
A machine language routine for the Apple.

Hyde, Bill, "Update DOS 3.2 to 3.3 on Dan's Disk Utilities," pg. 4.
How to update this important utility for the Apple.

Hyde, Bill, "Language Card Loader," pg. 6.
Machine language routine for the Apple.

Wright, Don, "Auto Run Tapes and Shapes," pg. 9.
How to disable the Auto-Run flag.

Mikiten, Terry, "Magic Scroller," pg. 10-12.
A tutorial type routine in machine language for the Apple.

868. PEEK(65) 1, No. 9 (September, 1980)

Holt, Kenneth D., "Systems Programming Note on 65U Stepping Rate," pg. 10.
Improved technique for modifying the stepping rate on OSI systems.

Sanders, Jim, "Defeating the OSD-OSU Screen Position Location 22," pg. 11.
Examples with the manipulation of the cursor position register on OSI systems.

869. The Apple Shoppe 1, No. 7 (July/August, 1980)

Anon., "Pascal PEEK and POKE," pg. 7-8.
A tutorial for Pascal users on PEEK and POKE usage.

Jeske, Mark, "General Purpose Plotter Program," pg. 11-13.
A Hi-Res program for the Apple.

Anon, "Programming the Graphics Tablet," pg. 14-18.
A tutorial article and "Hi-Res Labeling Program."

Crouch, Bill, "Down to Business," pg. 18-20.
A better way to format numbers; some notes on developing an inventory program.

Anon., "Spinterm Hi-Res Dump in Assembly."
A Hi-Res Screen Dump program for the Apple.

Hugard, James, "Printing in Pascal," pg. 27-29.
A discussion of printing techniques in Pascal systems.

870. Call—A.P.P.L.E. 3, No. 7 (September, 1980)

Anon., "DOS 3.3 Bugs," pg.3.
How to fix a bug in writing to random access files under DOS 3.3 of the Apple Disk system.

McVay, Ray, "Game Sounds," pg. 7-11.
A simple pair of tone and noise generators to put spice into Integer BASIC games.

Reynolds, Lee, "Memory Saving Techniques in Apple II BASIC," pg. 15-16.
A tutorial for the Apple II.

Pump, Mark and Golding, Val J., "Automatic Free Space with Catalog," pg. 23.
A program to include in your Hello program on booting diskettes.

Walter, C. Edward, "Apple Silentype Fix," pg. 26.
A fix for the Silentype printer TAB problems.

Kersey, Bert, "Print the Unprintables," pg. 27-28.
How to print those characters which Integer BASIC will not accept.

Lingwood, David A., "Error Message Printing for 'ONERR GOTO'," pg. 29.
How to use this useful function on the Apple.

Curtis, Roger, "File Burp," pg. 30-33.
A very useful utility for nosing around in PASCAL disk files.

Huelsdonk, Bob, "Making BASIC Behave! Part V."
Continuing with a discussion of file techniques.

Widnall, Sheila, "Lower Case for Apple Writer Using the Paymar Chip," pg. 41-43.
Some modifications for the Apple Writer.

871. Creative Computing 6, No. 9 (September, 1980)

Douchant, Gary, "Graphing Polar Functions," pg. 10.
A plotting program for the Atari.

Hansen, Chris, "Computer Countdown," pg. 98.
Using PETs in the Middle school.

Stone, Deborah, "Computers at an Alternative School," pg. 46-47.
Use of the Apple in a private tutorial school.

Lubar, David, "A Buyer's Guide to Apple II Software," pg. 54-55.
Several games and utilities are reviewed.

Schwartz, Marc D., "Integrating CAI and Videotape," pg. 116-117.
An Apple II, a video player and a TV monitor work together in CAI.

Piele, Donald T., "How to Solve It—With the Computer," pg. 126-131.
Classroom usage of the Apple in problem solving.

Carlson, Ronald, "Complements and Supplements," pg. 140-142.
Use of the computer to solve geometry problems involving angles.

Carpenter, Chuck, "Apple-Cart," pg. 174-178.
Music programs, Disk-O-Tape, Applesoft Revealed, Clearing Memory Variables, etc.

Blank, George, "Outpost: ATARI," pg. 180-182.
Atari instruction programs, Real time Clock, Buying Memory, etc.

Yob, Gregory, "Personal Electronic Transactions," pg. 190-194.
The stringy floppy for the PET, anti-glare screen for the PET, Microphys software, a phone number program, etc.

872. Southeastern Software Newsletter Issue 22 (October, 1980)

McClelland, George, "Andromeda 16K Expansion Board," pg. 1-3.
Experiences using this new board in the Apple II.

Staff, "What is an Assembler?", pg. 4-8.
Introduction to the use of an assembler on the Apple.

873. The ABACUS II 2, Issue 9 (September, 1980)

Robbins, Greg, "Applesoft Disk Dump," pg. 3-5.
A Disk Dump to assist you in viewing the contents of an Apple diskette.

Robbins, Greg, "Applesoft HEX-DEC Converter," pg. 6.
A routine for the Apple.

Davis, James P., "Sound Effects Programs," pg. 7-8.
Two sound effects for the Apple.

Yee, David R., "Taming the Reset Key on your Apple II Plus," pg. 9-10.
Comments on the RESET function.

Anon., "Color Generation in Hi-Res," pg. 12-17.
The third and final article on Hi-Res Color for the Apple.

874. Nibble No. 5 (September, 1980)

Connolly, Rick, "PIP—The Personal Inventory Program," pg. 7-19.
The first of two articles on this Apple utility.

Severa, Tony, "Data Base Management for Beginners," pg. 23-25.
Description of an Apple-based Data Base program.

Harvey, Mike, "Numeric Key Pad Lab!", pg. 28-31.
Hardware and Software for building this accessory for the Game I/O connector of the Apple.

Kelley, Derek A., "I Am Buddha," pg. 33-35.
An Eliza-like program for the Apple.

Crossman, Craig, "Fun With Apple's Assembler," pg. 36-37.
How to use the Apple Assembler—a tutorial.

Mottola, R.M., "Passing Variables in Applesoft BASIC," pg. 39-40.
A tutorial for the Apple on variables.

Reynolds, William III, "Managing and Moving Disk Buffers," pg. 40-41.
Useful information on using the MAXFILES command.

Reynolds, William III, "Monitor Execution, BASICally," pg. 41.
How to enter Monitor command during the execution of the BASIC program itself.

Crossman, Craig, "Apple Tricks," pg. 43.
A real nifty trick for tricky programmers!"

Haehn, Lou, "Taking the Load off Your Mind," pg. 45-46, 59-63.
How to have the Apple provide an audio monitoring of a Tape Load.

Laird, Alexander, "Applesoft vs. Integer BASIC," pg. 46.
Speed comparisons between these two BASICs on the Apple.

Laird, Alexander, "Apple Tape Data File Control," pg. 46-48.
"Two-Tape Write/Read," a program on how to Save and Recall alphabetic and Numeric data with tape cassettes.

875. The Seed 2, No. 9 (September, 1980)

Won, Ted, "Copying the Menu Program," pg. 6.
How to change the top line of the program menu on Apple diskettes.

Isenberg, Ed., "Error proofing," pg. 7.
How to avoid errors in your program, for the Apple.

Anon, "Apple Monitor," pg. 9-10.
All about Apple windows, scrolling partial screens, etc.

Baxter, Bruce E., "Screen Write/File Routine," pg. 11.
This routine makes it simple to edit the Apple Screen and save the screen image on disk.

Hyde, Randy, "The Apple II Monitor," pg. 14-16.
Part Three of a tutorial. This part discusses zero page locations used by the Apple monitor.

Anon., "Screen Editing on the Apple II," pg. 15-16.
Helpful hints in screen editing.

Friar, Mason, "Integer ONERR GOTO," pg. 17-20.
Recovering from errors on the Apple.

Reynolds, Lee, "EXEC Files on the Apple II," pg. 21-22.
A tutorial on the EXEC function.

876. Dr. Dobb's Journal 5, Iss. 8, No. 48 (September, 1980)

Gordon, H.T., "Thoughts on Small Systems and Monitors," pg. 16-21.
Mostly on the SYM-1.

Maurer, W.D., "A Note on 6502 Indirect Addressing," pg. 26-31.
A tutorial on this important addressing function of the 6502.

877. L.A.U.G.H.S. 2, No. 4 (September, 1980)

Anon., "Understanding D.O.S.," pg. 3-5.
Discussion of DOS 3.1 3.2, 3.2.1, and the new 3.3, for the Apple Disk system.

Roe, David H., "16 vs. 13," pg. 6-7.
How the new 3.3 DOS gets 16 rather than 13 sectors into each track of an Apple diskette.

878. The Harvest 2, No. 1 (September, 1980)

Sherman, Dave, "File Access," pg. 1-3.
A tutorial for the Apple.

Anon., "Machine Language Subroutines in BASIC Programs," pg. 7-8.
An explanation of those seemingly endless strings of HIMEM:'s in listings of some BASIC programs.

Holle, Dav., "First DOS 3.3 Bug Found!!", pg. 8.
A bug and an antibiotic, for the Apple DOS 3.3.

879. From The Core (October, 1980)

Boyarsky, Bill, "Dem Ol' Muffin Blues," pg. 4.
Several bugs for 3.3 listings and fixes.

Huffman, David, "Examining the Diskette Directory from a HEX-ASCII Dump."
A tutorial for the Apple.

Sethre, Tom, "DOS 3.3," pg. 9-10.
A good description of the new Apple DOS.

880. KB Microcomputing, No. 45 (September, 1980)

Baker, Robert W., "PET-Pourri," pg. 10-12.
Disk programming tips for the PET, BASIC Program Symbol List, etc.

Baker, Robert W., "Write Self-Modifying PET Programs," pg. 30-31.
Easy way to store small amounts of data, for the PET.

Strand, David M., "Memory Expansion Cnadidates," pg. 32.
Simple chip replacement is all it takes to add 16K to some PETs.

Cordelli, Gary, "PET Machine-Language Masquerade," pg. 34-35.
Programming trickery lets machine language load and run from BASIC.

Strasma, James, "Add a Reset Button to any PET," pg. 36-37.
Corral those runaway routines and still preserve programs in memory.

Conover, Karen V., "The Phantom Tape Drive," pg. 40-42.
File handling on the PET.

Yob, Gregory, "Get Your PET on the IEEE 488 Bus," pg. 44-54.
Third part of a continuing article on the PET/IEEE 488 combination.

Pytlik, William F., "PET I/O Port Expander," pg. 56-57.
Music with the PET.

Ormvedt, Neil J., "The PET/CMH/H14 Connection," pg. 60.
Use of the Heath H14 printer with the PET.

Re', Ugo V., "Improving the OSI Challenger C2," pg. 124-131.
Part 2 of this article continues with video, cassette and keyboard modifications.

Martellaro, John, "Apple II Hi-Res Graphics Memory Mapping," pg. 134-136.
Tutorial on the Apple Hi-Res graphics.

Mazur, Jeffery G., "Romplus Plus."
A review.

881. Personal Computing 4, No. 9 (September, 1980)

Lubar, David, "Will It Fit?", pg. 47-48.
Estimating program size on the Apple.

Schnell, Pet, "Gas Prices," pg. 54.
What you lose each time the price of gasoline goes up—for the Apple.

882. The Apple-Dillo (September, 1980)

Huffman, David, "Examining the Diskette Directory from a HEX-ASCII Dump," pg. 4-6.
A tutorial for the Apple.

Sethre, Tom, "DOS 3.3," pg. 7-8.
A description of the new DOS 3.3 for the Apple.

883. Softalk 1 (September, 1980)

Depew, William, "Dealing With DOS from Assembly Language," pg. 10-11, 25-26.
A discussion of DOS techniques for the Apple.

883-a. Byte 5, No. 9 (September, 1980)

Wexler, Steven, "Penny Pincher's Joystick Interface," pg. 86-90.
A cheap joystick for the KIM-1.

Hooper, Philip K., "Making 6502 Indirect Subroutine Calls Efficient," pg. 98.
Comparison to three type calls on the Apple.

Bochardt, Ottmar E., "Relocating Assemblers and Linking Loaders," pg. 194-202.
6502 techniques useful in programming.

Campbell, Gordon, "6502 Loop Control," pg. 322-333.
Programming tips for 6502 users.

Sokol, Daniel D., "Notes on Absolute Location Interfaces to Apple Pascal."
Two listings for Apple Pascal users.

884. Recreational Computing 9, Issue 2 (September/October, 1980)

Lindsay, Len, "Word Search—A Hunt for Hidden Words," pg. 28-33.
Find the words in those matrices of letters—for PET and CBM Computers.

Zimmermann, Mark, "Textrapolation," pg. 48.
A PET program which extrapolates on text already typed into it by finding the best match with words already entered.

885. SoftSide 2, No. 12 (September, 1980)

Immings, Bart, "Saving Talkertables," pg. 7.
How to save Talkertables for Bob Bishop's Appletalker.

Truckenbrod, Joan, "Computer Aided Drawing and Design," pg. 15, 83.
These techniques for the Apple can also be used with the Atari.

Anon., "The Stereo Generator," pg. 18-19, 82.
Play Stereo with the Apple.

Johnson, Paul, "Super Barricade," pg. 24.
A game for the Atari.

Johnson, Douglas, "Foosball," pg. 30-32.
A game for the Apple.

Johnson, Paul, "Sleuth," pg. 42-43.
A game for the Atari.

Johnson, Paul, "Ricochet," pg. 48-49.
A game for the Atari.

Pelczarski, Mark, "Developing Data Base," pg. 50-51, 66-68.
Handling data on the Atari.

886. The Apple Orchard 1, No. 2 (Fall, 1980)

Bishop, Bob, "Apple II Hi-Res Graphics: Resolving the Resolution Myth," pg. 7-10.
Limitations of the video hardware are discussed.

Rowe, Pete, "The Mysterious Orange Vertical Line," pg. 11.
A discussion of a recurring problem in Apple graphics.

Spurlock, Loy, "Understanding Hi-Res Graphics," pg. 12-21.
How to include text in your Hi-Res graphics programs on the Apple.

Yarkoni, Barry, "A Look Inside the Apple III," pg. 29-31.
All about the Apple III and the SOS operating system.

Crossly, John, "ASCII, EBCDIC, and the Apple," pg. 31.
A routine for converting ASCII to EBCDIC or vice-versa.

Anon., "Yes, There is a Fix for APPEND in DOS 3.2 and 3.2.1!", pg. 31.
A short routine to supply the errant File marker.

Anon., "RFI: The FCC and Your Apple," pg. 32-35.
A discussion of the Radio Interference problem and suggested remedies.

Kellner, Jo, "Pascal Operand Formats—Or, the Secret Life of a Variable," pg. 38-40.
All about Pascal Variables.

Anon., "Auto-Run Apple Without DOS," pg. 42-44.
Auto-Run programs in ROM.

Crossley, John, "Initializing Apple Peripherals with POKEs," pg. 43.
Discussion of some techniques on initializing.

Anon., "Applewriter Modification for Lower Case Display," pg. 44.
A fix for a problem using the Paymar chip.

Budge, Joseph H., "Inside Initialization," pg. 49-52.
Some improvements for the DOS.

Kamins, Scott, "Lockesmythe and the Dedicated Programmer," pg. 54-58.
Writing user-proof interactive code.

Anon., "Linking Machine Language Routines to Applesoft Programs," pg. 61.
Precautions on linking programs.

Silverman, Ken, "Don't Overload Your Apple II," pg. 67-69.
A table is given on the power requirements of various cards for the Apple.

887. Peelings II, 1, No. 3 (September/October, 1980)

"Software Review."
Devoted to review of Apple Software in considerable detail.

888. Applications (Apple Users Group, Sydney, Australia) 3, No. 1 (February, 1980)

Webster, Ian, "Basic Integer BASIC," pg. 8-11.
A tutorial on Apple Integer BASIC, with Token-Hex chart and Vector Table address chart.

Aldrich, Darrall, "Start and Length of Binary Files," pg. 12.
A program to assist in copying binary files.

Webster, Ian, "Digging into DOS," pg. 15-18.
BASICs of the Apple Disk Operating System.

Anon., "More Memory," pg. 20.
Manufacturer's Part Numbers and speed ratings for 11 types of dynamic RAM chips for the Apple. Also gives a listing for a simple memory test.

889. Compute 1, Issue 6 (September/October, 1980)

Lock, Robert, "Basically Useful BASIC," pg. 22.
Three financial programs will help you cope with inflation.

DeJong, Marvin L., "Solving Equations with a Computer," pg. 32-36.
A number of techniques to use the talents of a 6502 Micro.

Semancik, Susan, "Computers and the Handicapped," pg. 40-41.
Use the PET with the Prestodigitizer Board and Level I Braille.

Baker, Al, "Programming Hints: Apple and Atari," pg. 52-55.
More on Menu Selection using the joystick or paddles.

Ostrowsky, Sherm, "Randomize for the Apple II," pg. 59.
Make the Apple II random numbers truly random.

Schmoyer, Jeff, "Screendump," pg. 60-63.
Print out the contents of an Apple II text screen to any printer with this machine language utility.

Budge, Joseph H., "Thesus Versus the Minotaur: PASCAL Visits Ancient Greece," pg. 64-67.
A game in Apple PASCAL.

Butterfield, Jim, "Some Routines from Applesoft BASIC," pg. 68-69.
Examination specific memory dumps of routines in DISK or ROM Applesoft. Also gives Applesoft memory map of Page Zero.

Patchett, Craig, "Designing Your Own Atari Graphics Modes," pg. 71-74.
A tutorial on Atari graphics.

Schulman, Steven, "What To Do If You Don't Have Joysticks," pg. 75.
How to use keys in place of a joystick on the Atari.

Isaacs, Larry, "Screen Print from Machine Language on the Atari," pg. 76-79.
A screen printing routine for the Atari.

Veludo, Henrique, "Graphics of Polar Functions," pg. 80-81.
Plot polar functions such as spirals, roses, polygons, etc. on the Atari.

Spencer, P.T., "Waterloo Structured BASIC for the PET," pg. 82-84.
A special EPROM for the PET gives extra functions in addition to standard PET BASIC.

Butterfield, Jim, "TelePET," pg. 86-88.
All about using Modems on the PET system to communicate over the phone lines.

Baker, Robert W., "Word Pro Converter," pg. 89-91.
A utility to extend the capabilities of the Commodore Word Pro program.

Brannon, Charles, "Multitasking on Your PET? : QUADRA-PET," pg. 90-91.
Partition your PET memory into four 8K blocks to get four independent workspaces.

Poirier, Rene W., "Oops! A Crucial Update to Disk ID Changer," pg. 92-93.
A special routine to correct a useful utility for the PET.

Spencer, Peter, "Variable-Field-Length Random Access Files on the 2040 Disk Drive," pg. 94-96.
A useful file utility for the PET.

Deal, Elizabeth, "Flexible GET for the PET," pg. 98-99.
A flexible GET routine for the PET.

Butterfield, Jim, "ROM-antic Thoughts," pg. 100.
Comments to aid in making a decision on upgrading to new ROMS on the PET.

Herman, Harvey B., "Converting ASCII Files to PET BASIC," pg. 102-103.
A utility program for the PET.

Baker, Robert W., "Compactor," pg. 104-108.
Program to delete all REMarks, unnecessary spaces, leading colons, etc. on the PET.

Butterfield, Jim, "A Few Entry Points, Original/Upgrade/4.0 ROM," pg. 110.
Entry Points seen in various programmer's machine language. For the PET.

Campbell, G.A., "Feed Your PET Some Applesoft," pg. 112-119.
Using the listing given you can feed an Apple Tape program to the PET.

ADVERTISERS' INDEX

March 1981

PAINT YOUR APPLE

And don't spare any of the 21 vibrant colors provided with Datasoft's MICRO-PAINTER™ computer program.

MICRO-PAINTER™ is a modestly priced software package that bridges the gap between Apple hardware and the artist in us all.

Apple II* users can now heighten their creative and artistic IQs as they electronically paint extraordinary pictures.

And since the MICRO-PAINTER™ uses state-of-the-art technology in its programming and implementation, anyone will find the program easy to use and the results — magnificent.

Children can ease their transition into a computerized society by familiarizing themselves with computer operations while they create beautiful pictures.

Hobbyists can entertain friends with colorful designs and unusual color combinations.

Businessmen can enhance demonstrations, presentations or illustrations where the emphasis is on color.

The MICRO-PAINTER™ even magnifies images for dot-by-dot coloring, inverts colors for various color combinations and saves or displays pictures automatically.

So if you've been waiting to reveal your true artistic colors (or wishing you had more) call or write Datasoft, Inc., 16606 Schoenborn Street, Sepulveda, CA 91343, (213) 894-9154 or toll free (800) 423-5630 for details. Dealer inquiries invited.

Ask your local dealer for information on Datasoft Products.

MICRO-PAINTER

COMPUTER PAINTSET BY **Datasoft Inc.**

*Apple II is a registered trademark of Apple Computer Inc.

The home computer you thought was years away is here.

C8P DF

Ohio Scientific's top of the line personal computer, the C8P DF. This system incorporates the most advanced technology now available in standard configurations and add-on options. The C8P DF has full capabilities as a personal computer, a small business computer, a home monitoring security system and an advanced process controller.

Personal Computer Features

The C8P DF features ultra-fast program execution. The standard model is twice as fast as other personal computers such as the Apple II and PET. The computer system is available with a GT option which nearly doubles the speed again, making it comparable to high end mini-computer systems. High speed execution makes elaborate video animation possible as well as other I/O functions which until now, have not been possible. The C8P DF features Ohio Scientific's 32 x 64 character display with graphics and gaming elements for an effective resolution of 256 x 512 points and up to 16 colors. Other features for personal use include a programmable tone generator from 200 to 20KHz and an 8 bit companding digital to analog converter for music and voice output, 2-8 axis joystick interfaces, and 2-10 key pad interfaces. Hundreds of personal applications, games and educational software packages are currently available for use with the C8P DF.

Business Applications

The C8P DF utilizes full size 8″ floppy disks and is compatible with Ohio Scientific's advanced small business operating system, OS-65U and two types of information management systems, OS-MDMS and OS-DMS. The computer system comes standard with a high-speed printer interface and a modem interface. It features a full 53-key ASCII keyboard as well as 2048 character display with upper and lower case for business and word processing applications.

Home Control

The C8P DF has the most advanced home monitoring and control capabilities ever offered in a computer system. It incorporates a real time clock and a unique FOREGROUND/BACKGROUND operating system which allows the computer to function with normal BASIC programs at the same time it is monitoring external devices. The C8P DF comes standard with an AC remote control interface which allows it to control a wide range of AC appliances and lights remotely without wiring and an interface for home security systems which monitors fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring. In addition, the C8P DF can accept Ohio Scientific's Votrax voice I/O board and/or Ohio Scientific's new universal telephone interface (UTI). The telephone interface connects the computer to any touch-tone or rotary dial telephone line. The computer system is able to answer calls, initiate calls and communicate via touch-tone signals, voice output or 300 baud modem signals. It can accept and decode touch-tone signals, 300 baud modem signals and record incoming voice messages.

These features collectively give the C8P DF capabilities to monitor and control home functions with almost human-like capabilities.

Process Controller

The C8P DF incorporates a real time clock, FOREGROUND/BACKGROUND operation and 16 parallel I/O lines. Additionally a universal accessory BUS connector is accessible at the back of the computer to plug in additional 48 lines of parallel I/O and/or a complete analog signal I/O board with A/D and D/A and multiplexers.

Clearly, the C8P DF beats all existing small computers in conventional specifications plus it has capabilities far beyond any other computer system on the market today.

C8P DF is an 8-slot mainframe class computer with 32K static RAM, dual 8″ floppies, and several open slots for expansion.

Prices start at under $3,000.

Computers come with keyboards and floppies where specified. Other equipment shown is optional.

For literature and the name of your local dealer, CALL 1-800-321-6850 TOLL FREE.

OHIO SCIENTIFIC

a M/A-COM Company

1333 SOUTH CHILLICOTHE ROAD
AURORA, OH 44202 • [216] 831-5600