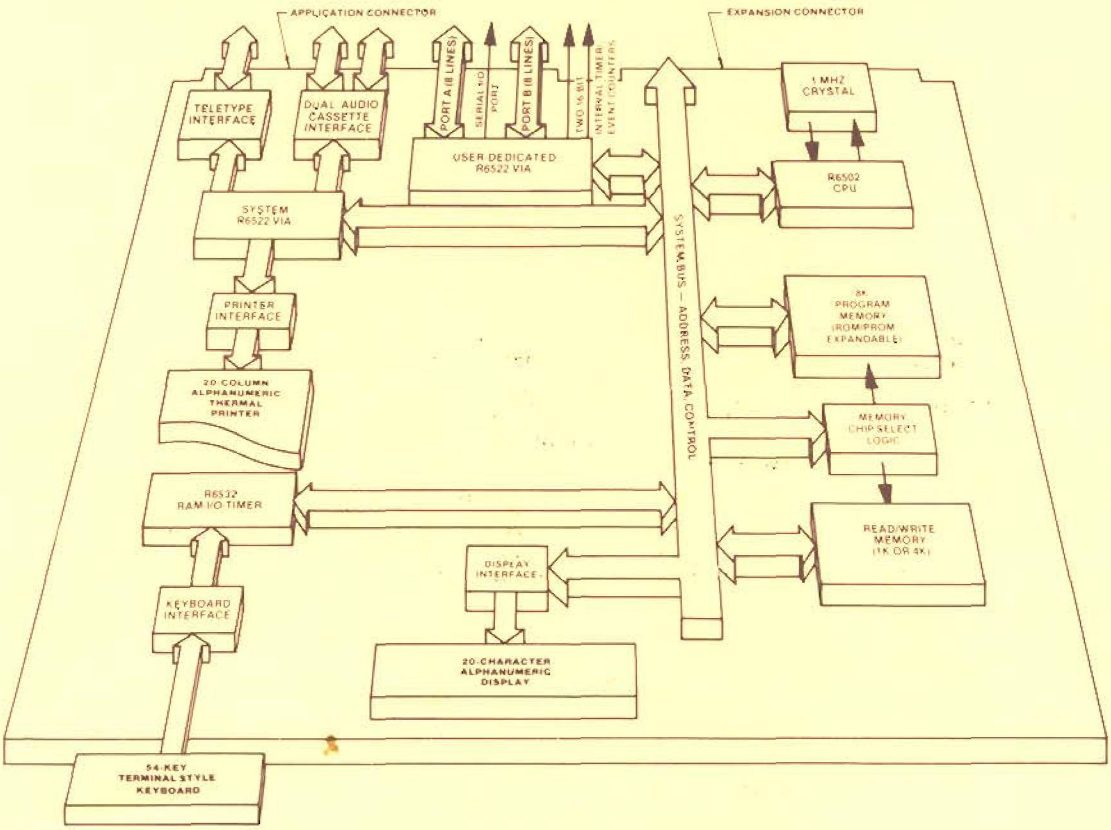
# MICRO™

## The Magazine of the APPLE, KIM, PET and Other 6502 Systems




## *Rockwell & Synertek*

## *EXPAND THE 6502 WORLD*


**NO 7** Oct - Nov 1978 **$1.50**

# MICRO™

OCTOBER/NOVEMBER 1978

ISSUE NUMBER SEVEN



Advertiser's Index



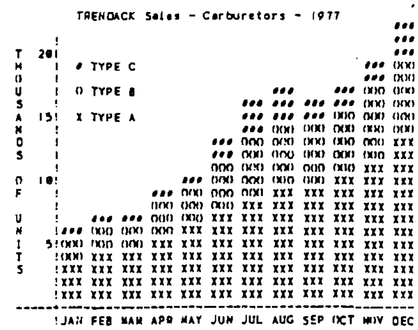

MICRO is published bi-monthly by
The COMPUTERIST, Inc., P.O. Box 3, So. Chelmsford, MA 01824.
Controlled Circulation postage paid at Chelmsford, MA 01824.
Publication Number: COTR 395770. Subscription in U.S. $6.00/6 issues.



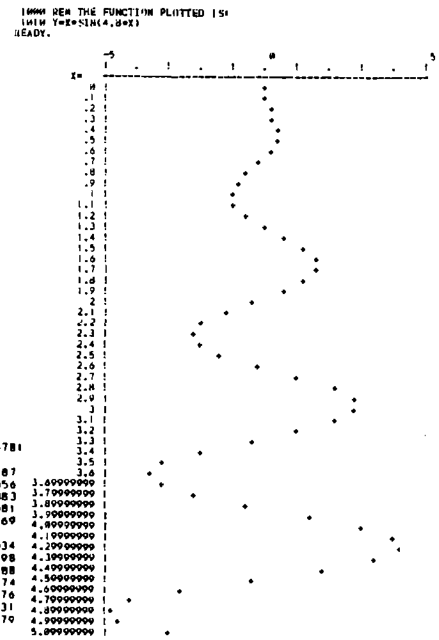

Please address all correspondence, subscriptions, and address changes to: MICRO, P.O. Box 3, So. Chelmsford, MA 01824.

# COMMODORE PET HARD COPY OUTPUT USING PET ADA 1200

```
1 OPEN 6,8:CMD 6:LIST
10 REM ARCSIN AND ARCCOS FUNCTIONS FOR THE COMMODORE PET
70 REM "Listed on a GE TermiNet 300
80 REM "using a CmC ADA 1200.
90 REM
100 REM OPEN OUTPUT FILE ON DEVICE #8.
110 OPEN 5,8
120 REM
500 REM GET A SINE VALUE
510 INPUT S
520 C=S
530 REM
1000 REM THE SINE OF THE ANGLE IS S
1010 REM IF THE SIN IS IN THE RANGE OF -1 TO 1, THEN COMPUTE.
1020 IF S<1 AND S>-1 GOTO 1050
1030 AS=90:GOTO2000
1040 REM THE ARCSINE IS AS
1050 AS=ATN(S/((1-S*S)^.5))
1060 REM THE RESULT IS IN RADIANS. CONVENT TO DEGREES.
1070 AS=AS*180/
2000 REM THE COSINE OF THE ANGLE IS C
2010 REM IF THE COSINE IS IN THE RANGE OF -1 TO 1,
2011 REM        AND NOT = TO 0, THEN COMPUTE
2020 IF C<>0 THEN 2040
2030 AC=90:GOTO 3000
2040 IF C<1 AND C>-1 THEN 2070
2050 AC=0:GOTO 3000
2060 REM THE ARCCOS IS AC
2070 AC=ATN((1-C*C)^.5/C)
2080 REM THE RESULT IS IN RADIANS. CONVENT TO DEGREES.
2090 AC=AC*180/
3000 PRINT#5," SIN,COS        ARCSIN        ARCCOS"
3010 PRINT#5,S;
3020 FOR N=1 TO 13-LEN(STR$(S)):PRINT#5," ";:NEXT
3030 PRINT#5,AS;
3040 FOR N=1 TO 13-LEN(STR$(AS)):PRINT#5," ";:NEXT
3050 PRINT#5,AC
3100 GOTO 500
READY.
```




| X | X^2 | X^3 | X^PI | X^5.25 | X^.5 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 50 | 2500 | 125000 | 217506.221 | 830983735 | 7.07106781 |
| 100 | 10000 | 1000000 | 1919487.58 | 3.16227767E+10 | 10 |
| 150 | 22500 | 3375000 | 6861077.28 | 2.65753572E+11 | 12.2474487 |
| 200 | 40000 | 8000000 | 16939435.3 | 1.20339209E+12 | 14.1421356 |
| 250 | 62500 | 15625000 | 34146855.2 | 3.88315787E+12 | 15.8113883 |
| 300 | 90000 | 27000000 | 60548854.8 | 1.01131532E+13 | 17.3205081 |
| 350 | 122500 | 42875000 | 98271022.8 | 2.27173273E+13 | 18.7082869 |
| 400 | 160000 | 64000000 | 149490141 | 4.57946723E+13 | 20 |
| 450 | 202500 | 91125000 | 216427747 | 8.49895829E+13 | 21.2132034 |
| 500 | 250000 | 125000000 | 301345240 | 1.47772127E+14 | 22.3606798 |
| 550 | 302500 | 166375000 | 406540098 | 2.43727268E+14 | 23.4520788 |
| 600 | 360000 | 216000000 | 534342301 | 3.84852281E+14 | 24.4948974 |
| 650 | 422500 | 274625000 | 687112634 | 5.85861722E+14 | 25.4950976 |
| 700 | 490000 | 343000000 | 867239597 | 8.6449943E+14 | 26.4575131 |
| 750 | 562500 | 421875000 | 1.07713798E+09 | 1.24185712E+15 | 27.3861279 |




## RS-232 PRINTER ADAPTER FOR THE COMMODORE PET

The CONNECTICUT microCOMPUTER ADApter model 1200 is the first in a line of peripheral adapters for the COMMODORE PET. The CmC ADA 1200 drives an RS-232 printer from the PET IEEE-488 bus. The CmC ADA 1200 allows the PET owner to obtain hard copy program listings, and to type letters, manuscripts, mailing labels, tables of data, pictures, invoices, graphs, checks, needlepoint patterns, etc., using a standard RS-232 printer.

The CmC ADA model 1200B comes assembled and tested, without power supplies, case, or RS-232 connector for $98.50. The CmC ADA 1200C comes complete for $169.00. Specify baud rate when ordering. (300 baud is supplied unless otherwise requested. Instructions for changing the baud rate are included.)

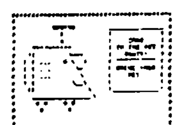
## WORD PROCESSOR FOR THE COMMODORE PET

CONNECTICUT microCOMPUTER now has a word processor program for the COMMODORE PET. This program permits composing and printing letters, flyers, advertisements, manuscripts, articles, etc., using the COMMODORE PET and an RS-232 printer.

Script directives include line length, left margin, centering, and skip. Edit commands allow the user to insert lines, delete lines, move lines, change strings, save onto cassette, load from cassette, move up, move down, print and type.

The CmC Word Processor Program addresses an RS-232 printer through a CmC printer adapter.

The CmC Word Processor Program is available for $29.50.

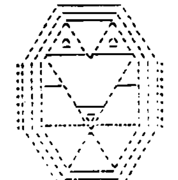


## RS-232 TO CURRENT LOOP/TTL ADAPTER

The CmC ADApter model 400 has two circuits. The first converts an RS-232 signal to a 20 ma current loop signal, and the second converts a 20 ma current loop signal to an RS-232 signal. With this device a computer's teletype port can be used to drive an RS-232 terminal, or vice versa, without modification of the port. The CmC ADA 400 can also be paralelled to drive a teletype or RS-232 printer while still using the computer's regular terminal. The CmC ADA 400 can easily be modified to become an RS-232 to TTL and TTL to RS-232 ADApter. The CmC ADA 400 does not alter the baud rate and uses standard power supplies. The current loop is isolated from the RS-232 signal by optoisolators.

The CmC ADA 400 is the perfect partner for KIM if you want to use an RS-232 terminal instead of a current loop teletype.

The CmC ADA 400S comes with drilled, plated through solder pads and sells for $24.50. The CmC ADA 400B comes with barrier strips and screw terminals and sells for $29.50.



This announcement was composed on a COMMODORE PET and printed on a GE TermiNet using a CmC ADA 1200C printer adapter and the CmC Word Processor Program.

| Qnty | Description | baud rate | price | total |
|---|---|---|---|---|
| | CmC ADA 1200B (basic) | | $98.50 | |
| | CmC ADA 1200C (complete) | | $169.00 | |
| | CmC Word Processor Program (cassette) | | $29.50 | |
| | CmC ADA 400S (solder pads) | | $24.50 | |
| | CmC ADA 400B (barrier strips) | | $29.50 | |
| | Subtotal | | | |
| | Connecticut residents add 7% sales tax | | | |
| | Handling and shipping - add per order | | | $3.00 |
| | Foreign air mail - add $5.00 per order | | | |
| | Total included with order | | | |

Mail with remittance or charge information to:


CONNECTICUT microCOMPUTER
150 Pocono Road, Room 8
Brookfield, Conn. 06804


NAME

COMPANY

ADDRESS

CITY

STATE ZIP

CHARGE TO: VISA : MASTER CHARGE  M/C INTERBANK NUMBER  :Expiration date

Credit card number

SIGNATURE

## IN THIS ISSUE . . .

With this issue we introduce a new format for MICRO. We were dissatisfied with the quality of the last couple of issues of MICRO, particularly the last issue, and decided to try a different type of printing. This new format is similar to the old, but is on lighter paper, printed on a web press, saddle stiched instead of side stapled, and does not have the old MICRO border. We have kept the features that most people said they wanted - especially the three hole punch. Of course, we will not know the quality of the new printer's product until after this goes to press. If you have any comments, let us hear from you.

Rick Auricchio, who wrote "An Apple II Programmer's Guide" in MICRO number 4, has provided another super article in "BREAKER: An Apple II Debugging Aid". This article/program allows the Apple user to debug his program with real breakpoints which permit the user to interrupt his program at any point, gain control, and then continue execution. The program, written in assembler has a lot of useful techniques and is presented in its entirety.

Those of you planning to add more RAM to your Apple II will find some valuable comparative information about 16K RAMS in Allen Watson III's article on "MOS 16K RAM for the Apple II". This info includes a table on how to decode how the various manufacturers encode their access times.

William M. Shryock Jr. presents an "Improved Star Battle Sound Effects" program for the Apple II based on the original article by Andrew H. Eliason in issue number 6.

Gary A. Creighton has a number of items for the PET under the title "PET Update". Included are a discussion of the RND (Random Number) Function use, a short program for Machine Language Storing in BASIC, some rules for USR Parameter Passing, and a machine language program to Save Mancine Language and Load Directly. A most useful set of goodies for the PET user.

Marvin L. De Jong's series on "6502 Interfacing for Beginners" continues with a discussion of "The Control Signals". The article presents the basic theoretical information, and then a program and hardware test configuration for experimenting with the control signals.

Quite often you may find that you have two sets of object code that are very similar, but not identical. It would be useful to have some way to let the computer compare the two sets of code and display the differences. This may sound simple, but since the addition of a single line of code would make all subsequent lines "different" even though they were identical except for the slight offset, it is not so simple. J. S. Green presents the solution and a program in "6502 Opcode Sequence Matcher".

Ever have doubts about your PET's memory? Then you will want to try "A Memory Test Program for the Commodore Pet" by Michael J. McCann. The program requires that the lowest 4K of memory be working and can be used to test all other memory in the PET.

Marc Schwartz presents some rules and ideas for "Apple Calls and Hex-Decimal Conversion", a useful tool when trying to generate the decimal equivalents for hex codes.

Once upon a time there were hardly any articles about 6502s at all. Now William R. Dial's "6502 Bibliography" is up to reference number 379, and this includes many multiple references. Since a reference of interest is of limited value if you do not know where to find the original, a list of "6502 Information Resources" has been compiled by William R. Dial that tells where to obtain the various magazines he has been using in the bibliography and how much they cost.

Every once in a while someone will ask "What can you do with a KIM-1 now that the PET is here?" Joseph L. Powlette and Charles T. Wright show how to use the "KIM-1 as a Digital Voltmeter".

An automated "Cassette Tape Controller" is the subject of Fred Miller's KIM article. He presents a complete hardware/software system to aid the user who wants to control cassette tapes from his KIM.

Andrew H. Eliason discusses the "Apple II High Resolution Graphics Memory Organization", and presents a few short programs that help to understand and use this feature of the Apple.

Chris Sullivan presents the first program that he wrote for the new Synertek SYM-1, "A Digital Clock Program for the SYM-1". The program is a 24 hour clock and has a number of SYM specific subroutine calls and special locations which make it a good introduction for the SYM owner.

Commodore thought they were being pretty smart making the PEEK in BASIC incapable of PEEKing at BASIC itself. Harvey B. Herman was even smarter and shows how he is "Peeking at PET's BASIC". He raises some questions about Commodore's basic strategy.

"KIMBASE" is a major program by Dr. Barry Tepperman. While the purpose of the program is to convert from almost any number system to any other, its main value to many readers may be in the numerous subroutines which provide support multiplying, dividing, and other functions.

# WE'RE STILL NUMBER ONE !

Robert M. Tripp, Editor

It's been a whole year since I sat down to write "We're Number One!" for the first issue of MICRO. Since then a lot has happened within the microprocessor/microcomputer world, and if anything, the position of the 6502 as the leader has been strengthened.

## THE 6502 MICROPROCESSOR FAMILY

There have been a couple of major changes in the basic 6500 family of microprocessor products. Most significant has been the emergence of Synertek and Rockwell International as major producers of 6500 type products. While many companies recognized that the 6500 series of products being developed by MOS Technology were in many technical aspects superior to the 8080 and 6800 product lines, they were reluctant to commit to a sole source product manufactured by a relatively small company. Now that Synertek and Rockwell have made major commitments to develop and support the 6500 line, its growth and acceptance should accelerate.

Rockwell and Synertek are not simply second sourcing existing MOS Technology products, but are undertaking a number of significant new 6500 related product developments. Rockwell has introduced the R6500/1 one-chip microcomputer. Synertek is soon to announce a 6551 ACIA. Also in the works by Rockwell and/or Synertek are a 6545 CRT Controller, a 6509 16 bit microprocessor, and a number of other products. It looks as though most development work at MOS Technology has slowed or stopped and that most of their efforts are devoted to supporting the PET and KIM-1 systems.

A searing blast at the 6502 microprocessor which was written by Jack Hemenway and appeared in EDN was very solidly "put down" by articles by several qualified writers which appeared in a later issue.

## THE 6502 MICROCOMPUTERS

This has been a very big year for 6502 based systems. Most of the trade talk and magazine articles are about the PET, TRS-80, and the Apple II, and two-out-of-three ain't bad! The Apple II was just becoming available a year ago when MICRO started, and in fact was featured on the first MICRO cover. Since then the growth of the Apple II has been one of the brightest success stories of the year. In a year when many of the original 8080 based companies found themselves in deep trouble, the 6502 based Apple Computer Company flourished. A year ago it was impossible to get a Commodore PET. They had been demonstrated at some computer shows, but were not yet available. Since then they have come on strong. The "grass roots" support for the PET seems very strong, judging from the number of small magazines that have sprung up devoted to the PET.

As our new years starts, there are two major new 6502 system developments. The Synertek SYM-1 is a single board computer which is essentially an upgrade of the KIM-1. It has more RAM, ROM, and I/O than the KIM, plus a much more powerful monitor program, plus a number of other features. It is just becoming available now, and selling for $269 with 1K RAM, is hoped to do for Synertek what the KIM-1 did for MOS Technology. The AIM 65 is Rockwell's way of announcing its serious entry into the 6502 world. This single board system includes a full typewriter style keyboard, twenty character LED display and a twenty column printer, plus room for 4K RAM, up to 20K ROM, and an extensive 8K monitor. This product is sure to generate a great deal of interest in the 6502 from a variety of users. Both Synertek and Rockwell will be selling an assembler in ROM and an 8K BASIC in ROM by the end of the year.

In addition to these major 6502 microcomputer systems, a number of other smaller manufacturers have introduced 6502 based systems in the past year. The only major drop-out during the year was ECD's MICROMIND. Since this system was never really delivered from production to any customers, it's loss was probably of little significance, except to those loyal customers who had their money tied up for a year or so.

## 6502 SOFTWARE

Whereas a year ago there were only a small handful of programs available for the 6502, there must by now be hundreds of them. Both the PET and the Apple II have generated large markets for 6502 based software, and many stores now have large quantities of programs for sale.

## MICRO

We have been very pleased with the growth of MICRO in its first year. The first issue was 28 pages long and went to about 450 subscribers and stores. This issue is twice the size and will immediately go out to about 2000 subscribers and about 1500 more copies will go to the computer stores. A distributorship has been established in Europe to handle the growing interest over there. And, due to popular demand, "The BEST of MICRO" will soon be published so that new subscribers can get the information from the first year of MICRO. Over 3000 copies of each issue have been distributed, many as "back issues" to new subscribers. We are also quite proud of the quality of the articles which have been contributed over the year. We anticipate similar growth during the coming year as the 6502 continues in phenomenal expansion.

Our plans for the coming year include increasing the size of MICRO as required to print all of the worthwhile articles we receive. Our new printing format will permit us some increase in size without requiring an increase in price. If we continue to receive more good stuff than we can print, then we will consider becoming a monthly publication. In order to serve the fast growing European market, we have arranged to have MICRO distributed by L P Enterprises in Britain. This will help keep the cost to 6502 owners in Europe reasonable.

Our success in the coming year depends on your input. We can be no better than the material submitted to us. You have done a great job so far, so keep up the good work.

# BREAKER: AN APPLE II DEBUGGING AID

Rick Auricchio
59 Plymouth Ave.
Maplewood, NJ 07040

When debugging an Assembly-language program, one of the easiest tools the programmer can use is the Breakpoint. In its most basic form, the Breakpoint consists of a hardware feature which stops the CPU upon accessing a certain address; a "deluxe" version might even use the Read/Write and Sync (instruction fetch) lines to allow stopping on a particular instruction, the loading of a byte, or the storing of a byte in memory. Since software is often easier to create than hardware (and cheaper for some of us!), a better method might be to implement the Breakpoint with software, making use of the BRK opcode of the 6502 CPU.

A Breakpoint, in practice, is simply a BRK opcode inserted over an existing program instruction. When the user program's execution hits the BRK, a trap to the Monitor (via the IRQ vector $FFFE/FFFF) will occur. In the APPLE, the Monitor saves the user program's status and registers, then prints the registers and returns control to the keyboard. The difficult part, however, comes when we wish to resume execution of the program: the BRK must be removed and the original instruction replaced, and the registers must be restored prior to continuing execution. If we merely replace the original opcode, however, the BRK will not be there should the program run through that address again.

The answer to this problem is BREAKER: a software routine to manage Breakpoints. What the debugger does is quite simple: it manages the insertion and removal of breakpoints, and it correctly resumes a user program after hitting a breakpoint. The original instruction will be executed automatically when the program is resumed!

## Is it Magic?

No, it's not magic, but a way of having the computer remember where the breakpoints are! If the debugger knows where the breakpoints are, then it should also know what the original instruction was. Armed with that information, managing the breakpoints is easy. Here's how the debugger works:

During initialization, BREAKER is "hooked-in" to the APPLE monitor via the Control-Y user command exit, and via the COUT user exit. The control-Y exit is used to process debugger commands, and the COUT exit is used to "steal control" from the Monitor when a BRK occurs.

Breakpoint information is kept in tables: the LOCTAB is a table of 2-byte addresses--it contains the address at which a breakpoint has been placed. The ADTAB is a table of 1-byte low-order address bytes; it is used to locate a Break Table Entry (BTE for short). The BTE is 12 bytes long (only the first 9 are used, but 12 is a reasonably round number) and it contains the following items:

* Original user-program instruction
* JMP back to user-program
* JMP back for relative branch targets

When adding a breakpoint, we must build the BTE correctly, and place the user-program break address into the LOCTAB. There are eight (8) breakpoints allowed, so that we have a 16-byte LOCTAB, 8-byte ADTAB, and 96 bytes of BTE's.

As the breakpoint is added, the original instruction is copied to the first 3 bytes of the BTE, and it is "padded" with NOP instructions ($EA) in case it is a 1 or 2-byte instruction. A BRK opcode ($00) is placed into the user program in place of the original instruction's opcode (other instruction bytes are not altered). The next 3 bytes of the BTE will contain a JMP instruction back to the next user-program instruction.

If the original instruction was a Relative Branch, one more thing must be considered: if we remove the relative branch to the BTE, how will it branch correctly? This problem is solved by installing another JMP instruction into the BTE for a relative branch--back to the Target of the branch, which is computed by adding the original PC of the branch, +2, +offset. This Absolute address will be placed into the JMP at bytes 7-9 of the BTE. The offset which was copied from the original instruction will be changed to $04 so that it will now branch to that second JMP instruction within the BTE; the JMP will get us to the intended target of the original Relative Branch.

A call to the routine "INSDS2" in the Monitor returns the length and type of an instruction for the "add" function. The opcode is supplied in the AC, and LENGTH & FORMAT are set appropriately by the routine.

Removal of a breakpoint involves simply restoring the original opcode, and clearing the LOCTAB to free this breakpoint's BTE.

Displaying of breakpoints prints the user-program address of a breakpoint, followed by the address of the BTE associated with the breakpoint (the BTE address is useful--its importance will be described later).

When the breakpoint is executed, a BRK occurs and the APPLE Monitor gets control. The monitor will "beep" and print the user program's registers. During printing of the registers, BREAKER will take control via the COUT exit. (Remember, we get control on every character printed - but it's only important when the registers are being printed. That's when we're at a breakpoint). While it has control, BREAKER will grab the user-program's PC and save it (we must subtract 2 because of the action of the BRK instruction). If no breakpoint exists at this PC (we scan LOCTAB), then the Mointor is continued. If a breakpoint does exist here, then the BTE address is set as the "continue PC". In other words, when we continue the user program after the break, we will go to the BTE; the original instruction will now be executed, and we will branch back to the rest of the user program.

## Using BREAKER

The first thing to do is to load BREAKER into high memory. It must then be initialized via entry at the start address. This sets up the exits from the Monitor. After a Reset, you must re-initialize via "YcI" to set up the COUT exit again. Upon entry at the start address, all breakpoints are cleared; after "YcI", they remain in effect.

To add a breakpoint, type: aaaaYcA . (Yc is control-Y). This will add a breakpoint at address 'aaaa' in the user program. A 'beep' indicates an error; you already have a breakpoint at that address. To remove a breakpoint, type: aaaaYcR. This will remove the breakpoint at address 'aaaa' and restore the original opcode. A 'beep' means that there was none there to start with.

Run your user-program via the Monitor's "G" command. Upon hitting a breakpoint, you will get the registers printed, and control will go back to the monitor as it does normally. At this point, all regular Monitor commands are valid, including "YcA", "YcR", and "YcD" for BREAKER.

To continue execution (after looking at stuff maybe modifying some things), type: YcG . This instructs BREAKER to resume execution at the BTE (to execute the original instruction), then to transfer control back to the user program. Do not resume via Monitor "G" command--it won't work properly, since the monitor knows nothing of breakpoints. To display all breakpoints, type: YcD. This will give a display of up to 8 breakpoints, with the address of the associated BTE for each one.

### Caveats

Some care must be taken when using BREAKER to debug a program. First, there is the case of BREAKER not being initialized when you run the user program. This isn't a problem when you start, because you'll not be able to use the Yc commands. But if you should hit Reset during testing, you must re-activate via "YcI", otherwise BREAKER won't get control on a breakpoint. If you try a YcG, unpredictable things will happen. If you know that you hit a breakpoint while BREAKER was not active, you can recover. Simply do a "YcI", and then display the breakpoints (YcD). Resume the user-program by issuing a Monitor "G" command to the BTE for the breakpoint that was hit (since BREAKER wasn't around when you hit the breakpoint, you have to manually resume execution at the BTE). Now all is back to normal. You can tell if BREAKER is active by displaying locations $38 and $39. If not active, they will contain $F0 FD.

It's also important to note that any user program which makes use of either the Control-Y or COUT exits can't be debugged with BREAKER. Once these exits are changed, BREAKER won't get control when it's supposed to.

BREAKER DEBUGGER: Routines to Handle up to 8
Breakpoints, for use in Debugging of User Code.

```
                        **** APPLE-2 MONITOR EQUATES
                        *
002E                    FORMAT   EQU     X'2E'          INSTRUCTION FORMAT
002F                    LENGTH   EQU     X'2F'          INSTRUCTION LENGTH
003C                    A1L      EQU     X'3C'          WORK AREA
003D                    A1H      EQU     X'3D'
003E                    A2L      EQU     X'3E'
003F                    A2H      EQU     X'3F'
0040                    A3L      EQU     X'40'
0041                    A3H      EQU     X'41'
                        *
0036                    CSWL     EQU     X'36'          COUT SWITCH WORD
0037                    CSWH     EQU     X'37'
                        *
F88E                    INSDS2   EQU     X'F88E'        DISASSEMBLER
F940                    PRNTYX   EQU     X'F940'        PRINT Y/X REGS IN HEX
FDDA                    PRBYTE   EQU     X'FDDA'        PRINT AC IN HEX
FDED                    COUT     EQU     X'FDED'        CHAR OUT
FF65                    RESET    EQU     X'FF65'        MONITOR RESET
FF69                    MON      EQU     X'FF69'        MONITOR ENTRY
                        *
                        * CHANGE 'LOWPAGE' TO LOCATE
                        *  ELSEWHERE IN MEMORY. IT IS
                        *  NOW SET FOR A 32K SYSTEM.
                        *
 0000007D               LOWPAGE  EQU     X'7D'          3 PGS BEFORE END MEMORY
7D00                             ORG     LOWPAGE**8     ORG OUT TO MEMORY TOP
7D00     4C 36 7F       INIT     JMP     INITX          =>INITIALIZATION ENTRY
                        *
                        * --- DATA AREAS --- *
                        *
7D03     00             FW1      DC      0              'FINDPC' WORK BYTE 1
7D04     00             FW2      DC      0              'FINDPC' WORK BYTE 2
7D05     00             PCL      DC      0              'GO' PC LO
7D06     00             PCH      DC      0              'GO' PC HI
                        *
                        ** SKELETON BREAK-TABLE ENTRY (BTE) **
                        *
7D07     00             SKEL     DC      0              SKELETON BTE
7D08     EA                      NOP                    NOPS FOR PADDING
7D09     EA                      NOP
7D0A     4C 00 00                JMP     0              JUMP BACK INLINE
7D0D     4C                      DC      X'4C'          JUMP OPCODE FOR BRANCHES
                        *
```

```
                                        *
                                        * -- LO ADDRESS OF BTE'S KEPT IN ADTAB -- *
                                        *
 7D0E        26                         ADTAB    DC      BTE0&255        LO ADDRESS
 7D0F        32                                  DC      BTE1&255
 7D10        3E                                  DC      BTE2&255
 7D11        4A                                  DC      BTE3&255
 7D12        56                                  DC      BTE4&255
 7D13        62                                  DC      BTE5&255
 7D14        6E                                  DC      BTE6&255
 7D15        7A                                  DC      BTE7&255
                                        *
                                        ** -- LOCTAB CONTAINS ADDRESS OF USER-PROGRAM INSTRUCTION
                                        *      WHERE WE PLACED THE BREAKPOINT IN THE FIRST PLACE.
                                        *
 7D16                                   LOCTAB   DS      2*8             SPACE FOR 16 PCH/L PAIRS
                                        *
                                        ** -- BREAK-TABLE ENTRIES (BTE'S) --- *
                                        *
 7D26                                   BTE0     DS      12              12-BYTES RESERVED
 7D32                                   BTE1     DS      12
 7D3E                                   BTE2     DS      12
 7D4A                                   BTE3     DS      12
 7D56                                   BTE4     DS      12
 7D62                                   BTE5     DS      12
 7D6E                                   BTE6     DS      12
 7D7A                                   BTE7     DS      12              ENOUGH FOR 8 BREAKPOINTS
                                        *
                                        * END OF DATA AREAS
                                        *  THE REST IS ROM-ABLE.
                                        *


                               ****************************************************************
                               *          NAME:     FINDPC
                               *          PURPOSE:  CHECK IF PC IN FW1/FW2 MATCHES ANY IN LOCTAB
                               *          RETURNS:  CARRY SET IF YES; XREG=ADTAB INDEX 0-7
                               *                    CARRY CLR IF NOT; XREG=GARBAGE
                               *          VOLATILE:DESTROYS AC
                               ****************************************************************
 7D86      A2 0F               FINDPC     LDXIM     15              BYTE-INDEX TO END OF TABLE
 7D88      AD 04 7D            FPC00      LDA       FW2             GET FOR COMPARE
 7D8B      DD 16 7D                       CMPX      LOCTAB          A PCH MATCH?
 7D8E      D0 08                          BNE       FPC02           =>NO. TRY NEXT 2-BYTE ENTRY
 7D90      AD 03 7D                       LDA       FW1             GET PCL NOW
 7D93      DD 15 7D                       CMPX      LOCTAB-1        A PCL MATCH?
 7D96      F0 06                          BEQ       FPC04           =>YES! WE HAVE A BREAKPOINT!
 7D98      CA                  FPC02      DEX                       BACK UP ONE
 7D99      CA                             DEX                        AND ANOTHER
 7D9A      10 EC                          BPL       FPC00           =>DO ENTIRE TABLE SCAN
 7D9C      18                             CLC                       =>DONE; SCAN FAILED
 7D9D      60                             RTS
                               *
 7D9E      48                  FPC04      PHA                       HOLD AC
 7D9F      8A                             TXA                       HALVE VALUE IN XREG
 7DA0      4A                             LSRA                       SINCE IT'S 2-BYTE INDEX
 7DA1      AA                             TAX
 7DA2      68                             PLA
 7DA3      38                             SEC                       SET 'SUCCESS'
 7DA4      60                             RTS


                          ****************************************************************
                          *          NAME:     BREAK
                          *          PURPOSE:  HANDLE ENTRY AT BRK AND PROCESS BREAKPOINTS
                          *          NOTE:     THIS ROUTINE GETS ENTERED ON *EVERY* 'COUT'
                          *                    CALL--IT KNOWS ABOUT BRK BECAUSE THE MONITOR'S
                          *                    REGISTERS ARE SETUP TO PRINT USER REG CONTENTS.
                          *                    AFTER PROCESSING IS DONE, IT RESTORES THE MONITOR'S
                          *                    REGS AND RETURNS.
                          ****************************************************************
 7DA5      E0 FB          BREAK      CPXIM     X'FB'           IS XREG SET FOR EXAMINE-REGS?
 7DA7      D0 27                     BNE       BRKXX           =>NO GET OUT NOW.
```

```
7DA9        C9 A0       BRK02    CMPIM    X'A0'      IS AC SETUP CORRECTLY TOO?
7DAB        D0 23                BNE      BRKXX      =>NOPE. FALSE ALARM!
7DAD        A5 3C                LDAZ     A1L        GET USER PCL
7DAF        38                   SEC                  AND BACK IT UP
7DB0        E9 02                SECIM    2            BY 2 BYTES SINCE
7DB2        8D 03 7D             STA      FW1           BRK BUMPED IT!
7DB5        A5 3D                LDAZ     A1H        GET PCH
7DB7        E9 00                SECIM    0          DO THE CARRY
7DB9        8D 04 7D             STA      FW2         AND SAVE THAT TOO
7DBC        20 86 7D             JSR      FINDPC     A BREAKER OF OURS HERE?
7DBF        90 0B                BCC      BRK04      =>NOPE. WE WON'T HANDLE IT!
7DC1        BD 0E 7D             LDAX     ADTAB      YES; GET BTE ADDRESS THEN
7DC4        8D 05 7D             STA      PCL         AND SET IT AS THE 'GO'
7DC7        A9 7D                LDAIM    LOWPAGE      PC FOR THE 'GO' COMMAND.
7DC9        8D 06 7D             STA      PCH        (OUR PAGE FOR BTE'S)
                        *
7DCC        A9 A0       BRK04    LDAIM    X'A0'      SET AC BACK FOR MONITOR
7DCE        A2 FB                LDXIM    X'FB'       AND XREG TOO
7DD0        4C F0 FD    BRKXX    JMP      X'FDF0'    =>NO. RIGHT BACK TO COUT ROUTINE!


                  ****************************************************************
                  ***      PROCESS THE 'GO' COMMAND (RESUME USER EXECUTION) **
                  *         COMMAND FORMAT: ( * Yc G ) .
                  ****************************************************************
7DD3  AD 05 7D    CMDGO    LDA      PCL                  GET RESUME PCL
7DD6  85 3C                STAZ     A1L                   AND SETUP FOR MONITOR
7DD8  AD 06 7D             LDA      PCH                   TO SIMULATE AN 'XXXX  G' COMMAND
7DDB  85 3D                STAZ     A1H                     NORMALLY.
7DDD  4C B9 FE             JMP      X'FEB9'              =>SAIL INTO MONITOR'S 'GO'


                      ****************************************************************
                      **      WE GET CONTROL HERE ON THE CONTROL-Y USER EXIT FROM THE
                      *       MONITOR (ON KEYINS). ALL COMMANDS ARE SCANNED HERE;
                      *       CONTROL WILL PASS TO THE APPROPRIATE ROUTINE.
                      ****************************************************************
7DE0        A2 FF     KEYIN     LDXIM    X'FF'            CHAR INDEX
7DE2        E8        KEYIN00   INX                       SET NEXT CHARACTER
7DE3        BD 00 02            LDAX     X'0200'          GET CHAR FROM KEYIN BUFFER
7DE6        C9 99               CMPIM    X'99'            CONTROL-Y CHARACTER?
7DE8        D0 F8               BNE      KEYIN00          =>NO. KEEP SCANNING
7DEA        E8                  INX                       BUMP OVER CTL-Y
7DEB        BD 00 02            LDAX     X'0200'          GRAB COMMAND CHARACTER
7DEE        C9 C7               CMPIM    X'C7'            IS IT 'G' (GO) ?
                      *
                      * A BRANCH-TABLE WOULD BE
                      *  NEATER, BUT IT WOULD
                      *  TAKE UP MORE CODE FOR
                      *  THE FEW OPTIONS WE HAVE.
                      *
7DF0        F0 E1               BEQ      CMDGO            =>YES.
7DF2        C9 C1               CMPIM    X'C1'            IS IT 'A' (ADD) ?
7DF4        F0 18               BEQ      CMDADD           =>YES.
7DF6        C9 C4               CMPIM    X'C4'            IS IT 'D' (DISPLAY) ?
7DF8        F0 0B               BEQ      XXDISP           =>YES.
7DFA        C9 D2               CMPIM    X'D2'            IS IT 'R' (REMOVE) ?
7DFC        F0 0A               BEQ      XXREMOVE         =>YES.
7DFE        C9 C9               CMPIM    X'C9'            IS IT 'I' (INIT) ?
7E00        F0 09               BEQ      XXINIT           =>YES.
7E02        4C 65 FF  BADCMD    JMP      RESET            NOTHING; IGNORE IT!
                      *
7E05        4C A8 7E  XXDISP    JMP      CMDDISP          EXTENDED BRANCH
7E08        4C 08 7F  XXREMOVE  JMP      CMDREMOV         EXTENDED BRANCH
7E0B        4C 4F 7F  XXINIT    JMP      CMDINIT          EXTENDED BRANCH
```

```
                         ****************************************************************
                         **       PROCESS THE 'ADD' COMMAND..ADD A BREAKPOINT AT
                         **        LOCATION SPECIFIED IN COMMAND
                         *          COMMAND FORMAT: ( * aaaa Yc A ) .
                         ****************************************************************
7E0E      A0 00          CMDADD   LDYIM     0                    CHECK OPCODE FIRST
7E10      B1 3E                   LDAIY     A2L                  OP AT AAAA A BRK ALREADY?
7E12      F0 EE                   BEQ       BADCMD               =>YES. ILLEGAL!
                         *
                         * --- SCAN LOCTAB FOR AN AVAILABLE BTE TO USE --- *
                         *
7E14      A2 0F                   LDXIM     15                   BYTE INDEX TO LOCTAB END
7E16      BD 16 7D       ADD00    LDAX      LOCTAB               GET A BYTE
7E19      D0 05                   BNE       ADD02                =>IN USE
7E1B      BD 15 7D                LDAX      LOCTAB-1             GET HI HALF
7E1E      F0 06                   BEQ       ADD04                => BOTH ZERO; USE IT!
7E20      CA             ADD02    DEX                            MOVE BACK TO
7E21      CA                      DEX                             NEXT LOCTAB ENTRY
7E22      10 F2                   BPL       ADD00                  AND KEEP TRYING!
7E24      30 DC                   BMI       BADCMD               =>DONE? ALL FULL! REJECT IT.
                         *
7E26      A5 3E          ADD04    LDAZ      A2L                  GET aaaa VALUE
7E28      9D 15 7D                STAX      LOCTAB-1             SAVE LO HALF
7E2B      8D 0B 7D                STA       SKEL+4               STUFF LO ADDR INTO BTE
7E2E      A5 3F                   LDAZ      A2H                  GET aaaa VALUE
7E30      9D 16 7D                STAX      LOCTAB               SAVE HI HALF
7E33      8D 0C 7D                STA       SKEL+5               STUFF HI ADDR INTO BTE
7E36      8A                      TXA                            GRAB INDEX FOR LOCTAB
7E37      4A                      LSRA                           MAKE ADTAB INDEX
7E38      AA                      TAX                             AND STUFF BACK INTO XREF
7E39      A9 7D                   LDAIM     LOWPAGE              BTE'S HI ADDRESS VALUE
7E3B      85 41                   STAZ      A3H                  HOLD IN WORK AREA
7E3D      BD 0E 7D                LDAX      ADTAB                GET BTE LO ADDR FROM ADTAB
7E40      85 40                   STAZ      A3L                  SAVE IN WORK AREA
7E42      A0 07                   LDYIM     7                    7-BYTE MOVE FOR SKEL BTE
7E44      B9 07 7D       ADD06    LDAY      SKEL                 GET SKEL BYTE
7E47      91 40                   STAIY     A3L                  MOVE TO BTE
7E49      88                      DEY                            SET NEXT
7E4A      10 F8                   BPL       ADD06                => MOVE ENTIRE SKELETON
7E4C      C8                      INY
7E4D      B1 3E                   LDAIY     A2L                  GET ORIGINAL OPCODE
7E4F      91 40                   STAIY     A3L                   INTO BTE
7E51      20 8E F8                JSR       INSDS2               INSDS2 (TO DISASSEMBLE)
7E54      A9 00                   LDAIM     0                    SET BRK OPCODE
7E56      91 3E                   STAIY     A2L                   OVER ORIGINAL OPCODE
7E58      A5 2F                   LDAZ      LENGTH               GET INSTRUCTION LENGTH
7E5A      38                      SEC
                         *
                         * --- SET UP JMP TO NEXT INST. IN THE BTE --- *
                         *
7E5B      A0 04                   LDYIM     4
7E5D      71 40                   ADCIY     A3L                  ADD TO PC FOR DESTINATION
7E5F      91 40                   STAIY     A3L                  STUFF INTO BTE
7E61      C8                      INY
7E62      B1 40                   LDAIY     A3L                  RUN UP THE CARRY
7E64      69 00                   ADCIM     0                     RIGHT HERE


                         ****************************************************************
                         *        DISPLAY ALL ACTIVE BREAKPOINTS
                         *         COMMAND FORMAT: (* Yc D )
                         ****************************************************************
7EA8      A2 0F          CMDDISP  LDXIM     15                   INDEX TO LOCTAB END
7EAA      BD 16 7D       DISP00   LDAX      LOCTAB               GET A BYTE
7EAD      D0 0B                   BNE       DISP04               =>IN USE
7EAF      BD 15 7D                LDAX      LOCTAB-1             TRY BOTH BYTES TO BE SURE
7EB2      D0 06                   BNE       DISP04               => DEFINITELY IN USE.
7EB4      CA             DISPNXT  DEX                            SET NEXT ENTRY
7EB5      CA                      DEX                             IN LOCTAB
7EB6      10 F2                   BPL       DISP00               => MORE TO GO
7EB8      30 C7                   BMI       CMDRET               =>DONE: EXIT TO MONITOR
                         *
```

```
7EBA        8A          DISP04    TXA                       GET INDEX
7EBB        48                    PHA                        SAVE IT
7EBC        BC 16 7D              LDYX      LOCTAB          GET SUBJECT-INST PCH
7EBF        BD 15 7D              LDAX      LOCTAB-1         AND ITS PCL
7EC2        84 3B                 STYZ      X'3B'           SET UP PCH/PCL FOR
7EC4        85 3A                 STAZ      X'3A'            DISASSEMBLER...
7EC6        AA                    TAX
7EC7        20 40 F9              JSR       PRNTYX          PRINT Y,X BYTES IN HEX
7ECA        A9 A0                 LDAIM     X'A0'           PRINT ONE
7ECC        20 ED FD              JSR       COUT             SPACE HERE
7ECF        68                    PLA                       RESTORE INDEX
7ED0        48                    PHA
7ED1        4A                    LSRA                      CONVERT TO ADTAB INEX
7ED2        AA                    TAX
7ED3        A9 BC                 LDAIM     X'BC'           '<' CHARACTER
7ED5        20 ED FD              JSR       COUT            PRINT IT
7ED8        A9 7D                 LDAIM     LOWPAGE         BTE HI ADDRESS
7EDA        85 3F                 STAZ      A2H             SET INDIRECT POINTER
7EDC        20 DA FD              JSR       PRBYTE          PRINT HEX BYTE
7EDF        BD 0E 7D              LDAX      ADTAB           GET BTE LO ADDR
7EE2        85 3E                 STAZ      A2L             SET INDIRECT POINTER
7EE4        20 DA FD              JSR       PRBYTE          PRINT BTE FULL ADDRESS
7EE7        A9 BE                 LDAIM     X'BE'           '>' CHARACTER
7EE9        20 ED FD              JSR       COUT            PRINT IT
                        *
                        * --- DISASSEMBLE THE ORIGINAL INSTRUCTION. PICK UP
                        *      ORIGINAL OPCODE FROM BTE, ORIGINAL ADDRESS
                        *      FIELD FROM USER PROGRAM LOCATION.
                        *
7EEC        A9 A0                 LDAIM     X'A0'           PRINT ONE
7EEE        20 ED FD              JSR       COUT             SPACE HERE
7EF1        A0 00                 LDYIM     0               INDEX
7EF3        B1 3E                 LDAIY     A2L             GET OPCODE FROM BTE
7EF5        20 DA FD              JSR       PRBYTE          PRINT OPCODE
7EF8        B1 3E                 LDAIY     A2L             GET OPCODE FROM BTE
7EFA        20 8E F8              JSR       INSDS2           AND GET FORMAT/LENGTH
7EFD        20 04 7F              JSR       JSRKLUGE        SNEAK INTO INSDSP @ F8D9
7F00        68                    PLA
7F01        AA                    TAX                       RESTORE LOCTAB INDEX
7F02        10 B0                 BPL       DISPNXT         => DISPLAY THE REST!
                        *
                        *-------------
                        * KLUGE ENTRY INTO SUBROUTINE
                        *  WHICH FORCES JSR PRIOR TO
                        *   A PHA INSTRUCTION. WE HAVE
                        *    TO JSR TO THIS JMP!
                        *
7F04        48          JSRKLUGE  PHA                       PUSH MNEMONIC INDEX
7F05        4C D9 F8              JMP       X'F8D9'         CONTINUE WITH INSTDSP
                        ******** END OF KLUGE! *****


                    ******************************************************************
                    *        REMOVE A BREAKPOINT AT LOCATION aaaa
                    *         COMMAND FORMAT: ( aaaa Yc R )
                    ******************************************************************
7F08        A5 3E       CMDREMOV  LDAZ      A2L             GET ADDRESS LO
7F0A        8D 03 7D              STA       FW1             HOLD IT FOR FINDPC
7F0D        A5 3F                 LDAZ      A2H             GET ADDRESS HI
7F0F        8D 04 7D              STA       FW2
7F12        20 86 7D              JSR       FINDPC          A BREAKPOINT HERE?
7F15        B0 03                 BCS       REMOV02         =>YES
7F17        4C 65 FF              JMP       RESET           =>NO: BELL FOR YOU!
                        *
7F1A        BD 0E 7D    REMOV02   LDAX      ADTAB           GET THE LOCTAB ENTRY
7F1D        85 40                 STAZ      A3L             HOLD IT
7F1F        8A                    TXA                       NOW CREATE LOCTAB INDEX
7F20        0A                    ASLA
7F21        AA                    TAX
7F22        A9 00                 LDAIM     0               CLEAR OUT THE
7F24        A8                    TAY                        APPROPRIATE
7F25        9D 16 7D              STAX      LOCTAB            LOCTAB ENTRY
7F28        9D 17 7D              STAX      LOCTAB+1           FOR THIS BKPT
```

```
7F2E        A9 7D          LDAIM    LOWPAGE               HI ADDR FOR BTE
7F2D        85 41          STAZ     A3H                   HOLD FOR ADDRESSING
7F2F        B1 40          LDAIY    A3L                   GET OPCODE OUT OF BTE
7F31        91 3E          STAIY    A2L                    AND PUT BACK INTO ORIGINAL INST
7F33        4C 69 FF       JMP      MON                   =>ALL DONE.


                          *****************************************************************
                          *        INITIALIZATION CODE. ENTERED AT START ADDR TO INITIALIZE.
                          *      IT CLEARS LOCTAB, SETS UP THE Yc AND 'COUT' EXITS.
                          *
                          *        AFTER EVERY 'RESET', MUST RESETUP WITH * Yc I .
                          *****************************************************************
7F36      A9 4C       INITX     LDAIM    X'4C'              JMP OPCODE
7F38      8D F8 03              STA      X'3F8'             STUFF IN Yc EXIT LOC
7F3B      A9 7D                 LDAIM    KEYIN/256          KEYIN: HI ADDRESS
7F3D      8D FA 03              STA      X'3FA'             STUFF INTO JMP
7F40      A9 E0                 LDAIM    KEYIN&X'FF'        KEYIN: LO ADDRESS
7F42      8D F9 03              STA      X'3F9'             STUFF INTO JMP ADDRESS
7F45      A9 00                 LDAIM    0
7F47      A2 0F                 LDXIM    15                 INDEX TO LOCTAB END
7F49      9D 16 7D    INIT00    STAX     LOCTAB             CLEAR IT OUT
7F4C      CA                    DEX                          SO THERE ARE
7F4D      10 FA                 BPL      INIT00               NO BREAKPOINTS
                      *
                      * ---- ENTER HERE AFTER HITTING 'RESET' KEY, PLEASE --- *
                      *
7F4F      A9 A5       CMDINIT   LDAIM    BREAK&255          BREAK: LO ADDRESS
7F51      85 36                 STAZ     CSWL               STUFF INTO 'COUT' EXIT HOOK
7F53      A9 7D                 LDAIM    BREAK/256          BREAK: HI ADDRESS
7F55      85 37                 STAZ     CSWH               STUFF INTO 'COUT' EXIT HOOK
7F57      4C 69 FF              JMP      MON                INIT DONE; BACK TO MON.
                                END
```

Listing 1 - BREAKER Program for Apple II

Notes on how to read the assembler listing:

A few of the syntax expressions allowed by my time-sharing cross assembler may appear cryptic. Here's a key to their meanings:

1. All HEX numbers appear as X' rather than $ expressions.

2. The ampersand (&) means logical "AND" thus:

KEYIN&X'FF'

resolves to the low-order 8 bits of the KEYIN address.

Table 1 - BREAKER Command Summary

| Command | Function |
|---|---|
| aaaa Yc A | Add breakpoint at location aaaa. Won't allow you to add one over an already existing breakpoint. Maximum of 8 breakpoints allowed. |
| Yc D | Display all breakpoints. |
| Yc I | Initialize after RESET key. Just sets up 'COUT' exit again without resetting any breakpoints. |
| aaaa Yc R | Remove breakpoint from location aaaa. Restores original opcode. |

## MOS 16K RAM FOR THE APPLE II

Allen Watson III
430 Lakeview Way
Redwood City, CA 94062

MOS 16K dynamic RAM is getting cheaper. At the time of this writing, one mail-order house is offering 16K bytes of RAM (eight devices) for $120. Apple II owners can now enhance their systems for less than the Apple dealers' price. However, there is a potential drawback to the purchase of your own 16K RAM chips: speed. You may wonder why, since the Apple's 6502 CPU is running at only about 1 MHz, but things aren't quite that simple.

To begin with, the Apple II continually refreshes its video display and dynamic RAM. It does this by sharing every cycle between the CPU and the refresh circuitry, a half-cycle for each. This means that the RAM is being accessed at a 2 MHz rate.

That doesn't sound too fast, with the slowest 16K parts rated at 300ns access time; but you have to remember that the RAM chips are 16-pin parts by virtue of a multiplexed address bus. There are two address-strobe signals during each memory access cycle, and the access-time specification will be met only if the delay between these strobe signals is within specified limits. In the Apple II this delay is 140ns, which is too long. Furthermore, the Apple II timing doesn't allow long enough RAS precharge or row-address hold time for the slow parts. Judging by the spec sheets, 200ns parts are preferable to 250ns parts, and 300ns parts shouldn't be used at all. In my Apple, 300ns parts caused a zero to turn into a one once in a while.

Many mail-order houses do not mention device speeds in their ads. The best thing to do is to deal only with those suppliers who specify speeds, but for those who didn't, the table below shows the codes used by some 16K dynamic RAM manufacturers to indicate the speeds of their devices. Good luck, and caveat emptor!

SPEED CODES USED BY 16K DYNAMIC RAM MANUFACTURERS

| | | Access Time (ns) | | | |
|---|---|---|---|---|---|
| Manufacturer | Part No. | 150 | 200 | 250 | 300 |
| A M D | 9016 | -F | -E | -D | -C |
| Fairchild | F16K | -2 | -3 | -4 | -5 |
| Intel | 2117 | -2 | -3 | -4 | |
| MOSTEK | 4116 | -2 | -3 | -4 | |
| Motorola | MCM4116C | -15 | -20 | -25 | -30 |
| National | MM5290 | -2 | -3 | -4 | |
| N E C | µD416 | -3 | -2 | -1 | |
| T I | 4116 | -15 | -20 | -25 | |
| Zilog | Z6166 | -2 | -3 | -4 | |

---

## IMPROVED STAR BATTLE SOUND EFFECTS

William M. Shryock, Jr.
P.O. Box 126
Williston, ND 58801

```
10  POKE 0,160: POKE 1,1: POKE
    2,162: POKE 3,0: POKE 4,138
    : POKE 5,24: POKE 6,233: POKE
    7,1: POKE 8,208: POKE 9,252
    : POKE 10,141
20  POKE 11,48: POKE 12,192: POKE
    13,232: POKE 14,224: POKE 15
    ,150: POKE 16,208: POKE 17,
    242: POKE 18,136: POKE 19,208
    : POKE 20,237: POKE 21,96
30  CALL -936: VTAB 12: TAB 9: PRINT
    "STAR BATTLE SOUND EFFECTS"

40  SHOTS= RND (15)+1
50  LENGTH= RND (11)*10+120
60  POKE 1,SHOTS: POKE 15,LENGTH:
    CALL 0
70  FOR DELAY=1 TO RND (1000): NEXT
    DELAY
80  GOTO 40
```

This version can be used in low res. programs without having to reset HIMEM. Also it can all be loaded from BASIC.

Gary A. Creighton
625 Orange Street, No. 43
New Haven, CT 06510

I am writing this article because I'm tired of seeing the same rehash of pseudo-facts being repeated about the PET. If I read one more time about the small keyboard or the RND function not working correctly...! As you will see, the 2001 has an extremely well designed Interpreter which can be used effectively as subroutines either from the SYS command, or the USR command. Parameter passing will be revealed as an easy operation, and returning USR with a value is just as simple. The RND function may be substituted with a twelve byte USR program to make it completely random and non-repeating (as it stands, it repeats every 24084 times through) and I will show the use of negative arguments. Unfortunately, RND(0) was apparently a mis-calculation on Microsoft's part. They figured that ROM empty locations would turn out to be more random than the end product shows. They load non-existent memory locations into the RND store area (218-222) thus causing a resulting RND value which fluctuates between a few different values. When ROM is finally installed in that area (36932) the RND(0) will have the dubious quality of being some fixed number.

### RND FUNCTION USE

The RND function may be set at any time to execute a known series of RND #'s by using a known negative argument just before RND with a positive one. The ability to have available a known list of random numbers is very important in a lot of sciences.

```
10 R=RND(-1)
20 FOR X=1 TO 5
30 PRINT INT(1000*RND(1)+1),
40 NEXT X
```

Gives the sequence: 736, 355, 748, 166,629

Since RND(-low#) gives such a small value, use a negative argument in the range (-1 E10 to -1 E30) if you need one repeatable RND number with a useful value, e.g., RND(-1 E20)= .811675238.

Concerning the true random nature of RND and it's ability to act randomly at all times; time must be combined with RND. This is possible with a RANDOMIZE subroutine or faster still, redoing RND(+) with a USR routine.

```
10000 REM (RANDOMIZE)
10010 R1=PEEK(514) : R2=PEEK(517)
10020 POKE 220, R1 : POKE 221, R2
10030 RETURN
```

This routine may be used at program initialization and as the program halts for an INPUT. It will start a new sequence of RND numbers whenever called.

When the computer does a sequence without intervention, the following USR program is suggested which will return a truly random number quickly; without repeating.

```
10 REM (TRUE RND USING USR FUNCTION)
20 POKE 134,214 : POKE 135,31 : CLR
30 FOR X=8150 TO 8165
40 READ BYTE : POKE X, BYTE
50 NEXT X
60 DATA 173,2,2,133,220,173,5,2,133,221,76
65 DATA 69,223,0,0,0
70 POKE 1, 214 : POKE 2, 31
```

### MACHINE LANGUAGE STORING IN BASIC

When using machine language, always precede storing by setting up BASIC's upper boundary. This is done by:

```
     POKE 134, ITEM : POKE 135, PAGE : CLR
e.g. POKE 134, 0    : POKE 135, 25   : CLR
```

sets upper boundary to 6400 and BASIC use will be confined to 1024 to 6399 unless reset or turned off.

You can use the following program for storing decimal. Changing INDEX to 10000 to appropriate position and typing in DATA lines in 100 to 9997.

```
0 REM ("MACHINE STORE")
1 REM WRITTEN BY GARY A. CREIGHTON, JULY 78
2 REM ( SET INDEX=ORIGIN IN LINE 10000 )
3:
15 REM FIX UPPER STRING BOUNDARY
20 GOSUB 10000
25 X=INDEX / 256
30 PAGE=INT(X)
35 ITEM=(X-PAGE)* 256
40 POKE 134, ITEM
45 POKE 135, PAGE
50 CLR
55 :
60 REM  LOAD MACHINE LANGUAGE
65 GOSUB 10000 : LOC=INDEX
70 READ BYTE : IF BYTE<0 THEN END
75 POKE LOC, BYTE
80 LOC=LOC+1 : GOTO 70
85 :
90 REM MACHINE LANGUAGE DATA
100 DATA
     .
     .
9997 DATA
9998 DATA 0,0,0,-1
9999 :
10000 INDEX=(START OF MACHINE LANGUAGE)
10010 RETURN
```

### USR PARAMETER PASSING

The following are parameter passing rules for the USR function and should be added to the "MACHINE STORE" program.

```
0 REM ("USR(0 TO 255)")
46 POKE 1, ITEM
48 POKE 2, PAGE
100 REM (USR INPUT 0-255; OUTPUT 0-255)
110 DATA 32,121,214    : REM JSR 54905
120 DATA (Your program using input value)
     .
     .
5000 DATA (Setup output value in Accum.)
5010 DATA 76,245,214    : REM JMP 55029
10000 INDEX 6400
```

OR

```
0 REM ("USR(0 TO 65535)")
46 POKE 1, ITEM
48 POKE 2, PAGE
100 REM (USR INPUT 0-65535;OUTPUT 0-65535)
110 DATA 32,208,214  : REM JSR 54992
    (Note: Check if 0-65535. RTS with:
           Y and M(8)= ITEM
           A and M(9)= PAGE
120 DATA (Your program using 2 byte passed
         value)
     .
     .
5000 DATA (Setup output vlaue ITEM in Y;
           PAGE in A)
5010 DATA 132,178      : REM STYZ 178
5020 DATA 133,177      : REM STAZ 177
5030 DATA 162,144      : REM LDXIM 144
5040 DATA 56           : REM SEC
5050 DATA 76,27,219    : REM JMP 56091
     (Setup output value and RTS)
```

The input parameter may be any complex expression and you can of course:

input 0-255 and output 0-65535, or
input 0-65535 and output 0-255.

### SAVE MACHINE LANGUAGE AND LOAD DIRECTLY

The reason for the 0,0,0 at the end of the preceding machine language programs is that the saving routine described next SAVES machine language until 0,0,0 or an ERROR is printed. After it has been saved in this way, it may be LOADED and VERIFIED with little effort.

Add to "MACHINE STORE" program (all assembly is in decimal).

```
        0 REM ("SAVEM")
        100 REM ERAM=31 (or last page of RAM on your PET)
        110 DATA 32,200,0     : REM  JSR   200    check if : or end of line
        120 DATA 208,3        : REM  BNE   OVER
        130 DATA 76,158,246   : REM  JMP   63134  jump 'SAVE' if SYS 8000 only
 OVER   140 DATA 32,17,206    : REM  JSR   52753  check if ','
        150 DATA 32,164,204   : REM  JSR   52388  analyze arithmetical argument
        160 DATA 32,208,214   : REM  JSR   54992  check if 0-65535
        170 DATA 132,247      : REM  SYTZ  247    'save from' item
        180 DATA 133,248      : REM  STAZ  248    'save from' page
        190 DATA 170          : REM  TAX
        200 DATA 152          : REM  TYA
        210 DATA 208,1        : REM  BNE   OVR2
        220 DATA 202          : REM  DEX
 OVR2   230 DATA 136          : REM  DEY          back up 1
        240 DATA 132,80       : REM  STYZ  80     initialize CHK pointer item
        250 DATA 134,81       : REM  STXZ  81     initialize CHK pointer page
        260 DATA 169,173      : REM  LDAIM 173
        270 DATA 133,79       : REM  STAZ  79     LDA instruction in 0079
        280 DATA 169,96       : REM  LDAIM 96
        290 DATA 133,82       : REM  STAZ  82     RTS instruction in 82
        300 DATA 32,200,0     : REM  JSR   200
        310 DATA 201,44       : REM  CMPIM 44     check if ',' before filename
        320 DATA 208,3        : REM  BNE   OVR3
        330 DATA 32,194,0     : REM  JSR   194    move code pointer over ','
 OVR3   340 DATA 32,51,244    : REM  JSR   62515  get options for "SAVE'
 AGAIN  350 DATA 230,80       : REM  INCZ  80
        360 DATA 208,2        : REM  BNE   OVR4
        370 DATA 230,81       : REM  INCZ  81     add 1 to CHK pointer
 OVR4   380 DATA 32,79,0      : REM  JSR   79     look at next CHK code
        390 DATA 208,27       : REM  BNE   CHEND
        400 DATA 160,1        : REM  LDYIM 1      check for 0,0,0
        410 DATA 177,80       : REM  LDAIY 80
        420 DATA 208,21       : REM  BNE   CHEND
        430 DATA 200          : REM  INY
        440 DATA 177,80       : REM  LDAIY 80
        450 DATA 208,16       : REM  BNE   CHEND
        460 DATA              24     : REM CLC
        470 DATA              165,80 : REM LDAZ   80
        480 DATA              105,4  : REM ADCIM  4
        490 DATA              13
        460 DATA 24           : REM  CLC
        470 DATA 165,80       : REM  LDAZ  80
        480 DATA 105,4        : REM  ADCIM 4
        490 DATA 133,299      : REM  STAZ  229    'save to' item
        500 DATA 165,81       : REM  LDAZ  81
        510 DATA 105,0        : REM  ADCIM 0
        520 DATA 133,230      : REM  STAZ  230    'save to' page
        530 DATA 76,177,246   : REM  JMP   63153  complete 'SAVE'
```

```
CHEND   540 DATA 165,81       : REM  LDAZ  81
        550 DATA 201,31       : REM  CMPIM ERAM
        560 DATA 240,10       : REM  BEQ   CHKNF  check: 'not found' if last
        570 DATA 144,210      : REM  BCC   AGAIN  look at next if less than
        580 DATA 32,184,31    : REM  JSR   END
        590 DATA 162,85       : REM  LDXIM 85
        600 DATA 76,108,195   : REM  JMP   70028  ("?END) NOT FOUND ERROR"
CHKNF   610 DATA 165,80       : REM  LDAZ  80
        620 DATA 201,253      : REM  CMPIM 253
        630 DATA 144,196      : REM  BCC   AGAIN  again if enough room
        640 DATA 32,184,31    : REM  JRS   END
        650 DATA 160,40       : REM  LDYIM 40
        660 DATA 76,133,245   : REM  JMP   62853  ("?END) NOT FOUND ERROR"
END     670 DATA 169,13       : REM  LDAIM 13
        680 DATA 32,234,227   : REM  JSR   58346
        690 DATA 169,63       : REM  LDAIM 63
        700 DATA 32,234,227   : REM  JSR   58346
        710 DATA 169,69       : REM  LDAIM 69
        720 DATA 32,234,227   : REM  JSR   58346
        730 DATA 169,78       : REM  LDAIM 78
        740 DATA 32,234,227   : REM  JSR   58346
        750 DATA 169,68       : REM  LDAIM 68
        760 DATA 32,234,227   : REM  JSR   58346  "?END"
        770 DATA 96           : REM  RTS
        780 REM (FORMAT: SYS 8000,INDEX,"FILENAME",DEVICE#,I/O OPTION)
```

After typing and saving normally, type RUN when READY. Save "SAVEM" using itself to save itself by typing:

SYS 8000,8000, "SAVE(SYS 8000)"

when READY., REWIND TAPE #1 and type:

VERIFY "SAVE(SYS 8000)"

**MACHINE LANGUAGE LOAD PROCEDURE**

After SAVEing machine language, you have the capability of LOADing directly if you follow these rules.

Loading machine language before BASIC program:

```
LOAD "machine language name"
NEW
A=PEEK(247) :B=PEEK(248)
POKE 134,A  :POKE 135,B
POKE 1,A    :POKE 2,B (only if USR, not SYS)
CLR
```

Then LOAD BASIC Program.

Loading machine language from BASIC program:

```
0 IF OK THEN RUN 6
1 OK=-1 : PRINT "PRESS REWIND ON TAPE #1"
2 WAIT 519,4,4 : REM wait til stop if play down but not motor
3 WAIT 59411,8,8 : REM wait til key on cassette pushed
4 WAIT 59411,8 : REM wait til stop on cassette pushed
5 LOAD "machine language name"
6 A=PEEK(247) : B=PEEK(248)
7 POKE 134,A  : POKE 135,B
8 POKE 1,A    : POKE 2,B : REM (only if USR, not SYS)
9 CLR
10 REM (BEGIN BASIC PROGRAM, MACHINE LANGUAGE LOADED)
```

# MICRO

## SUBSCRIPTION AND RENEWAL INFORMATION

If you are a subscriber to MICRO, then the code following your name on the mailing label is the number of the last issue your current subscription covers. If your code is 07, then this is your last issue. MICRO will NOT send out renewal notices. So, if your number is coming up, get your subscription renewal in soon. and, please check your label for correct address and notify us of any corrections or changes.

MICRO is currently published bi-monthly. The first issue was OCT/NOV 1977. The single copy price is $1.50. Subscriptions are $6.00 for six issues in the USA. Six issue subscriptions to other countries are listed below.
[Payment must be in US $.]

| | | |
|---|---|---|
| Surface: | Canada/Mexico | $7.00 |
| | All other countries | $8.00 |
| Air Mail: | Europe | See European Distributor Rates |
| | South America | $14.00 |
| | Central America | $12.00 |
| | All other countries | $16.00 |

Name: . . . . . . . . . . . . . . . . . . . . .

Addr: . . . . . . . . . . . . . . . . . . . . .

City: . . . . . . . . . . . . . . . . . . . . .

State: . . . . . . . . . . Zip: . . . . . . . .

Country: . . . . . . . . . . . . . . . . . . .

Amount: $ . . . . . . . . Start MICRO #: . . .

Send payment to:

MICRO, P.O. Box 3, S. Chelmsford, MA 01824, USA

Your name and address will be made available to legitimate dealers, suppliers, and other 6502 interests so that you may be kept informed of new products, current developments, and so forth - unless you specify that you do not wish your name released to these outside sources.

# 6502 INTERFACING FOR BEGINNERS; THE CONTROL SIGNALS

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

By now your breadboard should look like a rat's nest so we shall add just a few more wires. So far you have used several decoding chips to produce device select pulses (also called chip selects, port selects, etc.) These pulses activate a particular I/O port, memory chip, PIA device, interval timer or another microcomputer component. Almost all of these components must "know" more than that they have been addressed. They must know if the microprocessor is going to READ data from them or WRITE to them. The R/W control line coming from the R/W pin on the 6502 provides this information. It is at logic 1 for a READ (typically LDA XXXX) and at logic 0 for a WRITE (typically STA XXXX).

If you have ever tried to wrap your mind around timing diagrams for microcomputer systems you soon realize that system timing is also important. Suppose that a memory chip is selected by a device select pulse. A 21L02 chip, after being selected, must decode the lowest 10 address lines itself to decide which of its 1024 flip-flops will become the output data. This takes time, so the data at the output pin is not ready instantaneously. The 6502 simply waits for a specified amount of time, and at the end of this period it reads the information on the data bus. If the access time of the chip is too long, the 6502 will read garbage; otherwise it will get valid data.

Likewise, during a WRITE cycle, the microprocessor brings the R/W line to logic 0, selects the device which is to receive the data, and at the end of a cycle it signals the divice to read the data which the 6502 has put on the data bus. The signal which successfully concludes both a READ and a WRITE instruction is the so-called phase-two clock signal symbolized by $\phi_2$. In particular, it is the trailing edge (positive to zero transition) of this signal which is used.

All the timing for the microcomputer is done by the crystal oscillator on the microcomputer board and the clock circuitry on the microprocessor itself. A clock frequency of 1 MHz produces a machine cycle of 1 microsecond in duration. Near the beginning of the cycle the address lines change to select the divice which was addressed, and the R/W goes to logic 1 or logic 0 depending on whether a READ or a WRITE was requested. If a READ was requested, some device in the system responds by putting data on the data bus. Typically this happens during the second half of the cycle when $\phi_2$ is at logic 1. Finally, at the end of the cycle, but before the address lines or the R/W line have changed, $\phi_2$ changes from logic 1 to logic 0, clocking the data into the 6502. The same kinds of things happen during a WRITE cycle, except that now the external device uses the trailing edge of the $\phi_2$ signal to clock the data, while the 6502 puts the data on the bus at a slightly earlier time in the cycle. For details refer to the 6502 HARDWARE MANUAL.

The circuits you have built so far, together with a few more chips, will demonstrate the effect of the control signals. Refer to Figure 1 of the last installment of this column (MICRO, Issue 6, p. 30), and to Figure 1 of this issue. You will see the LS145 and the LS138 have not been changed too much, in fact all of the connections to the LS145 should stay the same. The device select pulse from the LS145 goes to G2A as before, but another signal goes to G2B in the new Figure 1. For the moment disregard the lower LS138 and LS367 in Figure 1 of this issue. The new signal to G2B of the LS138 is our WRITE signal. It is produced by NANDING the $\overline{R/W}$ signal with $\phi_2$ and it is an active-low signal. On the KIM-1 it is called RAM-R/W and is available on the expansion connector. Most other 6502 systems will very likely also have a RAM-R/W signal.

Its effect in Figure 1 is to inhibit the device select pulse from the LS138 whenever the R/W line is high (during all READ instructions), but to allow the device select pulse to occur when the R/W line is low <u>and</u> $\phi_2$ is high. Thus, the top LS138 in Figure 1 selects output ports only, and the device select pulse from it terminates on the trailing edge of the $\phi_2$, producing a logic 0 to logic 1 transition simultaneously (almost) with $\phi_2$. This pulse is inverted by the LS04. Consequently, a WRITE instruction produces a positive pulse at the G inputs of the LS75 whose duration is about 1/2 microsecond and whose trailing edge coincides with $\phi_2$.

The 74LS75 is a 4-bit bistable latch whose Q outputs follow the D (data) inputs only when the G inputs are at logic 1, in other words during the device select pulse from the LS04 inverter. The trailing edge of this pulse latches the Q outputs to the value of the D inputs during the device select pulse. If you had a great deal of trouble following this, you may want to check the reverse side of this page to make sure there is nothing valuable on it and then destroy this by burning or shredding! Otherwise proceed to to the experiment below.

Connect the circuit shown in Figure 1, omitting for the time being the lower LS138 and the LS367. You can also omit the connection of address line A3 to G1 on the top LS138 if G1 is connected to +5V as was indicated in the last issue. In other words, simply add the LS04 and the LS75 to your circuit of the last issue. The RAM-R/W signal must also be generated if your 6502 board does not have one. Simply use one <u>inverter</u> on the LS04 to invert the R/W signal to $\overline{R/W}$, then NAND it with the $\phi$ , and run the output of the NAND gate to the G2B pin on the LS138.

The address of the device is 800F if the connections are made as shown in the figure. If other pins on either the LS145 and/or the LS138 are changed the address will be different. The switches shown connected to the D inputs may be implemented with a DIP switch or jumper wires. An open switch corresponds to a logic 1 while a closed switch is logic 0. Set the 4 switches to any combination then load and run the following program:

```
0200  8D 0F 80     STA DSF.
```

The LEDs should indicate the state of the switches. If you add the statements

```
0203  4C 00 02     JMP START
```

then you should be able to change the switches and the LEDs will follow the switches. Try substituting an AD 0F 80 (LDA DSF) for the 8D 0F 80 instruction. Nothing should happen, even though the same address is being selected, because on LDA instruction the R/W line is high, inhibiting the LS138 from producing a device select. Finally, connect the data lines D0-3 from the 6502 to the D-inputs of the LS75, making very sure that the LS145 is de-selecting other locations. On the KIM-1 this means that pin 1 of the LS145 is connected to pin K on the application connector and pin 9 of the LS 145 is connected to pin J. The appropriate pull-up resistors must also be added. With the data lines connected run the following program:

```
0200  A9 04      LDAIM $04
0202  8D 0F 80     STA DSF.
```

Play around with different numbers in LDAIM instruction and explain your results. If nothing seems to make sense, it may be that your data lines need to be buffered, a topic we will take up next issue. If your results make sense you will have discovered that we have configured a 4-bit output port whose address is 800F. Adding another LS75 to connect to data lines D4-D7 and whose G connections also go to the output of the LS04 will give an 8-bit output port. Seven other output ports, addresses 8008 through 800E, could be added using the other device select signals from the LS138, LS04 inverters, and LS75 latches.

If you want to make an input port wire the circuit for the lower LS138 in Figure 1. If you don't have much more room on your circuit board you might want to simply reconnect the upper LS-138 to become the lower LS138. A couple of connections do the trick. Set the switches to anything you like and run the program below.

KIM-1 users should see the hex equivalent of the switch settings appear in the right-most digit on the display. Owners of other systems can omit the last two lines of the program, stop it, and examine the location 00F9 to see that the lowest four bits agree with the switch settings. Experiment with other switch settings to make sure that everything is operating correctly.

The completed circuit of Figure 1 gives one 4-bit output port (provided the data lines are connected to the D inputs of the LS 75) and one 4-bit input port, addresses 800F and 8007 respectively. These two ports are easily expanded (two more chips) to become 8-bit ports. Likewise the circuit of Figure 1 could be expanded to give a total of eight 8-bit input ports and eight 8-bit output ports.

Next issue we will look at a slightly different input port, and we will look in more detail into three-state devices and the data bus. You may want to keep your circuit together until then.

```
0200  AD 07 80  START  LDA DS7      Read input port data
0203  85 F9            STA DISP     and store it in location 00F9.
0205  20 1F 1F         JSR SCANDS   Jump to KIM display subroutine.
0208  4C 00 02         JMP START    Repeat program.
```

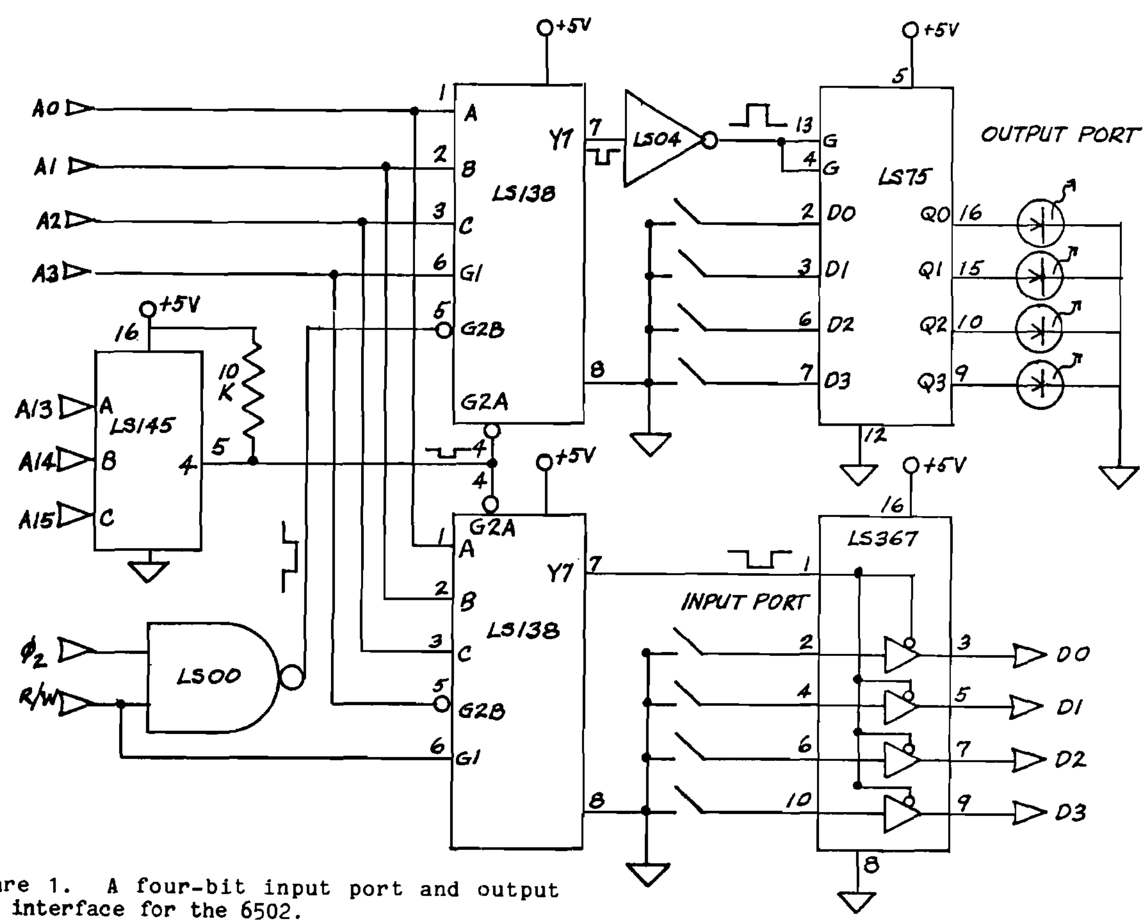



Figure 1. A four-bit input port and output port interface for the 6502.

# 650X OPCODE SEQUENCE MATCHER

J. S. Green
807 Bridge Street
Bethlehem, PA 18018

The motivation for writing this program stemmed from the fact that I have two machine code versions of the same 650X assembler (ASM65 by Wayne Wall, dated 1 May 77 and 13 Jun 77 respectively) but I only have a listing of the older version. Both are just short of 4 K bytes long. I wished to make some local changes to the newer version and therefore needed to establish a means of correspondence between it and the listing. A disassembler is helpful here but not adequate because of discontinuities in the two codes which make forward references very difficult to correlate manually.

I felt that when a program has been heavily modified, many opcode sequence segments whould remain constant even while their respective operands differ. Therefore, what was needed was a program that would correlate and point to parallel sequences of opcodes.

Several assumptions were made in order to simplify the programming task. It was presumed that the basic order of appearance of major portions of the code would be the same since there seemed to be little advantage in shuffling the deck, as it were. Also, in order to minimize the effect of spurious matches, it was decided that only significant sequences need be reported and that no portion of the code would be reported as a match more than once. This position saves the program, for example, from reporting every possible LDA,STA opcode sequence pairing (or even all of those of the same address mode).

## Process Description

As written, the scanning process of the matching program starts at the beginning of the two code strings, A and B, to be examined. Both initial positions are assumed to contian opcodes. An index or pointer to the B string is, in effect, moved along B, from opcode to opcode, until a match with the current A string opcode is found. If no match is found before the B list is exhausted, the A pointer is moved to the next A opcode position while the B pointer is reset to its previous starting point. This general procedure is repeated until the A list is exhausted, at which time the program terminates.

When a match is found, both pointers are moved together along their respective lists, from opcode to opcode, until the opcodes fail to match each other. If the matching sequence is significantly long the size and the start and end of both segments is displayed. The search for additional matching segments is resumed from the end of the just-reported segments so that their opcode elements cannot be matched more than once.

If the completed sequence is not significant, it is not displayed and the search is resumed from where the short sequence began, as if there had been no match at all.

The definition of significance refers to the minimum acceptable number of matching codes in a continuous sequence. The particular values used are left to the user. While our experience has shown a minimum value of eight to be useful, the actual values should reflect the length of the code being examined and the degree to which it has been hacked up.

The effect of a too-low significance value often results in a fewer number of matches being reported, rather than more as one might expect. This is because a spurious match of short segments can have the effect of masking out longer possible matches which would use the same code items were they still available.

## Operation

To operate the opcode matching program both lists of code must be in memory. They may be in ROM. They need not be at their operating address. (Indeed, if they have the same address at least one must be somewhere else anyway). Since the matching program reports storage, rather than operating addresses it is useful to choose storage addresses that have some degree of correspondence to the operating addresses, e.g., code operating at $21E3 might be stored at $41E3.

Enter initial values (all in hex LO,HI) as follows:

| | |
|---|---|
| $0000,01 | Significance value |
| $0002,03 | Start of list A |
| $0004,05 | Start of list B |
| $0006,07 | End of list A |
| $0008,09 | End of list B |

Only the starting address will be modified during program execution. The program will initially assume that the value at the start location is an opcode.

To run the program enter at OPMACH. As written, it will terminate by jumping to the monitor from END01. The routine may be made into a subroutine by placing an RTS here.

Since the program cranks the data a lot, there will be what seem to be long pauses between outputs. The program requires about 2 minutes to compare the aforementioned assemblers.

## Results

Several sets of results, using significance values of $06, $08 and $0A are shown below. In order to have both versions of code resident at the same time, it was necessary to store one version, at address $4000.

About 64 percent of the code of the two versions of the assembler correlate when a significance value of 8 is used. This is a reasonable percentage when one considers the fact that the non-significant, non-reported, sequences are easily identified since they lie in the same relative position between reported sequences.

An extensive manual comparison of the two code sets was made. (*So much for the work-saving aspects of the program!*) No false matches were identified when a significance value of 8 was used.

## Variations for Text Processing

Interesting variants of the program are possible. By altering or replacing the list pointer increment routines, AINC and BINC, the nature of the list pointer incrementation may be changed from the current conditional increment based on opcode to some other condition or to a constant such as plus one.

With a constant increment of one, the matching program may be used to compare sequences of any textural material in a somewhat crude, one for one fashion.

By having separate increment subroutines when seeking to locate the start of a matching segment in contrast to the incremental routines used when "running-out" a sequence, some fairly powerful text processing capabilities may be obtained at little additional cost. For example, when seeking to locate matching segments in natural language text, we might wish to start with the initial character of alphabetic strings, i.e., words. Therefore, by incrementing past all non-alphabetic characters to the next alphabetic character we can both speed up the process and insure that our sequences start with (what we have operationally defined as) words.

Similar techniques may be employed in the (now separate) within sequence increment routines to ingore, (i.e., increment past,) any non-alphabetic characters such as control characters, numbers, punctuation or whatever we like. Thus we are able to obtain a far more flexible and hopefully more useful definition of a matching sequence.

## Conclusions

The general techniques illustrated here are both effective and useful. The conditional matching approach has not been fully explored, but it is clear that it has interesting possibilities in the area of text processing. In the present application, correlating two lenghty strings of machine code, the approach made practical what otherwise would have been a difficult and dull task.

```
                   ;       **** OPCODE SEQUENCE MATCHER ****
                   ;       VERSION 1.04, 18 AUG 78
                   ;
                   ;       COPYRIGHT,1978
                   ;       COMMERCIAL RIGHTS RESERVED
                   ;       EXCEPT AS NOTED BY

                   ;       J. S. GREEN, COMPUTER SYSTEMS
                   ;       807 BRIDGE STREET
                   ;       BETHLEHEM, PA  18018
                   ;       (215) 867-0924
                   ;
                   ;       NOTE: THE BYTCNT SUBROUTINE IS FROM
                   ;       H. T. GORDON IN DDJ, #22 P.5.
                   ;       (COPYRIGHT BY PEOPLE'S COMPUTER COMPANY)
                   ;
                   ;
                   ;
                           .LOC   $0000
                   ;
                   ;       USER DEFINED VARIABLES (LO,HI)
0000 00 00         SIGNIF: .WORD              ;SIGNIFICANCE
0002 00 00         ABASE:  .WORD              ;START OF LIST A
0004 00 00         BBASE:  .WORD              ;START OF LIST B
0006 00 00         AMAX:   .WORD              ;END OF LIST A
0008 00 00         BMAX:   .WORD              ;END OF LIST B
                   ;
                   ;       OTHER PROGRAM VARIABLES
000A 00 00         APOINT: .WORD              ;LIST A POINTER
000C 00 00         BPOINT: .WORD              ;LIST B POINTER
000E 00 00         ASAVE:  .WORD              ;LIST A SEQUENCE START
0010 00 00         BSAVE:  .WORD              ;LIST B SEQUENCE START
0012 00 00         COUNT:  .WORD              ;SEQUENCE COUNTER
                   ;
                   ;       EXTERNAL SUBROUTINES (IN KIM)
                           .DEF   START=$1C4F        ;MONITOR RETURN POINT
                           .DEF   CRLF=$1E2F         ;CARRIAGE RETURN
                           .DEF   OUTCH=$1EA0        ;DISPLA A CHAR
                           .DEF   PRTBYT=$1E3B       ;DISPLA HEX BYTE
                           .DEF   OUTSP=$1E9E        ;DISPLA A SPACE
                   ;
                           .LOC   $0200
                   ;
0200 20 2F 1E      OPMACH: JSR    CRLF
0203 A2 29                 LDX#   $29         ;SIGN + HEADER COUNT
0205 BD 4F 03      OPMCH1: LDAX   SIGN        ;DISPLAY HEADER
0208 20 A0 1E              JSR    OUTCH
020B CA                    DEX
020C 10 F7                 BPL    OPMCH1
020E A5 01                 LDA    SIGNIF+1
0210 20 3B 1E              JSR    PRTBYT      ;DISPLAY SIGNIF HI
0213 A5 00                 LDA    SIGNIF
0215 20 3B 1E              JSR    PRTBYT      ;DISPLAY SIGNIF LO
0218 20 2F 1E              JSR    CRLF
021B 20 3B 03              JSR    BASPNT      ;POINTERS=BASES
```

```
021E A5 03       DO1:     LDA   ABASE+1
0220 C5 07                CMP   AMAX+1
0222 30 09                BMI   IF1         ;BR IF WHOLE JOB NOT DONE
0224 A5 02                LDA   ABASE
0226 C5 06                CMP   AMAX
0228 30 03                BMI   IF1         ;BR IF WHOLE JOB NOT DONE
022A 4C B7 02             JMP   ENDO1       ;HERE IF WHOLE JOB DONE
022D A2 00       IF1:     LDX#  0           ;DOES CURRENT PAIR MATCH•
022F A1 0A                LDAX@ APOINT
0231 C1 0C                CMPX@ BPOINT
0233 D0 64                BNE   ELS1        ;BR IF NOT THE SAME
0235 86 12       THEN1:   STX   COUNT       ;HERE ON SAME
0237 86 13                STX   COUNT+1     ;CLEAR THE COUNTER
0239 A2 03                LDX#  3
023B B5 0A       THN1A:   LDAX  APOINT      ;SAVES=POINTERS
023D 95 0E                STAX  ASAVE
023F CA                   DEX
0240 10 F9                BPL   THN1A
0242 A2 00       DO2:     LDX#  0           ;DO TILL NOT THE SAME
0244 A1 0A                LDAX@ APOINT
0246 C1 0C                CMPX@ BPOINT
0248 D0 26                BNE   ENDO2       ;BR IF NOT THE SAME
024A A5 0B                LDA   APOINT+1
024C C5 07                CMP   AMAX+1
024E 30 06                BMI   EXP21       ;BR IF LESS THAN
0250 A5 0A                LDA   APOINT
0252 C5 06                CMP   AMAX
0254 10 1A                BPL   ENDO2       ;BR TO ENDO
0256 A5 0D       EXP21:   LDA   BPOINT+1
0258 C5 09                CMP   BMAX+1
025A 30 06                BMI   EXP22       ;BR IF LESS THAN
025C A5 0C                LDA   BPOINT
025E C5 08                CMP   BMAX
0260 10 0E                BPL   ENDO2       ;BR TO ENDO IF LIMIT REACHED
0262 20 BA 02    EXP22:   JSR   AINC        ;MOVE A POINTER TO NEXT A OPCODE
0265 20 CE 02             JSR   BINC        ;MOVE B POINTER TO NEXT B OPCODE
0268 E6 12                INC   COUNT
026A D0 D6                BNE   DO2
026C E6 13                INC   COUNT+1
026E D0 D2                BNE   DO2         ;BR ALWAYS TO TOP OF DO
0270 EA          ENDO2:   NOP               ;A WASTED BYTE FOR "STRUCTURE"
0271 A5 13       IF2:     LDA   COUNT+1
0273 C5 01                CMP   SIGNIF+1
0275 30 0F                BMI   ELS2        ;BR IF NOT SIGNIF
0277 A5 12                LDA   COUNT
0279 C5 00                CMP   SIGNIF
027B 30 09                BMI   ELS2
027D 20 FE 02    THEN2:   JSR   REPORT      ;HERE ON SIGNIF, OUTPUT RESULT
0280 20 45 03             JSR   PNTBAS      ;POINTERS=BASES
0283 4C 96 02             JMP   ENDIF2
0286 A2 01       ELS2:    LDX#  1
0288 20 3D 03             JSR   BASPT1      ;APOINT=ABASE
028B A5 10                LDA   BSAVE
028D 85 0C                STA   BPOINT
028F A5 11                LDA   BSAVE+1
0291 85 0D                STA   BPOINT+1
0293 20 CE 02             JSR   BINC
0296 4C 9C 02    ENDIF2:  JMP   ENDIF1
0299 20 CE 02    ELS1:    JSR   BINC        ;
029C EA          ENDIF1:  NOP               ;ANOTHER SOP TO "STRUCTURE"
029D A5 0D       IF3:     LDA   BPOINT+1
029F C5 09                CMP   BMAX+1
02A1 30 11                BMI   ENDIF3      ;BR IF NOT DONE
02A3 A5 0C                LDA   BPOINT
02A5 C5 08                CMP   BMAX
02A7 30 0B                BMI   ENDIF3      ;BR IF NOT DONE
02A9 20 3B 03    THEN3:   JSR   BASPNT
02AC 20 BA 02             JSR   AINC
02AF A2 01                LDX#  1
02B1 20 47 03             JSR   PNTBS1
02B4 4C 1E 02    ENDIF3:  JMP   DO1
02B7 4C 4F 1C    ENDO1:   JMP   START
```

```
                    ;
                    ;        SUBROUTINES FOLLOW
                    ;
                    ;        MOVE TO NEXT A OPCODE
02BA A2 00          AINC:    LDX#   0
02BC A1 0A                   LDAX@ APOINT      ;GET OPCODE
02BE 20 E2 02                JSR    BYTCNT     ;CALCULATE SIZE
02C1 8A                      TXA               ;RESULT RETURNED IN X
02C2 18                      CLC
02C3 65 0A                   ADC    APOINT     ;ADD RESULT TO POINTER
02C5 85 0A                   STA    APOINT
02C7 A5 0B                   LDA    APOINT+1
02C9 69 00                   ADC#   0
02CB 85 0B                   STA    APOINT+1
02CD 60                      RTS
                    ;
                    ;        MOVE TO NEXT B OPCODE
02CE A2 00          BINC:    LDX#   0
02D0 A1 0C                   LDAX@ BPOINT      ;GET OPCODE
02D2 20 E2 02                JSR    BYTCNT     ;CALCULATE SIZE
02D5 8A                      TXA               ;RESULT RETURNED IN X
02D6 18                      CLC
02D7 65 0C                   ADC    BPOINT     ;ADD RESULT TO POINTER
02D9 85 0C                   STA    BPOINT
02DB A5 0D                   LDA    BPOINT+1
02DD 69 00                   ADC#   0
02DF 85 0D                   STA    BPOINT+1
02E1 60                      RTS
                    ;
                    ;        CALCULATE SIZE OF OPERAND (+1)
                    ;        BY H. T. GORDON (SEE DDJ #22. P.5)
02E2 A2 01          BYTCNT:  LDX#   1
02E4 2C E8 02                BIT    BYTCNT+6   ;TEST BIT 3
02E7 D0 08                   BNE    HAFOP      ;ALL X(8-F)
02E9 C9 20                   CMP#   $20
02EB F0 0E                   BEQ    THREE      ;ONLY $20
02ED 29 9F                   AND#   $9F        ;BITS 5,6 OUT
02EF D0 0B                   BNE    TWO        ;ALL EXCEPT (0,4,6)0
02F1 29 15          HAFOP:   AND#   $15        ;RETAINS ONLY BITS 0,2,4
02F3 C9 01                   CMP#   1
02F5 F0 05                   BEQ    TWO        ;X(9,B)
02F7 29 05                   AND#   5          ;BIT 4 OUT
02F9 F0 02                   BEQ    ONE        ;X(8,A) AND (0,A,6)0
02FB E8             THREE:   INX               ;RESID. X(9-F)
02FC E8             TWO:     INX
02FD 60             ONE:     RTS
                    ;
                    ;        DISPLAY SIGNIFICANT SEQUENCE LIMITS
02FE A2 01          REPORT:  LDX#   1
0300 B5 12          REPT1:   LDAX   COUNT      ;OUTPUT EXTENT OF MATCH
0302 20 3B 1E                JSR    PRTBYT
0305 CA                      DEX
0306 10 F8                   BPL    REPT1
                    ;        OUTPUT MULTIPLE SPACES
0308 20 31 03                JSR    OUTSP4     ;FOUR SPACES
030B A2 00                   LDX#   0
030D B5 0F          REPT2:   LDAX   ASAVE+1    ;OUTPUT START AND
030F 20 3B 1E                JSR    PRTBYT     ; END ADDR OF
0312 B5 0E                   LDAX   ASAVE      ; BOTH SEGMENTS
0314 20 3B 1E                JSR    PRTBYT
0317 20 34 03                JSR    OUTSP2
031A B5 0B                   LDAX   APOINT+1
031C 20 3B 1E                JSR    PRTBYT
031F B5 0A                   LDAX   APOINT
0321 20 3B 1E                JSR    PRTBYT
0324 20 31 03                JSR    OUTSP4
0327 E8                      INX
0328 E8                      INX
0329 E0 03                   CPX#   3
032B 30 E0                   BMI    REPT2
032D 20 2F 1E                JSR    CRLF
0330 60                      RTS
```

```
                  ;
0331 20 34 03     OUTSP4: JSR   OUTSP2      ;4 SPACES
0334 20 9E 1E     OUTSP2  JSR   OUTSP       ;2 SPACES
0337 20 9E 1E             JSR   OUTSP
033A 60                   RTS
                  ;
                  ;       MOVE ABASE & BBASE TO APOINT & BPOINT
033B A2 03        BASPNT: LDX#  3
033D B5 02        BASPT1  LDAX  ABASE
033F 95 0A                STAX  APOINT
0341 CA                   DEX
0342 10 F9                BPL   BASPT1
0344 60                   RTS
                  ;
                  ;       MOVE APOINT & BPOINT TO ABASE & BBASE
0345 A2 03        PNTBAS: LDX#  3
0347 B5 0A        PNTBS1: LDAX  APOINT
0349 95 02                STAX  ABASE
034B CA                   DEX
034C 10 F9                BPL   PNTBS1
034E 60                   RTS
                  ;
                  SIGN:   .ASCII            ' = FINGIS  '
034F 20
0350 3D
0351 20
0352 46
0353 49
0354 4E
0355 47
0356 49
0357 53
0358 20
0359 20

                  HEADER: .ASCII            'OT   MORF     OT   MORF    EZIS
035A 4F
035B 54
035C 20
035D 20
035E 20
035F 4D
0360 4F
0361 52
0362 46
0363 20
0364 20
0365 20
0366 20
0367 20
0368 4F
0369 54
036A 20
036B 20
036C 20
036D 4D
036E 4F
036F 52
0370 46
0371 20
0372 20
0373 20
0374 20
0375 45
0376 5A
0377 49
0378 53
                  ;
                          .END
```

```
0379
0000 SIGNIF       02BA AINC
0002 ABASE        02CE BINC
0004 BBASE        0271 IF2
0006 AMAX         0286 ELS2
0008 BMAX         02BA AINC
000A APOINT       02CE BINC
000C BPOINT       0271 IF2
000E ASAVE        0286 ELS2
0010 BSAVE        027D THEN2
0012 COUNT        02FE REPORT
1C4F START        0345 PNTBAS
1E2F CRLF         0296 ENDIF2
1EA0 OUTCH        033D BASPT1
1E3B PRTBYT       029C ENDIF1
1E9E OUTSP        029D IF3
0200 OPMACH       02B4 ENDIF3
0205 OPMCH1       02A9 THEN3
034F SIGN         0347 PNTBS1
033B BASPNT       02E2 BYTCNT
021E DO1          02F1 HAFOP
022D IF1          02FB THREE
02B7 ENDO1        02FC TWO
0299 ELS1         02FD ONE
0235 THEN1        0300 REPT1
023B THN1A        0331 OUTSP4
0242 DO2          030D REPT2
0270 ENDO2        0334 OUTSP2
0256 EXP21        035A HEADER
0262 EXP22
```

| | SIZE | FROM | TO | FROM | TO | SIGNIF = 0006 |
|---|---|---|---|---|---|---|
| | 0026 | 2000 | 2052 | 4000 | 4052 | |
| x | 0007 | 2069 | 207B | 4093 | 40A5 | |
| x | 0006 | 2099 | 20A5 | 42C2 | 42CE | |
| x | 0006 | 2224 | 2234 | 437C | 438C | |
| x | 000A | 2237 | 224D | 4784 | 479A | |
| x | 000B | 274E | 2761 | 479D | 47B0 | |
| x | 0008 | 279D | 27AC | 47BB | 47CA | |
| | 007A | 28D1 | 29BE | 47CF | 48BC | |
| | 0008 | 29BF | 29D1 | 48BC | 48CE | |
| | 0019 | 29DB | 2A0D | 48CE | 4900 | |
| | 004D | 2A17 | 2AC6 | 492D | 49DC | |
| | 002E | 2ACB | 2B33 | 49E1 | 4A49 | |
| | 0035 | 2B6E | 2BE5 | 4A49 | 4AC0 | |
| | 000C | 2BF2 | 2C04 | 4ACD | 4ADF | |
| | 0106 | 2CE2 | 2F01 | 4B27 | 4D46 | |

Note:
items tagged with an 'x' represent false matches.

| SIZE | FROM | TO | FROM | TO | SIGNIF = 0008 |
|---|---|---|---|---|---|
| 0026 | 2000 | 2052 | 4000 | 4052 | |
| 003D | 206C | 20F0 | 4052 | 40D6 | |
| 0020 | 20F3 | 213C | 40D6 | 411F | |
| 001F | 213C | 2180 | 4122 | 4166 | |
| 000E | 2187 | 21A7 | 416D | 418D | |
| 0046 | 21AA | 224D | 4198 | 423B | |
| 0087 | 2275 | 2394 | 4258 | 4377 | |
| 0009 | 23A8 | 23BB | 438F | 43A2 | |
| 0126 | 23C0 | 25E6 | 43A2 | 45C8 | |
| 004C | 25F1 | 269F | 45C8 | 4676 | |
| 0087 | 26C1 | 27C1 | 4692 | 4792 | |
| 000E | 27C8 | 27E2 | 479D | 47B7 | |
| 000C | 27E5 | 27F9 | 47BB | 47CF | |
| 007A | 28D1 | 29BE | 47CF | 48BC | |
| 0008 | 29BF | 29D1 | 48BC | 48CE | |
| 0019 | 29DB | 2A0D | 48CE | 4900 | |
| 004D | 2A17 | 2AC6 | 492D | 49DC | |
| 002E | 2ACB | 2B33 | 49E1 | 4A49 | |
| 0035 | 2B6E | 2BE5 | 4A49 | 4AC0 | |
| 000C | 2BF2 | 2C04 | 4ACD | 4ADF | |
| 0087 | 2DE5 | 2F01 | 4C2A | 4D46 | |

| SIZE | FROM | TO | FROM | TO | SIGNIF = 000A |
|---|---|---|---|---|---|
| 0026 | 2000 | 2052 | 4000 | 4052 | |
| 003D | 206C | 20F0 | 4052 | 40D6 | |
| 0020 | 20F3 | 213C | 40D6 | 411F | |
| 001F | 213C | 2180 | 4122 | 4166 | |
| 000E | 2187 | 21A7 | 416D | 418D | |
| 0046 | 21AA | 224D | 4198 | 423B | |
| 0089 | 2271 | 2394 | 4254 | 4377 | |
| 0126 | 23C0 | 25E6 | 43A2 | 45C8 | |
| 004C | 25F1 | 269F | 45C8 | 4676 | |
| 0089 | 26BC | 27C1 | 468D | 4792 | |
| 000E | 27C8 | 27E2 | 479D | 47B7 | |
| 000C | 27E5 | 27F9 | 47BB | 47CF | |
| 007A | 28D1 | 29BE | 47CF | 48BC | |
| 001D | 29D1 | 2A0D | 48C4 | 4900 | |
| 004D | 2A17 | 2AC6 | 492D | 49DC | |
| 002E | 2ACB | 2B33 | 49E1 | 4A49 | |
| 0035 | 2B6E | 2BE5 | 4A49 | 4AC0 | |
| 000C | 2BF2 | 2C04 | 4ACD | 4ADF | |
| 0089 | 2DE1 | 2F01 | 4C26 | 4D46 | |

# A MEMORY TEST PROGRAM FOR THE COMMODORE PET

Michael J. McCann
28 Ravenswood Terrace
Cheektowaga, NY 14225

It would be useful and convenient to be able to test PET's memory with a testing program rather than sending the machine back to Commodore for service. Towards this end I have written a memory test program in Commodore BASIC for the PET. The program is well commented, and should be self documenting. (see listing)

Since the program occupies the lowest 4K of PET's memory, use of the program will require that the lowest 4K of memory be operating normally. The amount of time required to run this program rapidly increases as the number of bytes under test is increased (see Figure 1.)

Testing large blocks of memory results in more rigorous testing at the expense of time. Therefore, when using this program the user will have to make a decision regarding rigor vs. time. As a bare minimum, I would suggest testing 100 bytes at a time.

In closing I would suggest that you get this program up and running before you have a problem. It may prove difficult to get a new program working when you have a major system problem.

```
10 REM MEMORY TEST PROGRAM FOR THE COMMODORE PET
20 REM PROGRAM WILL RUN ON 8K PET
30 REM BY MICHAEL J MCCANN
40 PRINT CHR$(147):EE=0:I=0
50 INPUT "START ADDRESS"; SA
60 IF SA<4097 OR SA>65535 GOTO 50
70 INPUT "STOP ADDRESS"; SP
80 IF ST>65535 OR SP<SA GOTO 70
90 PRINT CHR$(147):PRINT:PRINT
100 PRINT TAB(5)"WORKING"
105 PRINT:PRINT"FAULT IN ADDRESS:";
110 REM MEMORY ACCESS AND LOGIC CIRCUITRY TEST
120 REM WRITE ALL 0
130 FOR A=SA TO SP
140 POKE A,0
150 NEXT
160 REM CHECK FOR CORRECTNESS (=0)
170 FOR A=SA TO SP
180 IF PEEK(A)<>0 THEN EE=1:GOSUB 800
190 NEXT
200 REM WRITE ALL 255
210 FOR A=SA TO SP
220 POKE A,255
230 NEXT
240 REM CHECK FOR CORRECTNESS(=255)
250 FOR A=SA TO SP
260 IF PEEK(A)<>255 THEN EE=1:GOSUB 800
270 NEXT
280 REM BEAT TESTS
290 REM WRITE ALL 0
300 FOR A=SA TO SP
310 POKE A,0
320 NEXT
330 REM BEAT ONE ADDRESS WITH 255
335 AD=SA+I
340 POKE AD,255
350 POKE AD,255
360 POKE AD,255
370 POKE AD,255
380 POKE AD,255
390 REM CHECK ALL FOR 0 EXCEPT THE ADDRESS
    BEAT WITH 255
400 FOR A=SA TO SP
410 IF A=AD GOTO 430
420 IF PEEK(A)<>0 THEN EE=1:GOSUB 800
430 NEXT
440 IF AD=SP+1 THEN POKE AD,0: I=I+1: GOTO 335
450 I=0
460 REM WRITE ALL 255
470 FOR A=SA TO SP
480 POKE A,255
490 NEXT
500 REM BEAT ONE ADDRESS WITH 0
505 AD=SA+I
510 POKE AD,0
520 POKE AD,0
530 POKE AD,0
540 POKE AD,0
550 POKE AD,0
560 REM CHECK ALL FOR 255 EXCEPT THE ADDRESS
    BEAT WITH 0
570 FOR A=SA TO SP
580 IF A=AD GOTO 600
590 IF PEEK(A)<>255 THEN EE=1:GOSUB 800
600 NEXT
610 IF AD<>SP+1 THEN I=I+1:POKE AD,255:GOTO 505
620 REM ADDRESSING TEST
630 REM WRITE CONSECUTIVE INTEGERS (0-255) IN
    ALL LOCATIONS UNDER TEST
640 I=0
650 FOR A=SA TO SP
660 IF I=256 THEN I=0
670 POKE A,I
680 I=I+1
690 NEXT
700 REM CHECK FOR CORRECTNESS
705 I=0
710 FOR A=SA TO SP
720 IF I=256 THEN I=0
730 IF PEEK(A)<>I THEN EE=1:GOSUB 800
740 I=I+1
750 NEXT
760 PRINT
770 IF EE=0 THEN PRINT" NO MEMORY PROBLEMS DE-
    TECTED"
780 END
800 PRINT A;
810 RETURN
```

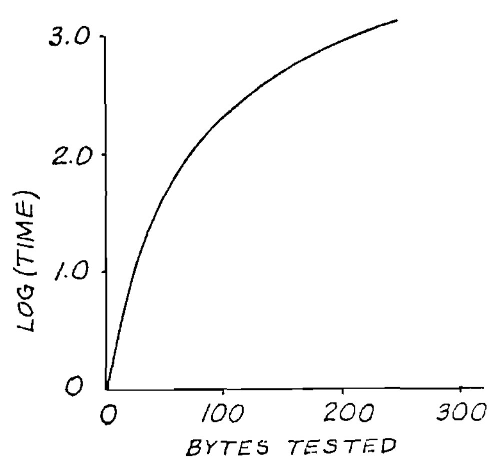



Figure 1. Graph of Log(Time Required) vs. Number of Bytes Tested. (Time in Seconds)

# MICROBES, A SUGGESTION, AND AN APOLOGY

## MICROBES

Ah, how often it is the things in life which appear so simple that cause us great anguish and gnashing of teeth. We present here what we hope is the last microbe in "A KIM Beeper" 4:43:

*The beeper (MICRO 5:24) still doesn't beep - it only clicks! This results from the EOR, of address 010D, operating on two <u>identical</u> operands except for the first iteration in each "beep." This results in a zero being stored in PBD, i.e., no toggling.*

*The low-order bit of A should be set before <u>each</u> EOR. But, more simply, EOR PBD, STA PBD may be replaced by INC PBD (and 3 NOPs, to preserve the branch).*

*The latter change is tested and beeping in the background.*

*Regards,*
*Randy Graves*

Even "Apple Pi" isn't simple any more! Neil D. Lipson of the Philadelphia Apple Users Group writes that "The Pi article by Bob Bishop (MICRO 6:15) is missing one thing. Add HIMEM:4096." But, that's not all! John Paladini writes that: "The value of Pi was not computed to 1000 decimal places, but rather 998. Such inaccuracies occur when computing a series where billions of calculations are required. My best guess is that in order to calculate Pi to 1,000 places using the given series one would have to compute to 1,004 places. The last two digits should read 89 not 96."

Although we made special efforts to make the McCann article "A Simple 6502 Assembler for the PET" error free, including careful proofing by us and the author, a couple of microbes slipped through. C. E. White and David Hustvedt wrote about the following problems:

1. After entering the program from the keyboard your must save it on tape before going through "RUN" again. If you don't EN and ZZ are set to zero.

2. Errors in the typed listing are:

```
1040   HX$+SX$ ...          S/B  HX$=SX$
4030   ;MN$(1B);...         S/B  ;MN$(IB);
5020   ;TAB(27) OP          S/B  ;TAB(27);OP
6060   ...NULL,0,NULL,0     S/B  three NULL,0's
6100   DATA CLC,1,...       S/B  DATA CLI,1,
6120   ..JMI,3,...          S/B  ...JMPI,3,
6250   ...CPX,2,...         S/B  ...CPXZ,2,
14350   GOTO 14380          S/B  GOTO 14480
```

3. When using the "BRK" command the system outputs the error statement "ILLEGAL QUANTITY ERROR IN 10020", READY.

## A SUGGESTION

We finally heard from an OSI owner. John Sheffield writes that the BASIC Disassembler for Apple and PET by McCann (MICRO 5:25) can work on an OSI Challenger IIP with only a small change: "In each line where BY% appears (lines 10, 30, 3050) just change it to BY and everything works fine. Change to read like this:

```
10 DIM MN$(256),BY(256),CO$(16)
30 READ MN$(E),BY(E)
delete line 100
3050 ON BY(IB) GOTO 3060,3090,4050
```

That's all that is needed. By the way that program works on IIP's with 8K of RAM or more."
I would be lead to believe that the BASIC Assembler would work with similar modifications.

John Sheffield had a "p.s." on his letter which said "don't let the IIP be buried under all the Apples and PETs". The staff of MICRO would love to publish material about the OSI products, if only we had some to print! In our first year we received only two articles about OSI. The first was one we "leaned on" a friend for when MICRO was just starting and needed material. The second was a scathing blast at OSI from top to bottom by an obviously disgruntled customer! We do not publish strongly negative material on the basis of a single input, and therefore this article was not published. If there are OSI owners with something to share, MICRO will be most happy to hear from you and print your info.

## AN APOLOGY

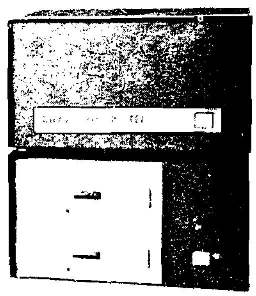
One of the trade marks of MICRO has been quality. We have made a great effort to obtain good articles and to present them in a high quality publication. We must therefore apologize for the printing quality of MICRO number 6. By the time we got the material back from the printer, who had done a reasonably good job on issues number 4 and 5, it was too late to do anything about the inferior quality of the product except to throw out obviously bad copies. We have gotten some letters and calls from readers who received incomplete or unreadable copies. If you have such a problem, please notify us by mail indicating which pages were defective, and we will promptly replace them.

We apologize for the poor quality of issue 6. We have changed printers starting with this issue, and hope that the quality will be better.

# COMPUTER SHOP

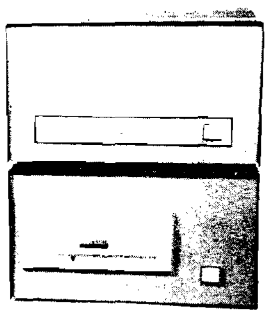
288 NORFOLK ST. CAMBRIDGE, MASS. 02139
corner of Hampshire & Norfolk St. 617-661-2670

## NOW WE HAVE OSI



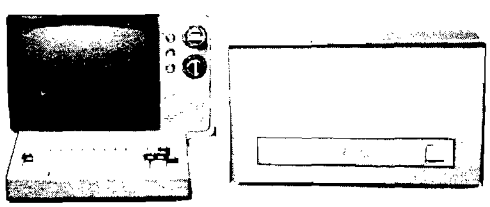
**C3-S1 Challenger III System with Dual Drive Floppy $3,590.00**
Complete with 32K RAM Memory, Dual Drive Floppy, Serial Port, cabinets and power supplies. This Challenger III features an eight slot heavy-duty main frame. You add only a serial ASCII Terminal.

**C2-S2S 32K RAM Serial Challenger II with Dual Drive Floppy $3,090.00**
Comes complete with 32K RAM Memory, Dual Drive Floppy Disk (500,000 characters storage), 6502 processor and serial port. You add only a serial ASCII Terminal to be up and running.



**C2-S1S Serial Challenger II with Single Drive Floppy $1,990.00**
Comes complete with 16K RAM Memory, Single Drive Floppy Disk (250,000 characters storage), 6502 processor and serial port. You need to add only Serial ASCII Terminal.

**C2-S1V Video Challenger II with Single Drive Floppy $2,490.00**
Comes complete with 16K RAM Memory, Single Drive Floppy Disk, 6502 processor, Challenger IIP type Video Interface and high quality keyboard. You add only a Video Monitor (or RF generator and tv set).



**C2-8P Challenger IIP with 8 Slot Cabinet $825.00**
Offers all features of the Challenger IIP plus more room for expansion. The keyboard has a separate case with connector cable. The roomy cabinet and heavy duty power supply are designed to handle up to eight system boards (allowing for 6 slots of expansion).


**C2-4P Challenger IIP $598.00**

## KIMS AND UPGRADES

| | |
|---|---|
| VF8 4K Memory assembled & tested | 129.00 |
| for low power RAM add | 10.00 |
| same in kit form | 74.50 |
| full set of sockets for Kit | 10.00 |
| VF8 Motherboard buffered for 4 Boards | 65.00 |
| Connector Assembly for KIM to VF8 | 20.00 |
| 8K S100 Memory Board with instructions.K | 165.00 |
| same but fully assembled and tested | 199.00 |
| CS100 Cabinet cut out for KIM | 129.00 |
| 3 Connector S100 Motherboard Assembly | 75.00 |
| CGRS S100 TIM Kit | 129.00 |
| CGRS S100 6502 CPU Kit | 179.00 |
| CGRS S100 Front Panel Kit | 129.00 |
| XITEX Video Terminal Board 16X64K | 155.00 |
| XITEX Video Terminal Board Assembled | 185.00 |
| KIM-1 | 245.00 |
| CS100 with CGRS, Xitex, 16KRAM, TV, KB | 1529.00 |
| Same but Assembled | 1989.00 |
| PS-5 Pwr Supp. 5V5A9V1A-12V1A6x6X2 | 75.00 |
| PS-5 Assembled | 90.00 |
| Total of Order..Circle Items wanted | $............ |
| Mass. Residents Sales Tax 5% | $............ |
| Shipping, 1%($2.00 min.) | $............ |
| Total Remittance or Charge | $............ |

BAC, VISA, MC NO ..............................

SIGNATURE ..............................

NAME ..............................

ADDRESS ..............................

CITY .............. STATE ......... ZIP .....

# THE MICRO SOFTWARE CATALOG: IV

Mike Rowe
P.O. Box 3
So. Chelmsford, MA 01824

Name: Bridge Challenger
System: PET or Apple II
Memory: 8K PET or 16K Apple II
Language: Not specified
Hardware: Not specified
Description: Bridge Challenger lets you and the dummy play four person Contract Bridge against the computer. The program will deal hands at random or according to your criterion for high card points, and you can save hands on cassette and reload them for later play. You can review tricks, rotate hands East-West, shuffle only the defense hands, or replay hands when the cards are known.
Copies: Not specified
Price: $14.95
Includes: Not specified
Author: Not specified
Available from:
Personal Software
P.O. Box 136
Cambridge, MA 02138
617/783-0694

Name: CURSOR - Programs for PET Computers
System: PET
Memory: 8K
Language: BASIC and Assembly Language
Hardware: Standard PET
Description: CURSOR is a cassette magazine with proven programs written just for the 8K PET. Each month the subscriber receives a C-30 cassette with five or more high quality programs for the PET. People can't read this "magnetic magazine", but the PET can! The CURSOR staff includes professional programmers who design and write many of the programs. They also carefully edit programs which are purchased from individual authors.
Copies: Not specified
Price: $24 for 12 monthly issues
Includes: Cassette
Authors: Many and varied
Available from:
Ron Jeffries, Publisher
CURSOR
P.O. Box 550
Goleta, CA 93017
805/967-0905

Name: PET Schematics and PET ROM Routines
System: PET
Memory: None
Language: None
Hardware: None
Description: PET Schmatics is a very complete set of accurately and painstakingly drawn schematics about your PET. It includes a 24" x 30" CPU board, plus oversized drawings of the Video Monitor and Tape Recorder, plus complete Parts layout - all the things you hoped to get from Commodore, but didn't!
PET ROM Routines are complete assembly listings of all 7 ROMs, plus identified subroutine entry points.
Copies: Not specified.
Price: PET Schematics - $34.95
PET ROM Routines - $19.95
Available from:
PET-SHACK Software House
Marketing and Research Co.
P.O. Box 966
Mishawaka, IN 46544

Name: S-C Assembler II
System: Apple II
Memory: 8K
Language: Assembly language
Hardware: Apple II, optional printer
Description: Combined text editor and assembler carefully integrated with the Apple II ROM-based routines. Editor inclues full Apple II screen editing, BASIC-like line-number editing, tab stops, and renumbering. LOAD, SAVE, and APPEND commands for cassette storage. Standard Apple II syntax for opcodes and address modes. Labels (1 to 4 characters), arithmetic expressions, and comments. English language error messages. Monitor commands directly available within assembler. Speed and suspension control over listing and assembly.
Copies: Just released, over 100 sold.
Price: $20.00 (Texas residents add 5% tax)
Includes: Cassette in Apple II format and a 28 page reference manual.
Author: Bob Sander-Cederlof
Available from:
S-C Software
P.O. Box 5537
Richardson, TX 75080

Name: PL/65 or CSL/65
System: SYSTEM 65 or PDP 11
Memory: 16K bytes RAM
Language: Machine language.
Hardware: Rockwell SYSTEM 65
Description: A high-level language resembling PL/1 and ALGOL is now available to designers developing programs for the 6500 microprocessor family using either the SYSTEM 65 development system of the PDP 11 computer. PL/65 is considerably easier to use than assembly language or object code. The PL/65 compiler outputs source code to the SYSTEM 65's resident assembler. This permits enhancing or debugging at the assembler level before object code is generated. In addition, PL/65 statements may be mixed with assembly language instructions for timing or code optimization.
Copies: Not specified.
Price: Not specified from Rockwell.
$500 from COMPAS.
Includes: Minifloppy diskette.
Authors: Not specified.
Available from:
Electronic Devices Division
Rockwell Internationsal
P.O. Box 3669
Anaheim, CA 92803
714/632-2321 (Leo Scanlon)
213/386-8776 (Dan Schlosky)

COMPAS - Computer Applications Corp.
413 Kellogg
P.O. Box 687
Ames, IA 50010
515/232-8181 (Michael R. Corder)

Name: PRO-CAL I
System: PET
Memory: Not specified.
Language: BASIC and machine language.
Hardware: Not specified.
Description: A reverse polish scientific calculator program, ideally suited for scientific and educational applications. Supports single key execution of more than 50 forward and inverse arithmetic, algebraic, trigonometric and exponential functions. It implements calculations in binary, octal, decimal, and hexidecimal modes with single keystroke conversion between modes and simultaneous decimal equivalen display.It also allows the recording and playback of calculator programs on cassette tape permitting the use of most calculator software already in existance up to a limit of 255 steps.
Copies: Not specified.
Price: $26.00 domestic, $28.00 foreign.
Includes: Software on cassette and an operating manual.
Authors: Not specified.
Available from:
Applications Research Co.
13460 Robleda Road
Los Altos Hills, CA 94022

Name: MICROCHESS
Systems: PET and Apple II
Memory: PET - 8K/Apple II 16K
Language: 6502 Machine Language
Hardware: Standard PET or Apple II
Description: MICROCHESS is the culmination of two years of chessplaying program development by Peter Jennings, author of the famous 1K byte chess program for the KIM-1. MICROCHESS offers eight levels of play to suit everyone from the beginner learning chess to the serious player. It examines positions as many as 6 moves ahead, and includes a chess clock for tournament play. Every move is checked for legality and the current position is display on a graphic chessboard. You can play White or Black, set up and play from special board positions, or even watch the computer play against itself.
Copies: Not specified.
Price: $19.95
Includes: Not specified.
Author: Peter Jennings
Available from:
Personal Software
P.O. Box 136
Cambridge, MA 02138
617/783-0694

Name: Financial Software
System: Apple II (easily modified for PET)
Language: Applesoft II
Hardware: Apple II
Description: Sophisticated financial programs used to aid in investment analysis. The following programs are currently available: Black-Scholes Option Analysis, Security Analysis using the Capital Asset Pricing Model, Bond Pricing I and II, Cash Flow and Present Value Analysis I and II, Stock Valuation, Rates of Return, Calculations and Mortgage Analysis.
Copies: Just released.
Price: $15.00 each or $50.00 for all 9 programs
Includes: Cassette, annotated source listings, operating and modifying instructions, sample runs and backgroud information.
Author: Eric Rosenfeld
Available from:
Eric Rosenfeld
70 Lancaster Road
Arlington, MA 02174

Name: Apple II BASEBALL
System: Apple II
Memory: 16K or more
Language: Integer BASIC
Hardware: Standard Apple II
Description: An interactive baseball game that uses color graphics extensively. You can play a 7 or 9 inning game with a friend, (it will handle extra innings), or play alone against the computer. Has sound effects with men running bases. Keeps track of team runs, hits, innings, balls and strikes, outs, batter-up and uses paddle input to interact with the game. Uses every available byte of memory.
Copies: Just released.
(Dealers inquiries invited)
Price: $12.50
Includes; Game Cassette, User Bookelt with complete BASIC listing.
Authors: Pat Chirichella and Annette Nappi
Available from:
Pat Chirichella
506 Fairview Avenue
Ridgewood, NY 11237

Name: DDT-65 Dynamic Debugging Tool
System: Any 6502 based system
Memory: 3K RAM/1K RAM for loader
Language: Machine Language
Hardware: 32 char/line terminal
Description: DDT-65 is an advanced debugger that allows easy assembly and disassembly in 650X mnemonics. Software single-stepping and automatic breakpoint insertion/deletion allow debuffing of code even in PROM. DDT-65 comes in a relocatable form on tape for loading into any memory or for PROM programming.
Copies: 11+
Price: $25.00
Include: 10 page manual, relocating tape cassette.
Ordering Info: KIM format cassette - K
Kansas City at 300 baud for OSI - O
Kansas City at 300 baud for TIM/JOLT - T
Author: Rich Challen
Available from:
Rich Challen
939 Indian Ridge Drive
Lynchburg, VA 24502

# APPLE CALLS AND HEX-DECIMAL CONVERSION

Marc Schwartz
220 Everit Street
New Haven, CT 06511

Rich Auricchio's "Programmer's Guide to the Apple II" (MICRO #4, April/May 1978) is a very useful step in getting out printed materials to help users fully exploit the Apple's potential. That his table of monitor routines can be used in BASIC programming is worth noting.

Many monitor routines can be accessed in BASIC by CALL commands addressed to the location of the first step of the routine. If the routine is located in hex locations 0000 to 4000, it is necessary only to convert the hex location to decimal and write CALL before the decimal number. Thus a routine located at hex 1E would be accessed by the command: CALL 30, since hex 001E = decimal 30.

If you do not have a hex-decimal conversion table handy, you can convert larger numbers to decimal with the help of the Apple by the following steps:

1. Start in BASIC (necessary for step 2)
2. Multiply the first (of four) hex digits by 4096, the second by 256, the third by 16 and the fourth by one. Add the four numbers to get the decimal equivalent. For example, to get the decimal conversion of 03E7, with the Apple in BASIC, press Control/C and type

```
>PRINT 0*4096 + 3*256 + 14*16 + 7
```

then press RETURN. You'll get your decimal answer: 839. To begin a monitor routine you wrote starting at 03E7, merely put CALL 839 in your program.

If the hex location of the routine is between C000 and FFFF, then another method of figuring out the corresponding decimal location must be used.

1. Start in BASIC
2. Press the RESET button.
3. Take the hex location of the routine and subtract if from FFFF. The Apple will help you do this; subtract each pair of hex digits from FF and press RETURN. The Apple will print the answer to each subtraction for you. For example the hex location of the routine to home cursor and clear screen is $FC58.

```
* FF - FC RETURN
  = 03
* FF - 58 RETURN
  = A7
```

So, $FFFF - $FC58 = $03A7.

Now convert to decimal as above, using BASIC (control/C) to assist you.

```
>PRINT 0*4096 + 3*256 + 10*16 + 7
```

and after pressing RETURN you will have your answer, 935.

4. Add one to the total, here giving 936.
5. Make the new total negative, or -936.
6. That's it. Now just put a CALL in front of the number: CALL -936.

Of course, these steps of converting hex locations to decimal are the same ones to take if you want to access the PEEK or POKE functions of the Apple. In all, they allow the BASIC programmer to take much fuller advantage of the capabilities of the computer.

And while on the subject of hex-decimal conversion, the Apple can help in decimal to hex conversion as well. For example to find the hex of a number, say 8765:

1. Start in BASIC

2. Divide the number by 4096, then find the remainder:

```
>PRINT 8765/4096,8765MOD4096 (return)
 2      573
```

3. Repeat the process with 256 and16:

```
>PRINT 573/256,573MOD256 (return)
 2      61
>PRINT 61/16, 61 MOD 16 (return)
 3       13
```

...giving 2 2 3 13 or 223C.

## WRITING FOR MICRO

One of the reasons we like the 6502 is that it seems to attract a lot of very interesting, active, enthusiastic users. We spend several hours each week talking to people who are so excited about what they are doing with their system that they just have to talk to someone. Oh, sometimes they pretend they have some "burning" question or whant to order some small item, but really they mostly want to tell someone about all of the fun they are having or the discoveries they are making.

While we enjoy these conversations, and consider them one of the "Fringe benefits" of editing MICRO, it disturbs us that many of these enthusiasts who are willing to spend five to ten dollars on a phone call to us, are not willing to spend a little time writing down their information for publication in MICRO where thousands can share it (and they can earn a few dollars).

MICRO, in order to serve its main purpose of presenting information about all aspects of the 6502 world, needs to receive information from a wide variety of sources. To achieve a more balanced content, we desparately need articles on: industrial, educational, business, home, and other real applications of systems; non-KIM, -APPLE, -PET systems, homebrew and commercial; techniques for programming, interfacing, and expanding systems; and many other topics. Look to your own experience. If you have anything to share, then take the time to write it down. The "Manuscript Cover Sheet" on the next page should serve as a guide and make it a little easier to submit your article.

## MANUSCRIPT COVER SHEET

Please complete all information requested on this cover sheet.

Date Submitted: ______________________________

Proposed Title: ______________________________

Author(s) Name(s): ______________________________

______________________________

Mailing Address: ______________________________
(This will be published.)

______________________________

Area Code: _____ Phone: ____________________
(This will NOT be published.)

AUTHOR'S DECLARATION OF OWNERSHIP OF MANUSCRIPT RIGHTS: This manuscript is my/our original work and is not currently owned or being considered for publication by another publisher and has not been previously published in whole or in part in any other publication. I/we have written permission from the legal owner(s) to use any illustrations, photographs, or other source material appearing in this manuscript which is not my/our property. If required, the manuscript has been cleared for publication by my/our employer(s). Note any exceptions to the above (such as material has been published in a club newsletter but you still retain ownership) here:

Signature(s): ______________________________

______________________________

Date: ____________________

Any material which you are paid for by The COMPUTERIST, whether or not it is published in MICRO, becomes the exclusive property of The COMPUTERIST, with all rights reserved.

### A Few Suggestions

All text material will be retyped. Therefore your format does not matter as long as it is readable. Double spaced, typed, is preferable, but not required. Any figures should be neatly drawn to scale as they will appear in MICRO. If we have to redraw the figures and diagrams, then we normally will pay less for that page. Photographs should be glossy prints either the same size as the final will be or twice the final size. We will re-assemble all programs to obtain clean listings using the syntax we have adopted (see inside back cover - MICRO #1). Since others will be copying your code, please try to thoroughly test it and make sure it is as error free as possible. Submit your articles early. We will try to get a proof back to you for final correction, but with our tight schedule this may not always be possible. Send your manuscripts to:

Robert M. Tripp, Editor, MICRO, P.O. Box 3, So. Chelmsford, MA 01824, U.S.A.

**6502 BIBLIOGRAPHY**
**PART VI**

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

361. Bridge, Theodore E. "High Speed Cassette I/O for the KIM-1", DDJ 3 Issue 6 No 26, Pg 24-25, (June/July, 1978). Will load or dump at 12 times the speed of KIM-1. Supplements the MICRO-ADE Editor-Assembler.

362. Baker, Robert "KIMER: A KIM-1 Timer", Byte 3 No 7 Pg 12, (July, 1978). The program converts the KIM-1 into a 24-hr digital clock.

363. Conley, David M. "Roulette on Your PET with Bells and Whistles", Personal Computing 2 No 7 Pg 22-24 (July, 1978). How to add extras in a program for added interest.

364. KIM-1/6502 User Notes, Issue 11, (May, 1978)
- Lewart, Cass R. "An LED Provides Visual Indication of Tape Input". An LED allows you to see that the tape recorder is feeding proper signals to KIM.
- Rehnke, E. "Hardware Comparison". The editor compares KIMSI vs. KIM-4 as expansion for KIM.
- Rehnke, E. "Software Comparison". The editor compares the MOS Technology Assembler/Editor from ARESCO versus the MICRO-ADE Assembler/Disassembler/Editor from Peter Jennings, Toronto.
- Edwards, Lew "Skeet Shoot, with Sound". Butterfield's "Skeet Shoot" modified with the Kushnier's phaser sound routine, for KIM.
- DeJong, Marvin "Digital Cardiotachtometer". KIM counts heartbeats per minute and displays count while measuring next pulse period.
- Rehnke, E. "Book review: 'Programming a Microcomputer: 6502'". Foster Caxton's recent book is highly recommended.
- Coppola, Vince "Loan Program in FOCAL". FOCAL-65 is used to figure interest on a loan.
- Flacco, Roy "Joystick Interface". A joystick, some hardware, are used to put the Lunar Lander (First Book of KIM) on the face of a Scope.
- Kurtz, Bob "Morse Code Reader Program". Use KIM in the hamshack.
- Zuber, Jim "Interfacing the SWTPC PR-40 Printer to KIM-1". An easy way to use this low cost printer.
- Nelis, Jody "Revision to Battleship Game". Modification to correct a small defect in the original program.

365. People's Computers 7 No 1 (July/Aug, 1978).
- Cole, Phyllis "SPOT". Several notes and tips of interest to PET owners.
- Cole, Phyllis "Tape Talk". Notes on problems associated with tape I/O on the PET.
- Gash, Philip "PLOT". Program plots any single-valued function y(x) on a grid.
- Julin, Randall "Video Mixer". A circuit to mix the three video signals put out by the PET's IEEE 488-bus.
- Bueck/Jenkins "PETting a DIABLO". How to make PET write using a Diablo daisy wheel printer.

366. Harr, Robt. Jr. and Poss, Gary F. "TV Pattern Generator", Interface Age 3 Issue 8 Pg 80-82; 160, (Aug, 1978). Pattern generator in graphics for the Apple II monitor.

367. Personal Computing 2 No 8 (Aug, 1978).
- Maloof, Darryl M. "PET Strings" (letter to Editor). Note on changing a character string to numeric values and vice-versa.
- Connors, Bob "PET Strings" (letter to Editor). More on changing character strings to numeric values.
- Bueck/Jenkins "Talking PET" (letter to the Editor). Notes on the interfacing of a Diablo daisy wheel printer with PET through the PET ADA device.

368. Lasher, Dana "The Kalculating KIM-1", 73 Magazine, No 215 Pg 100-104 (Aug, 1978). Calculator versatility for any KIM is provided by interfacing a calculator chip and a scanning routine with KIM.

369. OSI-Small Systems Journal 2 No 2 (Mar/Apr, 1978).
- Anon. "The 542 Polled Keyboard Interface". Polled keyboards have many advantages over standard ASCII keyboards.
- Anon. "Basic and Machine Code Interfaces". This is the first in a series of articles on BASIC and machine code.
- Anon. "Using the Model 22 OKIDATA Printer". A quick and dirty way to use those special font and scroll commands of the Model 22 OKIDATA Printer.

370. Dr. Dobbs Journal 3 Issue 7 No 27 (Aug, 1978).
Moser, Carl "Fast Cassette Interface for the 6502". Record and load at 1600 baud.
Meyer, Bennett "Yet Another 6502 Disassembler Fix". Changes to correct a number of errors in the five digit codes used for deciphering the instructions in the BASIC language disassembler published earlier in DDJ 3 No 1.
Anon. "Apple Users Can Access Dow Jones Information Service". With a telephone link-up, Apple II users can dial Dow Jones Information Service.

371. Kilobaud Issue 21 (Sept, 1978).
Wells, Ralph "Trouble Shooters' Corner". Another chapter in the saga of the compatibility of the Apple II with a VIA/PIA. See EDN May 20,1978; MICRO Issue 5, Pg 18, June/July, 1978.
Tenny, Ralph "Troubleshooters' Guide". Useful suggestions for those tackling repair and interfacing problems.
Young, George "Do-It-All Expansion Board for KIM". How to make an expansion board, expansion power supply, new enclosure, etc., for your KIM-1.
Ketchum, Don "KIM Organ". Play tunes directly from the KIM keyboard.
Grina, James "Super Cheap 2708 Programmer". An easy-to-build PROM programmer driven by the KIM-1.

372. Conway, John "Glitches Can Turn Your Simple Interface Task into a Nightmare". Difficulties in using an Apple II with a PIA in an I/O interface, apparently caused by a clock signal arriving a little early.

373. Notley, M. Garth "Plugging the KIM-2 Gap". Byte 3 No 9 Pg 123 (Sept, 1978). How to map the KIM-1 address range of 0400 to 13FF into a KIM-2 address range of 1000 to 1FFF.

374. Turner, Bill and Warren, Carl "How to Load Floppy ROM No 5", Interface Age 3 No 9 Pg 60-61 (Sept, 1978). Side No 1 is in Apple II format at 1200 baud, "The Automated Dress Pattern".

375. Smith, Wm. V.R. III "The Automated Dress Pattern for the Apple II". Interface Age 3 No 9 Pg 76-81 (Sept, 1978). A McCalls pattern is the basis for the program and accompanying Floppy ROM.

376. MICRO Issue 6 (Aug/Sept, 1978).
Husbands, Charles R. "Design of a PET/TTY Interface". Describes the hardware interface and software to use the ASR 33 Teletype as a printing facility for the PET.
Faraday, Michael "Shaping Up Your Apple". Information on using Apple II's High Resolution Graphics.
Eliason, Andrew H. "Apple II Starwars Theme". Disassembler listing of theme from Star Wars.
Bishop, Robert J. "Apple PI". How to calculate PI to 1000 places on your Apple II.
McCann, Michael J. "A Simple 6502 Assembler for the PET". Learn to use Machine language with this assembler.
Rowe, Mike "The Micro Software Catalog: III". Software listing for 6502 systems.
Gaspar, Albert "A Debugging Aid for the KIM-1". A program designed to assist the user in debugging and manipulating programs.
DeJong, Marvin L. "6502 Interfacing for Beginners: Address Decoding II". Good tutorial article.
Suitor, Richard F. "Brown and White and Colored All Over". Discussion of the colors in the Apple and their relation to each other and the color numbers.
Witt, James R. "Programming a Micro-Computer: 6502 by Caxton Foster". More accolades for this fine book.
Merritt, Cal E. "PET Composite Video Output". How to get video output for additional monitors.
Quosig, Karl E. "Power from the PET". How to tap the unregulated 8v and regulate to 5v.
Suitor, Richard F. "Apple Integer BASIC Subroutine Pack and Load". Loading assembly language programs with a BASIC program.
Creighton, Gary A. "A Partial List of PET Scratch Pad Memory". Tabulation of a number of important addresses.

377. Corbett, C. "A Mighty MICROMITE". Personal Computer World 1 No 4 Pg 12 (Aug, 1978). Descriptive article on the KIM-1 for the European and British readers.

378. Coll, John and Sweeten, Charles "Colour is an Apple II". Personal Computing World 1 No 4 Pg 50-55 (Aug, 1978). Description of the Apple II.

379. North, Steve "PET Cassettes from Peninsula School". Creative Computing 4 No 5 Pg 68 (Sept/Oct, 1978). A number of programs written in PILOT, a language designed for CAI dialog applications. This requires a program to interpret PILOT in Basic.

# 6502 INFORMATION RESOURCES

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

Did you ever wonder just what magazines were the richest sources of information on the 6502 microprocessor, 6502-based microcomputers, accessory hardware and software? For several years this writer has been assembling a bibliography 6502 references related to hobbv computers and small business systems (see MICRO No's 1, 3, 4, 5, and 6). A review of the number of times various magazines are cited in the bibliography gives a rough measure of the coverage of these magazines of 6502 related subjects. Even after such a fequency chart is compiled, an accurate comparison is difficult. Some of the magazines have been published longer than others. Some periodicals have been discontinued, others have been merged with continuing publications. Some give a lot of information in the form of ads, others are devoted mostly to authored articles. Regardless of the basis of the tabulation of references, however, some publications are clearly more useful sources of information on the 6502 than others.

The accompanying list of magazines has been compiled from the bibliography. At the top of the list are several publications which specialize in 6502-related subjects. These include this publication, MICRO, as well as the KIM-1 /6502 USER NOTES. Also in this category is OHIO SCIENTIFIC'S SMALL SYSTEMS JOURNAL, a publication which covers hardware and software for the Ohio Scientific 6502-based computers. KILOBAUD, BYTE and DR. DOBB'S JOURNAL all give good coverage on the 6502 as well as other microprocessors. KILOBAUD has more hardware and constructional articles than most computer magazines. ON-LINE is devoted mainly to new product announcements and has very frequent references to 6502 related items. Following these come a group of magazines with somewhat less frequent references to the 6502. Finally toward the end of the list are those magazines with only occasional or trivial references to the 6502. An attempt has been made to give up-to-date addresses and subscription rates for the magazines cited.

MICRO
$6.00 per 6 issues
MICRO
P.O. Box 3
S. Chelmsford, MA 01824

KIM-1/6502 USER NOTES
$5.00 per 6 issues
Eric Rehnke
P.O. Box 33077
Royalton, OH 44133

OHIO SCIENTIFIC--SMALL SYSTEMS JOURNAL
$6.00 per year (6 issues)
Ohio Scientific
1333 S. Chillicothe Rd.
Aurora, OH 44202

KILOBAUD
$15.00 per year
Kilobaud Magazine
Peterborough, NH 03458

BYTE
$12.00 per year
Byte Publications, Inc.
70 Main St.
Peterborough, NH 03458

DR. DOBB'S JOURNAL
$12.00 per year (10 issues)
People's Computer Co.
Box E
1263 El Camino Real
Menlo Park, CA 94025

ON-LINE
$3.75 per year (18 issues)
D. H. Beetle
24695 Santa Cruz Hwy
Los Gatos, CA 95030

PEOPLE'S COMPUTERS (Formerly PCC)
$8.00 per year (6 issues)
People's Computer Co.
1263 El Camino Real
Box E
Menlo Park, CA 94025

INTERFACE AGE
$14.00 per year
McPheters, Wolfe & Jones
16704 Marquardt Ave.
Cerritos, CA 90701

POPULAR ELECTRONICS
$12.00 per year
Popular Electronics
One Park Ave.
New York, NY 10016

PERSONAL COMPUTING (Formerly MICROTREK)
$14.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

73 MAGAZINE
$15.00 per year
73, Inc.
Peterborough, NH

CREATIVE COMPUTING
$15.00 per year
Creative Computing
P.O. Box 789-M
Morristown, NJ 07960

SSSC INTERFACE
(Write for information)
Southern California Computer Soc.
1702 Ashland
Santa Monica, CA 90405

EDN (Electronic Design News)
$25.00 per year
(Write for subscription info)
Cahners Publishing Co.
270 St Paul St.
Denver, CO 80206

RADIO ELECTRONICS
$8.75 per year
   Gernsback Publications, Inc.
   200 Park Ave., South
   New York, NY 10003

QST
$12.00 per year
   American Radio Relay League
   225 Main St.
   Newington, CT 06111

IEEE Computer
(Write for subscription info)
   IEEE
   345 E. 47th St.
   New York, NY 10017

ELECTRONICS
$14.00 per year
   Electronics
   McGraw Hill Bldg.
   1221 Ave. of Americas
   New York, NY 10020

POLYPHONY
$4.00 per year
   PAIA Electronics, Inc.
   1020 W. Wilshire Blvd.
   Oklahoma City, OK 73116

CALCULATORS, COMPUTERS
$12.00 per year (7 issues)
   Dynax
   P.O. Box 310
   Menlo Park, CA 94025

COMPUTER MUSIC JOURNAL
$14.00 per year (6 issues)
   People's Computer Co.
   Box E
   1010 Doyle St.
   Menlo Park, CA 94025

POPULAR COMPUTING
$18.00 per year
   Popular Computing
   Box 272
   Calabasas, CA 91302

MINI-MICRO SYSTEMS
$18.00 per year
   Modern Data Service
   5 Kane Industrial Drive
   Hudson, MA 01749

DIGITAL DESIGN
$20.00 per year
(Write for subscription info)
   Benwill Publishing Corp.
   1050 Commonwealth Ave.
   Boston, MA 02215

ELECTRONIC DESIGN
(26 issues per year)
(Write for subscription info)
   Hayden Publishing Co., Inc
   50 Essex St.
   Rochelle Park, NJ 07662

HAM RADIO
$12.00 per year
   Communications Technology
   Greenville, NH 03048

COMPUTER WORLD
$12.00 per year (trade weekly)
(Write for subscription info)
   Computer World
   797 Washington St.
   Newton, MA 02160

---

Editor's Note: In addition to the magazines regularly covered by the 6502 Bibliography, the following magazines may also be of interest to various 6502 readers:

PET GAZETTE
Free bi-monthly (Contributions Accepted)
   Microcomputer Resource Center
   1929 Northport Drive, Room 6
   Madison, WI 53704

Robert Purser's REFERENCE LIST
   OF COMPUTER CASSETTES
Nov 1978 $2.00/Feb 1979 $4.00
   Robert Purser
   P.O. Box 466
   El Dorado, CA 95623

THE SOFTWARE EXCHANGE
$5.00 per year (6 issues)
   The Software Exchange
   P.O. Box 55056
   Valencia, CA 91355

THE PAPER
$15.00 per year (10 issues)
   The PAPER
   P.O. Box 43
   Audubon, PA 19407

PET USER NOTES
$5.00 per year (6 or more issues)
   PET User Group
   P.O. Box 371
   Montgomeryville, PA 18936

CALL A.P.P.L.E
$10.00 per year (includes dues)
   Apple Puget Sound Program Library Exchar
   6708 39th Ave. SW
   Seattle, WA 98136

# KIM-1 AS A DIGITAL VOLTMETER

Joseph L. Powlette and Charles T. Wright
Hall of Science, Moravian College
Bethlehem, PA 18018

Several programs have been described in the literature which turn a KIM-1 microcomputer into a direct reading frequency counter. In "A Simple Frequency Counter Using the KIM-1" by Charles Husbands (MICRO, No. 3, Pp. 29-32, Feb/Mar,1978) and in "Here's a Way to Turn KIM Into a Frequency Counter" by Joe Laughter (KIM User's Note Issue 3, Jan, 1977), good use is made of KIM-1's interval timers and decimal mode to produce a useful laboratory instrument. A simple change in hardware will allow these same programs to serve as the basis of a direct reading digital voltmeter. This article describes an inexpensive voltage-to-frequency converter (VFC) circuit which is compatible with these programs and also describes some software modifications which will allow Husbands' program to operate down to low frequency (10 HZ) values.

## Hardware Configuration

The VFC circuit is shown in Figure 1. The 4151 chip is manufactured by Raytheon and is available from Active Electronic Sales Corp., P.O. Box 1035, Framingham, MA 01701 for $5.00 or from Jameco Electronics, 1021 Howard Street, San Carlos, CA 94070 for $5.95. The circuit parameters given in Figure 1 have been modified from the values suggested by the manufacturer in order to match the pulse requirement for the KIM IRQ signal. The frequency of the output pulse is proportional to the input voltage and the 1KΩ (multiturn) trimpot is used to adjust the full-scale conversion so that 10 volts corresponds to a frequency of 10 KHz. It is not necessary to calibrate the KIM-1 as a frequency meter since any variation in its timing can be compensated for by the trimpot. A known potential is connected to the VFC input and the trimpot adjusted until the KIM readout agrees with the known voltage value. The linearity of the VFC is better than 1% down to 10 mv (linearity of 0.05% can be achieved in a "precision mode" which is described in the Raytheon literature). The circuit will not respond to negative voltages and protection of the chip is provided by the 1N914 diode. If negative voltage readings are also required, the input to the VFC can be preceded by an absolute value circuit (see IC OP-AMP cookbook by Jung, p. 193, Sams Pub.).

To operate the system using Laughter's software the following connections should be made: 1) the output (pin 3) of the VFC to the PB0 input of KIM (pin 9 on the application connector) and 2) PB7 on the KIM to IRQ on the KIM (A-15 to E-4). Execution of the program should cause the voltage to flash on the KIM display in one second intervals.

The software described in Husbands' article will not operate below 500 Hz. This limit is caused by the fact that the contents of the interval timer are read to determine if the 100 millisecond interval has elapsed and since the interval counter continues to count (at a 1T rate) after the interval has timed out, there are times when the contents of the interval timer are again positive. If the interrupt should sample during this time, the branch on minus instruction will not recognize that the interval has elapsed. This problem will manifest itself as a fluctuating value in the display and is most likely to occur at low frequencies. One solution is to establish the interval timer in the interrupt mode and then allow the program to arbitrate the interrupt, i.e., to determine whether the interrupt was due to the input pulse or the expiration of the 100 millisecond interval timer. The necessary changes to Husbands' program are given in Figure 2. The hardware connections are: 1) output of the VFC (pin 3) to the KIM IRQ (pin 4 on the KIM expansion connector), and 2) PB7 on the KIM to IRQ on the KIM (A-15 to E-4). The modified program starts at 0004 with a clear interrupt instruction. Locations 17FE and 17FF should contain 21 00 and 17FA and 17FB should have values 00 10 (or 00 1C).

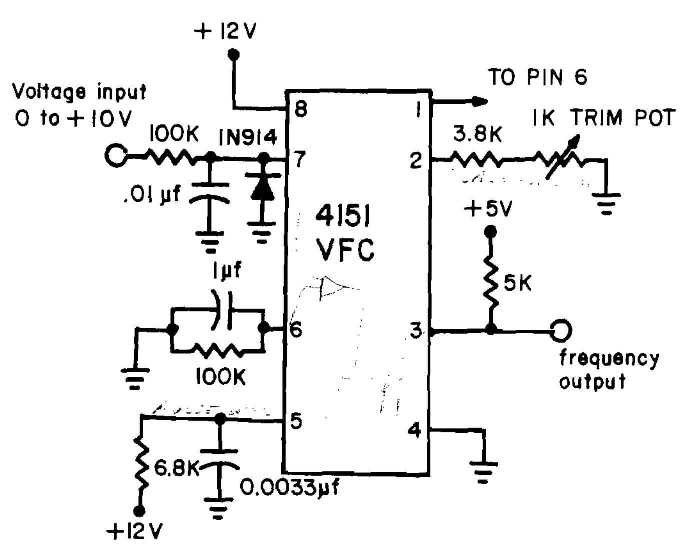



Figure 1. Voltage-to-Frequency Converter (VFC) circuit.

### Additional Comments

The program modifications above will also extend Husbands' frequency counter circuit down to 10 Hz (corresponding to 1 input interrupt in 100 milliseconds). Since the 74121 monostable multivibrator does not have an open collector output, PB7 should not be connected (along with the 74121 output) directly to the KIM IRQ. Two solutions are:

1. Leave PB7 unconnected. The expiration of the 100 millisecond clock will be recognized on the next input interrupt after the timer has timed out. The interval timer will not interrupt the microprocessor, however.

2. Connect PB7 to one input of a two input AND gate and the output of the monostable to the second input. The output of the AND gate should be connected to the KIM IRQ. The expiration of the 100 millisecond interval will now also interrupt the processor and will result in a faster response to a change in frequency values (from high to very low) as well as a more accurate low frequency count.

The authors would like to thank Charles Husbands for taking the time to answer our questions and for pointing out the article by Laughter.

```
                 ORG  $  0004

0004   58        CLI       clear interrupt flag
 .
0014   8D OF 17  STA       clock in interrupt mode
 .
0024   AD 07 17  LDA       read interrupt flag bit 7
 .
003C   8D OF 17  STA       clock in interrupt mode
 .
```

Figure 2. Changes in Husbands' program to extend the low frequency range to 10 Hz.

## HELPING MICRO HELP YOU

MICRO is published for a number of reasons. One very important reason is to provide a means for the distribution of information about 6502 related products. Our advertising rates are very low in relation to our circulation and specialized audience, and we welcome your money, but that is not what we want to discuss here. MICRO offers several ways for you to get good publicity - FREE ! It will take a little work on your part, but the price is right. There are three regular ways to get coverage in MICRO: the software catalog, the hardware catalog, and the list of 6502 related companies.

### THE MICRO SOFTWARE CATALOG

Appearing regularly since issue number 4, the software catalog provides a brief, standardized, description of currently available 6502 software. We were a bit surprized to find that the software catalog was one of the most often mentioned articles in the recent MICRO Reader Feedback. To participate in this catalog, you must follow a few simple rules:

1. The program must be currently available, not "under development".

2. You must provide the write-up following the standard format which is:

   Name of program:
   6502 system(s) it works on:
   Memory required:
   Language used (Assembler, BASIC,...):
   Hardware required:
   Description of program:
   Number of copies in circulation:
   Price:
   Includes: (Cassette, Source listings,...)
   Author:
   Available from:

### THE MICRO HARDWARE CATALOG

In issue number 6 we printed a call for hardware information for a Hardware Catalog. The formats of the material we received was so varied, that we have decided to impose a format for the sake of a more useful presentation of the material. To participate in this catalog, you must follow these rules:

1. The product must be currently available, either in stock or within four weeks delivery on new orders. Some units must have already been successfully delivered.

2. You must provide the write-up following the standard format which is:

   Name of product:
   6502 systems it works with:
   Other hardware required:
   Power requirements:
   Description of product:
   Number of units delivered to date:
   Price:
   Includes: (Manuals, Cables,...)
   Developed by:
   Available from:

A lot of material that has been received for the Catalogs has not been in a useable format. We are not trying to make it difficult for you to submit your material. We are trying to make it easy for the readers to understand your product. We do not understand your product as well as you do and can not therefore do as good a write-up as you can. And, we don't have any more time than you do! So, please submit your stuff in the requested format and we will print it.

### 6502 RELATED COMPANIES

In issue number 1 we printed a list of companies that we were aware of which produced products of interest to the 6502 world. It is time to update the list. If you feel that your company should be on the list, then send in the following information as soon as possible:

Name of company:
Address:
Telephone: (Optional)
Person to contact: (Optional)
Brief list of 6502 products: (Maximum of five typed lines, please)

While the Software and Hardware Catalogs will be appearing regularly in every issue, this list of 6502 Related Companies will only appear once, in issue number 8, the Dec/Jan issue. Therefore, send your information in as soon as possible.

# CASSETTE TAPE CONTROLLER

Fred Miller
7 Templar Way
Parsippany, NJ 07054

The ideal tape storage facility for micro-systems would be one in which the micro has complete control of all tape movement and play/record functions without "operator intervention" e.g. pushing buttons. Unfortunately most of us have budgets which only allow use of lower cost audio cassette units. Short of massive mechanical rebuilding, these units can only be externally controlled with a motor on/off function after the "operator" has set the proper record/play keys. All too often we goof and press the wrong button, have to move cassettes from one unit to another, or simply forget to set up the units at the right time.

The Cassette Tape Controller (CTC) described below offers a reasonably inexpensive capability as a compromise in the provision of automatic tape control for a KIM-1 system. CTC is a combination of a seven-IC hardware board and supporting software routines. It was developed to control two Pioneer Centrex KD-12 cassette units. The concept could be extended to more than two units or perhaps other models.

A summary of the functions provided are:

(1) Provide software-driven capability to start and stop a specific tape recorder by opening/closing the "remote control" circuit of the recorder (normally controlled by a switch on an external microphone).

(2) Provide software-driven capability to route the input (record) or output (playback) signals as appropriate.

(3) Provide override manual controls (toggles) to also accomplish (1) and (2), above.

(4) Light panel indicators (LEDs) associated with the play or record functions selected for each cassette unit as set by software or manual controls.

(5) Sense whether the selected tape recorder is set to play or record, or neither.

(6) Sense the position of auxiliary toggles for setting software options, etc., (option switches.

(7) Light indicators (LEDs) associated with the auxiliary toggles for operator communications.

(8) Provide an audible "beep" under software control.

## CTC General Description

The Cassette Tape Controller is a hardware/software facility to assist in the operation and use of audio cassette tape recorders for data read/write functions. The hardware provides the interface from a KIM-1 to two Pioneer Centrex KD-12 tape recorders. Besides the cassette input and output lines from KIM-1 four other lines (bit ports) are required for software control of the hardware.

The software and hardware control the recorder's motor circuits and determine if the appropriate manual keys on the recorder are set correctly. The software can provide alternative action (alert the operator or try another unit) in the case of improperly set keys.

The specific software illustrated below is written to "search" for a unit which is set in eitha "read" (playback) or "write" (record) mode. If none is found in the desired mode, an audible tone is sounded and the search is continued. The visible indication of each of the "read" or "write" LEDs blinking along with the audible tone provides the operator with a quick clue as to the erroneous settings. If the appropriate tapes are "mounted" the operator simply depresses the "requested" cassette unit key. Subsequent references by the software would locate the preset unit without communicating to the operator.

Additional facilities are built into the CTC hardware/software at little extra cost. These include the separately accessible audible tone and two option toggles with accompanying panel indicator LEDs. The toggles can be used for setting options selected by the operator and tested by the software. The associated indicators can also be used for some optional communication purposes. A third switch (momentary toggle or pushbutton) is used as a "break" command for software testing. A layout of the related hardware control panel is shown in Fig. 1.

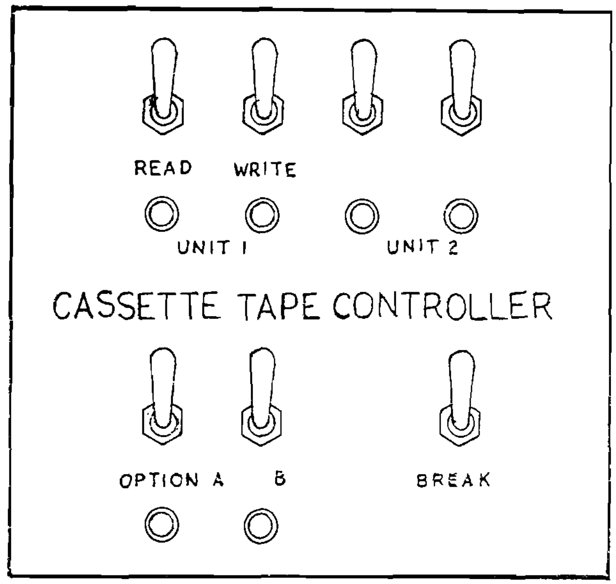



Figure 1.
Suggested Panel Layout
for Cassette Tape Controller

## Hardware Description

A key to the logic of CTC is the ability to sense actual cassette unit key settings. By sensing voltage levels at two externally accessible points in the KD-12 circuitry it is possible to determine one of the following states:

(1) unit set for read (playback) or fast forward or rewind

(2) unit set for write (record)

(3) no keys depressed

The circuit shown in Fig. 2 uses two ICs to address a function, one to enable and the other to sense results of enabling. This logic is further described in the comments accompanying the software source listing. Four non-critical DPDT relays are used to allocate signals and control motor circuits. The additional circuits, (1) pulse an audible tone generator, (2) light LED indicators, or, (3) sense toggle switch positions all depending upon addressed functions.

Three bits (PB 0-2) from KIM-1 Applications Port B are used to address the functions. Another bit line (PB 3) of the same port is used to feed status back to KIM-1.

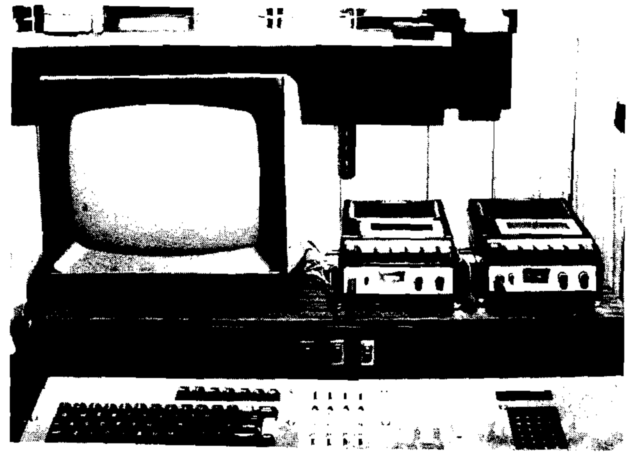
The KD-12 units are operated from external battery power (continually trickle-charged) to provide the most stable unit operation. HYPERTAPE speeds are extremely reliable in this configuration.

### Software Description

The controlling software consists of a series of routines which are accessible from user programs. The software shown in Fig. 3 is designed to "seek out" a cassette unit which is set for a given function, e.g., read. A brief study of the routines will show how this can be replaced or amended to select only a given cassette unit for a specific function. The additional routines are provided for "testing" the optional toggle switches, etc. Many of the routines are useful for other than tape cassette control, e.g., a JSR to BELL provides an audible "beep".

### Conclusion

The hardware and software described have been working very satisfactorily on the author's system for well over a year. The CTC software (along with tape and record I/O routines based on the HYPERTAPE routines) have been committed to EPROM (2708). Access to this capability is easy and provides convenient operation of tape file processing from user software programmed in any language used on the KIM-1 micro (BASIC, Assembler, HELP, etc.). Although the operator still must press the keys on the cassette units, the CTC system can save many a "rerun" or clobbered files due to careless operations.



Author's KIM Based System

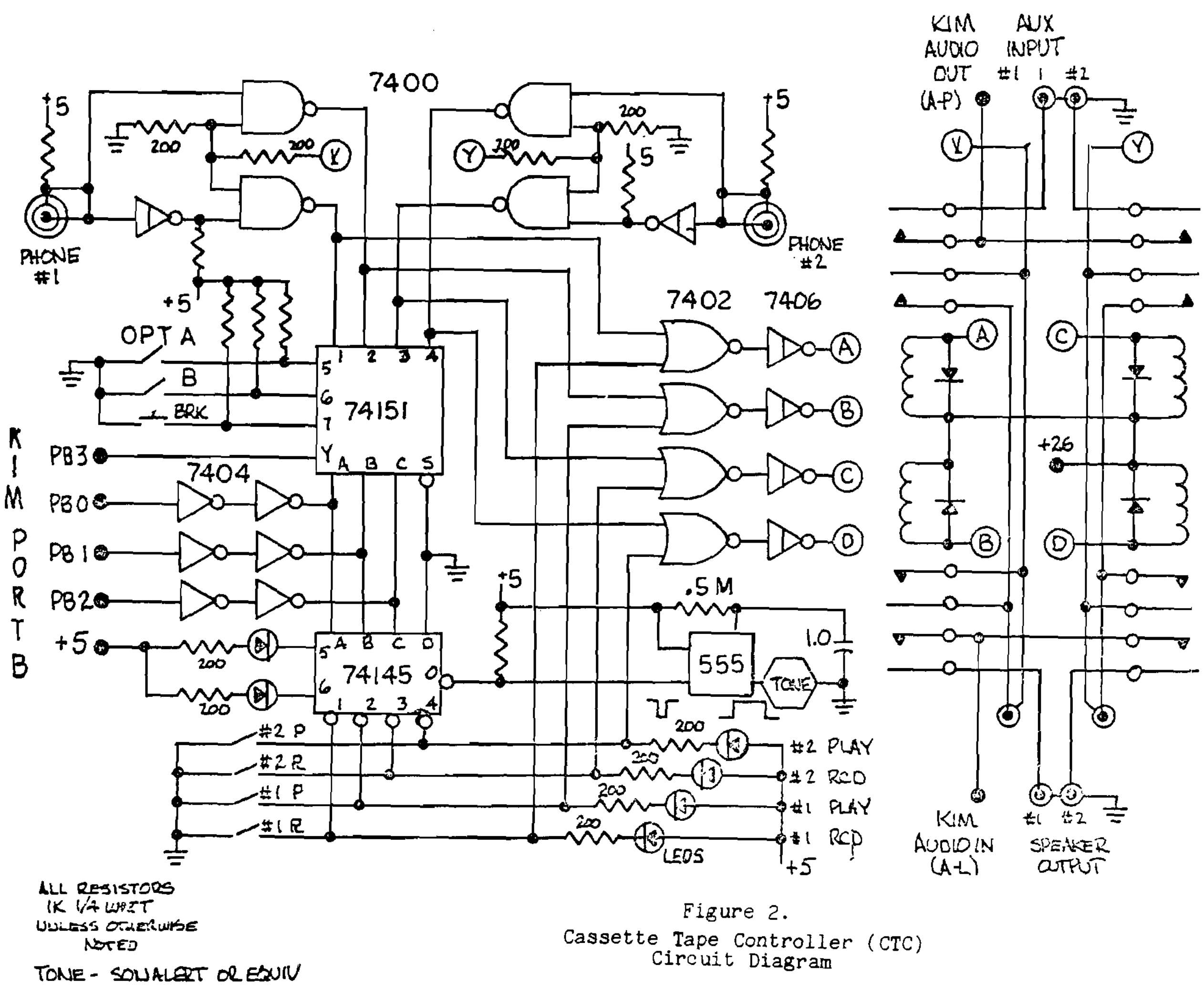



Figure 2.
Cassette Tape Controller (CTC)
Circuit Diagram

```
0200               KXFTAP ORG   $0200

                   *******************
                   *                 *
                   *   CASSETTE TAPE *
                   * CONTROLLER (CTC) *
                   *    BY R.MILLER   *
                   *                 *
                   *******************

                   *** KIM & ZERO PAGE PARAMETERS ***

0200               PBD    *     $1702
0200               PBDD   *     $1703
0200               TPFCT  *     $00EF
0200               INIT   *     $1E8C
ID=02

                   *** TAPE CASSETTE READ ROUTINES ***

0200 D8            RDTAPE CLD
0201 A9 02                LDAIM $02    TEST FOR UNIT#1 READY
0203 20 1B 02             JSR   TPTEST FOR READ?
0206 F0 0C                BEQ   CREAD  ...YES
0208 A9 04                LDAIM $04    ...NO, UNIT#2 READY?
020A 20 1B 02             JSR   TPTEST
020D F0 05                BEQ   CREAD  ...YES
020F 20 2B 02             JSR   BELL   ...NO, SOUND SIGNAL AND
0212 D0 EC                BNE   RDTAPE TRY AGAIN.

0214 EA            CREAD  NOP
                   .
                   .
                   . ROUTINE FOR READING TAPE
                   . GOES HERE
                   .
                   .

0215 20 33 02             JSR   CTLOFF TURN OFF CASSETTE MOTOR
0218 4C 8C 1E      RDEXIT JMP   INIT   AND RETURN VIA KIM INIT
ID=03

                   *** CASSETTE SUPPORT RTNS ***

021B 85 EF         TPTEST STA   TPFCT  SAVE UNIT/FCT
021D 8D 02 17             STA   PBD    PORT B CONTROL DATA
0220 20 3C 02             JSR   DELAY  ALLOW RELAY SETTLE
0223 AD 02 17             LDA   PBD    CK BITS 0-3 = TO
0226 29 0F                ANDIM $0F    ORIGINAL UNIT/FCT
0228 C5 EF                CMP   TPFCT
022A 60                   RTS          EQUAL MEANS UNIT READY

022B A9 00         BELL   LDAIM $00
022D 8D 02 17             STA   PBD    ZERO FCT SETS TONE
0230 20 3C 02             JSR   DELAY  WAIT, RESET & EXIT

0233 A9 07         CTLOFF LDAIM $07    BITS 0-2 TO O/P
0235 8D 03 17             STA   PBDD
0238 8D 02 17             STA   PBD    SET TO FCT#7 (OFF)
023B 60                   RTS
```

```
0190:
0200: 023C A9 FF      DELAY  LDAIM $FF
0210: 023E 8D 07 17          STA   $1707  SET TIMER TO 1/4 SEC
0220: 0241 2C 07 17          BIT   $1707
0230: 0244 10 FB             BPL   DELAY  +05
0240: 0246 60                RTS
0250:
0260: 0247 20 33 02   BRKCK  JSR   CTLOFF ENSURE OFF
0270: 024A 18                CLC
0280: 024B AD 02 17          LDA   PBD
0290: 024E 29 08             ANDIM $08    BIT 3 HIGH MEANS NO BRK
0300: 0250 D0 01             BNE   BKEXIT
0310: 0252 38                SEC
0320: 0253 60         BKEXIT RTS          NO CARRY MEANS NO BRK
ID=04

0010:
0020:                 *** CASSETTE WRITE ROUTINE ***
0030:
0040: 0254 D8         WRTAPE CLD
0050: 0255 A9 01             LDAIM $01    TEST FOR UNIT#1 READY
0060: 0257 20 1B 02          JSR   TPTEST FOR WRITE?
0070: 025A F0 0C             BEQ   CWRITE ...YES
0080: 025C A9 03             LDAIM $03    ...NO, TEST UNIT#2
0090: 025E 20 1B 02          JSR   TPTEST
0100: 0261 F0 05             BEQ   CWRITE ...YES
0110: 0263 20 2B 02          JSR   BELL   ...NO, SOUND SIGNAL AND TRY
0120: 0266 D0 EC             BNE   WRTAPE AGAIN
0130:
0140: 0268 EA         CWRITE NOP
0150:                 .
0160:                 .
0170:                 . CASSETTE WRITE ROUTINE
0180:                 . GOES HERE
0190:                 .
0200:                 .
0210: 0269 20 33 02          JSR   CTLOFF TURN OFF MOTORS
0220: 026C 4C 8C 1E          JMP   INIT   AND RETURN VIA KIM
ID=05

0010:                 *** ALT.SW TEST & LIGHT ***
0020:
0030: 026F A9 06      TSTSWA LDAIM $06    SET FOR ALT.SW #1
0040: 0271 D0 02             BNE   TSTSWB +02
0050:
0060: 0273 A9 05      TSTSWB LDAIM $05    SET FOR ALT.SW #2
0070: 0275 48                PHA          SAVE CODE
0080: 0276 20 33 02          JSR   CTLOFF INITL PORTS
0090: 0279 68                PLA          RETRIEVE CODE
0100: 027A 20 1B 02          JSR   TPTEST AND TEST SW
0110: 027D 18                CLC
0120: 027E D0 01             BNE   TSTX   IF NOT EQUAL
0130: 0280 38                SEC          MEANS SW IS NOT SET
0140: 0281 4C 33 02   TSTX   JMP   CTLOFF CARRY MEANS SW 'ON'
ID=
```

# APPLE II HIGH RESOLUTION GRAPHICS
## MEMORY ORGANIZATION

Andrew H. Eliason
28 Charles Lane
Falmouth, MA 02540

One of the most interesting, though neglected, features fo the Apple II computer is its ability to plot on the television screen in a high resolution mode. In this mode, the computer can plot lines, points and shapes on the TV display area in greater detail than is possible in the color graphics mode (GR) which has a resolution of 40 x 48 maximum.

In the high resolution (HIRES) mode, the computer can plot to any point within a display area 280 points wide and 192 points high. While this resolution may not seem impressive to those who have used plotters and displays capable of plotting hundreds of units per inch, it is nonetheless capable of producing a very complex graphic presentation. This may be easily visualized by considering that a full screen display of 24 lines of 40 characters is "plotted" at the same resolution. An excellent example of the HIRES capability is included in current Apple II advertisements.

Why, then, has reletively little software appeared that uses the HIRES features? One of the reasons may be that little information has been available regarding the structure and placement of words in memory which are interpreted by HIRES hardware. Information essential to the user who wishes to augment the Apple HIRES routines with his own, or to explore the plotting possibilities directly from BASIC. In a fit of curiosity and Apple-insomnia, I have PEEKed and POKEd around in the HIRES memory area. The following is a summary of my findings. Happy plotting!

Each page of HIRES Graphics Memory contains 8192 bytes. Seven bits of each byte are used to indicate a single screen position per bit in a matrix of 280H x 192V. The eighth bit of each byte is not used in HIRES and the last eight bytes of every 128 are not used.

The bits in each byte and the bytes in each group are plotted in ascending order in the following manner. First consider the first two bytes of page 1. (Page 2 is available only in machines with at least 24K).

```
BYTE           |  8192          |   8193          |
SCREEN
POSITION        0 1 2 3 4 5 6 7 8 9 10 11 12 13
BIT             0 1 2 3 4 5 6 0 1 2  3  4  5  6
               |V G V G V G|V|G V G  V  G  V |G|
                            |_|               |_|
     (Bit 7 not used)        7                 7

                                V = VIOLET
                                G = GREEN
```

Figure 1 represents the screen position and respective bit & word positions for the first 14 plot positions of the first horizontal line. If the bit is set to 1 then the color within the block will be plotted at the position indicated. If the bit is zero, then black will be plotted at the indicated position. It can be seen that even bits in even bytes plot violet, even bits in odd bytes plot green and vice versa. Thus all even horizontal positions plot violet and all odd horizontal politions plot green. To plot a single white point, one must plot the next higher or lower horizontal position along with the point, so that the additive color produced is white. This is also true when plotting single vertical lines.

The memory organization for HIRES is, for design and programming considerations, as follows:

Starting at the first word, the first 40 bytes (0-39) represent the top line of the screen (40 bytes x 7 bits = 280). The next 40 bytes, however, represent the 65th line (i.e., vertical position 64). The next 40 bytes represent the line at position 128 and the next 8 bytes are ignored. The next group of 128 bytes represent three lines at positions 8, 72 and 136, the next group at positions 16, 80 and 142, and so on until 1024 bytes have been used. The next 1024 bytes represent the line starting at vertical position 1 (second line down) in the same manner. Eight groups of 1024 represent the entire screen. The following simple porgram provides a good graphic presentation as an aid to understanding the above description. Note that there is no need to load the HIRES machine language routines with this program. Set HIMEM:8191 before you type in the program.

```
100 REM  SET HIMEM:8191
110 REM  HIRES GRAPHICS LEARNING AID
120 POKE -16304,0: REM  SET GRAPHICS MODE
130 POKE -16297,0: REM  SET HIRES MODE
140 REM   CLEAR PAGE - TAKES 20 SECONDS
150 FOR I=8192 TO 16383: POKE I,0: NEXT I
160 INPUT "ENTER BYTE (1 to 127)", BYTE
170 POKE -16302,0: REM  CLEAR MIXED GRAPHICS
180 FOR J=8192 TO 16383: REM  ADDRESS'
190 POKE J,BYTE: REM  DEPOSIT BYTE IN ADDRESS
200 NEXT J
210 POKE -16301,0: REM  SET MIXED GRAPHICS
220 GOTO 160
999 END
```

An understanding of the above, along with the following equations will allow you to supplement the HIRES graphics routines for memory efficient programming of such things as: target games, 3D plot with hidden line supression and 3D rotation, simulation of the low resolution C=SCRN (X,Y) function, etc. Also, you may want to do some clever programming to put Flags, etc., in the unused 8128 bits and 512 bytes of memory!

HI RES Graphics Equations and Algorithms

Where:

```
FB = ADDRESS OF FIRST BYTE OF PAGE.
     PAGE1 = 8192   PAGE 2 = 16384
LH = HORIZONTAL PLOT COORDINATE. 0 TO 279
LV = VERTICAL PLOT COORDINATE. 0 TO 191
BV = ADDRESS OF FIRST BYTE IN THE LINE OF
     40
BY = ADDRESS OF THE BYTE WITHIN THE LINE
     AT BV
BI = VALUE OF THE BIT WITHIN THE BYTE
     WHICH CORRESPONDS TO THE EXACT POINT
     TO BE PLOTTED.
```

Given: FB,LH,LV

```
BV = LV MOD 8 * 1024 + (LV/8) MOD 8 * 128
     + (LN/64) * 40 + FB
BY = LH/7 + BV
BI = 2^(LH MOD 7)
```

To Plot a Point (Without HIRES Plot Routine):

```
LH = X MOD 280 : LV = Y MOD 192 (OR)
                 LV = 192-Y MOD 192
FB = 8192
BV = LV MOD 8 * 1024 + (LV/8) MOD 8 * 128 +
     (LV/64) * 40 + FB
BY = LH/7 + BV
BI = 2^(LH MOD 7)
WO = PEEK (BY)
IF (WO/BI) MOD 2 THEN (LINE NUMBER + 2)
POKE BY, BI + WO
RETURN
```

To Remove a Point, Substitute:

```
IF (WO/BI) MOD 2 = 0 THEN (LINE NUMBER + 2)
POKE BY, WO-BI
```

To Test a Point for Validity, the Statement:

"IF (WO/BI) MOD 2" IS TRUE FOR A PLOTTED POINT AND FALSE (=0) FOR A NON PLOTTED POINT.

---

## RIVERSIDE ELECTRONIC DESIGN'S KEM AND MVM-1024:

## A USER'S EVALUATION

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

The price and availability of a variety of memory and application boards for the S 100 bus will make many KIM-1 owners think about expandind their systems to be compatible with this bus. The KIM Expansion Module (KEM) does the trick. In addition, one of the most attractive I/O modes is the keyboard/video monitor team. Riverside's MVM-1024, which interfaces neatly with the KEM, provides all the necessary circuitry to provide a 16 line by 64 character display on a video monitor. Programs which give the user a variety of display functions (homing the cursor, backspace, erase-a-line, etc.) and allow the user to communicate with the computer by way of the keyboard are also available from Riverside. Finally, all of the hardware and software is well documented in a series of application notes.

Space does not allow a complete description of all of the packages mentioned above. The reader should obtain the application notes and descriptions from Riverside if he is contemplating expansion. Summarily, the KEM buffers all of the address and data lines from the KIM-1, separating the latter into IN and OUT busses as required by the S 100; provides the necessary memory-mapped I/O ports for the keyboard, cursor, and video display; provides the logic for the S 100 signals; and provides four locations for the 1K 2708 EPROMs, in which may be stored display/monitor programs, PROM programmer software, or your favorite games.

The KEM does all of this without affecting any of the I/O ports on the KIM-1. That is, PAD and PBD may still be accessed from a connector on the KEM. The MVM-1024 contains its own memory and does not use any of the memory on the KIM-1. ASCII from the keyboard is loaded from address 13F8. To display a character, ASCII code for the character is stored in location 13FB. The cursor is controlled by the contents of two locations, 13F9 which contains a six bit word which determines the location of the character in a line, and 13FA which contains a four bit word which determines the line being used. Of course, the display/monitor programs do all of the necessary loading (LDA) and storing (STA) for you, but it is particularly easy to write short programs or subroutines which read the keyboard and/or output data on the video monitor

The danger in writing an equipment evaluation like this is in making it so concise that it is Greek to everyone except the hardened computer addict. So, I will conclude by saying that I was very satisfied with the performance of the Riverside hardware and software. I particularly liked their use of premium components such as LS TTL, the fact that the KIM-1 I/O ports are still available for applications, the keyboard polling software which allows the user to use NMI or IRQ interrupts for applications and the 4K of PROM space. Also, it is much easier to enter and de-bug programs with the display/monitor software. My only criticism is that it is not easy to lay out the system in a small package form.

# A DIGITAL CLOCK PROGRAM FOR THE SYM-1

Chris Sullivan
9 Galsworthy Place
Bucklands Beach
Aukland, New Zealand

The SYM-1 is a one board hobbyist computer similiar to the KIM but with a number of additional features. Since buying the SYM-1 I have had a great deal of fun playing around with both the software and hardware sides of it. The SYM-1 monitor, Supermon, is an incredible monitor in 4K ROM, some of it's subroutines are called by the following program.

This program started off as a lesson in familiarity with the 6502 instruction set and using the Supermon subroutines to advantage, but the present version has been modified many times in order to increase the clock accuracy and, as my knowledge of the 6502 instruction set grows, increase coding efficiency. To use it one should start execution at :200. Then enter an "A" or "P" (Shift ASCII 5 0) to signify AM or PM. Then enter the hours (two digits), the program then outputs a space to separate the hours from the minutes. Finally enter 2 digits to signify the minutes, the program will then increment the minutes by 1, and begin the clock sequence. This slight quirk makes it easier to set the clock using another clock, set up the "A" or "P", hours and first digit of the minutes, then enter the last digit of the minutes as the seconds counter of your setting clock reaches 0.

There is another slight quirk in that the clock counts "All 59", "A12 00", "A12 01", ...., "A12 59", "P01 00","P01 01" ..... This simplifies the programming and means that 12:30 near midday is in fact, 12:30 AM according to this clock! However this is not likely to confuse many people.

After setting up the initial time, the program adds 1 to the minutes and then carries on any carry into the hours, possibly changing "A" to "p" or vice versa. This section of the program could be made more efficient with full exploitation of the 6502 instruction set. The last section in the program is a 1 minute delay. I have rewritten this section many times in a search for an accurate 1 minute delay. The first part is a double loop which also scans the clock display, this loop takes about 59.8 seconds. The second part is a double loop to "tweak" the delay up to 60 seconds and consists of 2 delays using the onboard 6532 timer. This timer is initialised in 1 of 4 memory locations, specifying ÷1024, ÷64, ÷8, or ÷1 timing, e.g., the location to write to if one wants ÷1024 timing is A417. This location thus initialised is counted down in the 6532. The program reads this value until it becomes negative, at which time the delay is over.

Some improvements to the program could be made, for example better coding in the increment minutes section. One could also add an alarm feature, possibly using the on board beeper. The The section to update the time by one minute could be used as a part of a background real time clock, being called by a once-a-minute hardware interupt generated by an on board 6522 timer chip. Once a minute, processing would be interupted for 100 cycles or so in order to update the real time clock. Such clocks have many uses, one of which is to ensure that certain number-crunching programs don't get tied down in big loops.

This improved version occupies less RAM by using jumps to INBYTE rather than INCHAR and messy bit manipulations. The delay routine has been improved to use the on board 6532 timer, and also give greater resolution and hence greater timing accuracy.

Editor's Note: This program is present primarily for its value in showing how to access the SYM's monitor for some of the routines. It is not an "optimal" program for a 24 hour clock, but should be a good starting point for owners of SYMs who wish to write similar programs.

```
                    SYM-1 ELECTRONIC CLOCK

                    BY CHRIS SULLIVAN  AUGUST 27, 1978

                           ORG   $0200

                    SPACE  *     $0020  ASCII SPACE
                    ACCESS *     $8B86
                    INCHAR *     $8A1B
                    INBYTE *     $81D9
                    OUTCHR *     $8A47
                    OUTBYT *     $82FA

0200 20 86 8B  BEGIN  JSR   ACCESS
0203 20 1B 8A         JSR   INCHAR GET A OR P
0206 85 00            STAZ  $00
0208 18               CLC
0209 20 D9 81         JSR   INBYTE GET HOURS
020C 85 01            STAZ  $01
020E A9 20            LDAIM SPACE  SPACE CHARACTER
0210 20 47 8A         JSR   OUTCHR OUTPUT A SPACE
0213 20 D9 81         JSR   INBYTE GET MINUTES
0216 85 02            STAZ  $02
0218 F8               SED          SET DECIMAL MODE FOR REMAINDER OF PROGRAM
```

```
                    HAVING SET THE INITIAL TIME (LESS 1 MINUTE)
                    UPDATE THE TIME:

0219 18           TIMLOP CLC
021A A5 02               LDAZ  $02    GET MINUTES
021C 69 01               ADCIM $01    INCREMENT
021E 85 02               STAZ  $02
0220 38                  SEC
0221 E9 60               SBCIM $60    TEST IF NEW HOUR
0223 F0 03               BEQ   TIMEX
0225 4C 50 02            JMP   NORSET IF NOT A NEW HOUR

0228 A9 00        TIMEX  LDAIM $00
022A 85 02               STAZ  $02    SET MINUTES TO 00
022C 18                  CLC
022D A5 01               LDAZ  $01
022F 69 01               ADCIM $01    INCR HOURS
0231 85 01               STAZ  $01
0233 38                  SEC
0234 E9 13               SBCIM $13    TEST HOURS = 13
0236 F0 03               BEQ   TIMEY
0238 4C 50 02            JMP   NORSET

023B A9 01        TIMEY  LDAIM $01    YES, SET HOURS TO 1
023D 85 01               STAZ  $01
023F A5 00               LDAZ  $00    GET A OR P
0241 49 50               EORIM $50    ASCII P
0243 F0 07               BEQ   TIMEZ  IS 00 = ASCII P?
0245 A9 50               LDAIM $50    NO, THEN SET 00 TO P
0247 85 00               STAZ  $00
0249 4C 50 02            JMP   NORSET
024C A9 41        TIMEZ  LDAIM $41    YES, THEN SET 00 TO A
024E 85 00               STAZ  $00

0250 A5 00        NORSET LDAZ  $00    GET A OR P
0252 20 47 8A            JSR   OUTCHR
0255 A5 01               LDAZ  $01    GET HOURS
0257 20 FA 82            JSR   OUTBYT
025A A9 20               LDAIM SPACE
025C 20 47 8A            JSR   OUTCHR
025F A5 02               LDAZ  $02    GET MINUTES
0261 20 FA 82            JSR   OUTBYT
0264 D8                  CLD          CLEAR DECIMAL MODE
0265 A2 C0               LDXIM $C0    SETUP FOR ALMOST 60 SEC WAIT
0267 A0 7D        WAITA  LDYIM $7D    COUNTER
0269 A9 01        WAITB  LDAIM $01    NON-DISPLAYING CHARACTER
026B 20 47 8A            JSR   OUTCHR REFRESH DISPLAY
026E 88                  DEY
026F D0 F8               BNE   WAITB  LOW ORDER COUNTER
0271 CA                  DEX          HIGH ORDER COUNTER
0272 D0 F3               BNE   WAITA
0274 A2 02               LDXIM $02    TWEAK TIME UP TO 60 SECONDS
0276 A9 4D        WAITC  LDAIM $4D
0278 8D 17 A4            STA   $A417  DIVIDE BY 1024 TIMER
027B AD 06 A4     WAITD  LDA   $A406  REGISTER OF 6532
027E 10 FB               BPL   WAITD
0280 CA                  DEX
0281 D0 F3               BNE   WAITC
0283 F8                  SED
0284 4C 19 02            JMP   TIMLOP
                  VERIFY from 0200 thru 0286 is 356F.
```

The following subroutines called form part of the SYM-1's SUPERMON monitor:

ACCESS Enables the user program to write to system RAM, i.e. the RAM contained on the 6532. It is necessary to call ACCESS before calling most of the other system subroutines.

INCHAR Get one ASCII charcter from the input device (here the hex keypad) and return with it in the A register.

INBYTE Get two ASCII characters from the input device, using INCHAR and pack into a single byte in the A register.

OUTCHR Output the ASCII data in the A register to the output device (here the six digit LED display).

OUTBYT Convert the byte in the A register into two ASCII characters and output these to the output device.

Location A417 is used to initialise the 6532 timer to count down from the value stored in A417, with a divide by 1024 cycles. Thus the timer register on the 6532 is decremented by one every 1024 clock cycles. The timer register sits at location A406, and the time is considered to be "up" when the value at A406 becomes negative.

# PEEKING AT PET'S BASIC

Harvey B. Herman
Chemistry Department, U. of N. Carolina
Greensboro, NC 27412

Commodore, for reasons best known to them, has seen fit to prevent users from PEEKing at PET's ROM located, 8K BASIC. If you try to run a program that says, PRINT PEEK (49152), the answer returned will be zero instead of the actual instruction or data in decimal. Disassemblers written in BASIC will therefore not work properly if they use the PEEK command and try to disassemble 8K BASIC (decimal locations 49152 to 57520). I was curious to see how the PET's 8K BASIC was implemented and decided to write a machine language program which circumvents the restriction.

A listing of the above program which I have called MEMPEEK follows. It is decimal 22 bytes long, relocatable, and can be stored into any convenient area of memory. I have chosen to use the area devoted to the second cassette buffer starting at hex 33A. As long as the second cassette is not used the program should remain inviolate until the PET is turned off. Storing the program in memory is trivial if a machine language monitor is available. Otherwise convert the hex values to decimal and manually poke the values into memory. As of this writing, Commodore's free, long-awaited, TIM-like monitor has not arrived but I continue to hope.

MEMPEEK utilizes the user function (USR) which jumps to the location stored in memory locations 1 and 2. If MEMPEEK is stored in the second cassette buffer (hex 33A) initialize locations 1 and 2 to decimal 58 and 3 respectively. MEMPEEK was written so that the user function returns the decimal value of the instruction given by its argument (address). For example, if you want to peek at an address less than decimal 32768 (not part of the BASIC ROMs) use in your program Y=USR (address), where address is the location of interest and the value of Y is set to the instruction at that address. Since the argument of the user function is limited to +32767, use address -65536 for addresses larger than 32768. Thus to look at locations in the BASIC ROMs (all above 32768 and where MEMPEEK is particularly useful) use Y=USR (address -65536). It is not possible to look at location 32768 (the start of the screen memory) with this program but this should prove no handicap as PEEK could be used.

MEMPEEK takes advantage of two subroutines in the PET operating system. The first (located at hex D0A7) takes the argument (address) in the floating point accumulator (conveniently placed there by the user function) and converts it into a two byte integer stored at hex B3 and B4. Since I choose to use an indirect indexed instruction to find the desired instruction the order of the two bytes at hex B3 (MSB) and B4 (LSB) need to be reversed. The second subroutine at hex D278 converts a 2 byte integer representing the instruction from the accumulator (MSB) and the Y register (LSB) to floating point form and stores it in the floating point accumulator. This value, the instruction, is returned to BASIC as the result of the user function.

The program, MEMPEEK, is fairly simple but would be unnessary if the arbitrary restriction on PEEKing at BASIC was removed. The restriction makes no sense to me as even a relatively inexperienced machine language programmer (myself) was able to get around it. This type of program would of course not be difficult for competitors of Commodore to write. I wrote this program for the fun of it, to try to understand how BASIC works and in the hope others will find it useful. Furthermore, I hope I can discourage other manufacturers like Commodore from trying to keep hobbyists from a real understanding of their software by arbitrary restrictions.

MEMPEEK Program

```
033A              *=$33A
033A  20A7D0      JSR $D0A7 ; convert to integer
033D  A6B3        LDX $B3   ; interchange -
033F  A4B4        LDY $B4   ; $B3 and $B4
0341  86B4        STX $B4
0343  84B3        STY $B3
0345  A200        LDX #0    ; initialize index
0347  A1B3        LDA ($B3,X); find instruction
0349  A8          TAY
034A  A900        LDA #0
034C  2078D2      JSR $D278 ; convert to floating
034F  60          RTS       ; return to BASIC
0350              END
```

---

# KIMBASE

Dr. Barry Tepperman
25 St. Mary St., No. 411
Toronto, Ontario M4Y 1R2
Canada

KIMBASE is an application program written in the 6502 microprocessor machine language, designed to make use of the monitor subroutines and memory configuration of the KIM-1 microcomputer, for conversion of unsigned integers from one base to another. The input integer (designated NUMBER is to be no greater than 6 digits in length; large 6-digit integers may cause overflow in the multiplication subroutines with consequent errors in conversion. The base to be converted from (designated BASE1) and to be converted to (BASE2) are each in the range from $02_H$ to $10_H$; the lower limit is set by mathematical reality and the upper by the limited enumeration available from the KIM-1 keypad.

The program is started by placing NUMBER, lowest order byte last, in page zero 4C-4E, BASE1 (expressed in hexadecimal) in 4A, and BASE2 (also in hexadecimal) in 4B. The program starts at 0200, and will light up the KIM-1 LED display with either an error message (according to an error flag stored in zero page 02, called ERROR), or a result display with the input data and a final result up to $18_H$ digits in length (RESULT stored in 03-0E) in successive segments in a format to be discussed below, or a combination of both displays, in an endless loop until the RS key is pressed.

## Program Function

After initialization of data workspace, several tests of input data validity are conducted. KIMBASE recognizes four error states:

a) NUMBER will remain same after conversion (i.e. NUMBER=00000x where x is less than either base). KIMBASE sets ERROR=01, RESULT=NUMBER, and shows both error and result displays.

b) Either or both bases are outside the permissable limits of $02\text{-}10_H$. KIMBASE resets bases under 02 to equal 02 and bases exceeding $10_H$ to equal $10_H$, and executes program to display result without an error display.

c) BASE1=BASE2. KIMBASE sets ERROR=02, RESULT=NUMBER, and shows error and result displays.

d) NUMBER enumeration is impermissable, as one or more digits =BASE1 (e.g., attempting NUMBER=1C352A with BASE1=05). KIMBASE sets ERROR=03, shows error display, and aborts further execution.

Note that error states "a" and "c", above, are not mutually exclusive, and that KIMBASE sets the error flag ERROR and goes to the appropriate response routine after only one positive test. Errors are displayed as a continuous flashing LED readout "ErrorY" where Y=ERROR.

Following the test routines, if $BASE1 \neq 10_H$, KIMBASE converts NUMBER into its hexadecimal equivalent by successive generation of powers of BASE1, multiplication of the appropriate power by the individual digits of NUMBER (remapped by masking and shifting into array N), and successive addition of all the hexadecimal products. This intermediate result is placed in array HEXCON. A successive loop algorithm was used for multiplication rather than a shift-and-binary-add algorithm for economy of coding.

$$HEXCON = \left[ \sum_{y=1-6} N(Y) * BASE1^{(y-1)} \right]_{10}$$

This calculation is bypassed and NUMBER entered directly into HEXCON if $BASE1=10_H$.

After the conversion to hexadecimal, if $BASE2=10_H$, KIMBASE sets RESULT=HEXCON and the result display is initiated. If $BASE2 \neq 10_H$, HEXCON is converted into BASE2 by the common successive division procedure by BASE2 with mapping of remainders through an intermediate array into RUSULT.

Results are displayed on the KIM-1 6-digit display as successive 1-second displays of NUMBER, BASE1 and BASE2, and RESULT divided into 6-digit segments, in the format:

```
NNNNNN    (NUMBER1-NUMBER3)
IIbbOO    (II=BASE1; OO=BASE2)
RRRRRR    (RESULT1-RESULT3)
RRRRRR    (RESULT4-RESULT6)
RRRRRR    (RESULT7-RRSULT9)
RRRRRR    (RESULTA-RESULTC)
```

which loops endlessly. Where ERROR=01 or 02, the error message precedes the result display, and loops endlessly in the display.

All intermediate arrays and products have been retained in the zero page data workspace to facilitate any debugging or further elaboration of the program that other users may find necessary.

Users of non-KIM 6502-based microcomputers may implement KIMBASE easily with appropriate relocation of program and workspace (if necessary) and replacement of the display subroutines (SHOWER-TIMER1, SHORES-TIMER2) with appropriate machine-dependant output routines (or by BRK instructions with manual interrogation of the appropriate arrays to determine output).

*KIMBASE - MAIN PROGRAM LISTING*

```
*************** this section initializes data workspace and constants **********

         CLD                 0200   D8        select binary mode
         LDX   $#48            01   A2 48     set workspace byte counter
ZERO1    LDA   $#00            03   A9 00
         STA   ARRAY,X         05   95 01     zero common workspace
         DEX                   07   CA        decrement counter
         BNE   ZERO1           08   D0 F9     if ≠0 loop back
         LDA   $#0F            0A   A9 0F
         STA   MASK1           0C   85 0F     set MASK1=0F
         LDA   $#F0            0E   A9 F0
         STA   MASK2           10   85 10     set MASK2=F0
```

```
          LDA  $#Ø5          12    A9  Ø5
          STA  PWR           14    85  ØØ      set PWR=Ø5
          LDX  $#FF          16    A2  FF
          TXS                18    9A          set stack pointer=FF

*************** this section tests input data validity ************************

TST1NR    LDA  $#ØØ          19    A9  ØØ         TEST - ERROR STATE "a"
          CMP  NUMBER1       1B    C5  4C      NUMBER1=ØØ?
          BNE  TST1BS        1D    DØ  14      no? go to next test
          CMP  NUMBER2       1F    C5  4D      NUMBER2=ØØ?
          BNE  TST1BS        21    DØ  1Ø      no? go to next test
          LDA  NUMBER3       23    A5  4E
          CMP  BASE2         25    C5  4B      NUMBER3< BASE2?
          BCC  CORR1         27    9Ø  Ø3      yes? go to correction routine
          JMP  TST1BS        29    4C  33  Ø2  go to next test
CORR1     LDA  $#Ø1          2C    A9  Ø1
          STA  ERROR         2E    85  Ø2      set ERROR=Ø1
          JMP  CORR3A        3Ø    4C  5A  Ø2   and jump to CORR3A
TST1BS    LDX  $#Ø2          33    A2  Ø2         TEST - ERROR STATE "b"
TST1B2    LDA  BASE,X        35    B5  49
          CMP  $#Ø2          37    C9  Ø2      BASE(X) < Ø2?
          BCC  CORR2A        39    9Ø  ØB      yes? go to correction routine
          CMP  $#11          3B    C9  11      BASE(X)≥ 11?
          BCC  RESET1        3D    9Ø  ØB      no? bypass correction
CORR2B    LDA  $#1Ø          3F    A9  1Ø
          STA  BASE,X        41    95  49      otherwise set BASE(X)=1Ø
          JMP  RESET1        43    4C  4A  Ø2   and bypass next correction
CORR2A    LDA  $#Ø2          46    A9  Ø2
          STA  BASE,X        48    95  49      set BASE(X)=Ø2
RESET1    DEX                4A    CA          decrement loop counter
          BNE  TST1B2        4B    DØ  E8      and go back if ≠Ø
TST2BS    LDA  BASE2         4D    A5  4B         TEST - ERROR STATE "c"
          CMP  BASE1         4F    C5  4A      BASE2=BASE1?
          BEQ  CORR3         51    FØ  Ø3      yes? go to correction routine
          JMP  TST3BS        53    4C  6A  Ø2  otherwise bypass
CORR3     LDA  $#Ø2          56    A9  Ø2
          STA  ERROR         58    85  Ø2      set ERROR=Ø2
CORR3A    LDX  $#Ø3          5A    A2  Ø3
          LDY  $#ØC          5C    AØ  ØC
CORR3B    LDA  NUMBER,X      5E    B5  4B      read NUMBER
          STA  RESULT,Y      6Ø    99  Ø2  ØØ   into RESULT
          DEY                63    88          decrement counters
          DEX                64    CA                 ..............
          BNE  CORR3B        65    DØ  F7       and loop until complete
          JSR  SHOWER        67    2Ø  AØ  ØØ  display error message
 TST3BS   LDA  BASE1       ØØ6A    A5  4A
          CMP  $#1Ø          6C    C9  1Ø      BASE1=1Ø?
          BCC  TST2NR        6E    9Ø  ØC      no? go to next test
          LDX  $#Ø3          7Ø    A2  Ø3
 HEXMAP   LDA  NUMBER,X      72    B5  4B      yes? read NUMBER
          STA  HEXCON,X      74    95  25       into HEXCON
          DEX                76    CA
          BNE  HEXMAP        77    DØ  F9         for all 3 bytes
          JMP  HEX1          79    4C  1F  Ø3  and bypass hex conversion
 TST2NR   LDA  BASE1         7C    A5  4A         TEST - ERROR STATE "d"
          STA  BSTR1         7E    85  11      store BASE1
          ASL  ASL           8Ø    ØA  ØA
          ASL  ASL           82    ØA  ØA       and left shift 4 bits
          STA  BSTR2         84    85  12         to store BSTR2=(1Ø*BASE1)
          LDY  $#Ø2          86    AØ  Ø2 .
 TLP2     LDX  $#Ø3          88    A2  Ø3
 TLP1     LDA  NUMBER,X      8A    B5  4B      isolate each digit NUMBER(X)
          AND  MASK,Y        8C    39  ØE  ØØ   by masking
          CMP  BSTR,Y        8F    D9  1Ø  ØØ    and compare with BSTR
          BCC  TRESET        92    9Ø  Ø3      if less, reset loop
          JMP  CORR4         94    4C  AØ  Ø2  otherwise impermissable - correct
 TRESET   DEX                97    CA          decrement counter NUMBER
          BNE  TLP1          98    DØ  FØ      and repeat for corresponding digits
          DEY                9A    88          decrement counter BSTR/MASK
          BNE  TLP2          9B    DØ  EB      and repeat for remaining digits
          JMP  REMAP         9D    4C  A7  Ø2  go to REMAP
 CORR4    LDA  $#Ø3          AØ    A9  Ø3
          STA  ERROR         A2    85  Ø2      set ERROR=Ø3
          JSR  SHOWER        A4    2Ø  AØ  ØØ  and display error message
```

```
*************** this section remaps NUMBER for conversion to hex ***************

REMAP     LDX  $#Ø3        A7    A2 Ø3
REMAP1    LDA  NUMBER,X    A9    B5 4B        load NUMBER
          STA  NHI,X       AB    95 12         into NHI
          STA  NLO,X       AD    95 15          and into NLO
          DEX              AF    CA
          BNE  REMAP1      BØ    DØ F7        loop until done
          LDX  $#Ø3        B2    A2 Ø3
MASKS1    LSR  NHI,X       B4    56 12        right shift
          LSR  NHI,X       B6    56 12         NHI
          LSR  NHI,X       B8    56 12          4 bits
          LSR  NHI,X       BA    56 12           .......
          LDA  NLO,X       BC    B5 15
          AND  MASK1       BE    25 ØF        isolate right digit NLO
          STA  NLO,X       CØ    95 15
          DEX              C2    CA
          BNE  MASKS1      C3    DØ EF        loop until done
          LDY  $#Ø1        C5    AØ Ø1
          LDX  $#Ø3        C7    A2 Ø3
REMAP2    LDA  NLO,X       C9    B5 15        store NLO into N
          STA  N,Y         CB    99 18 ØØ
          INY              CE    C8            alternately
          LDA  NHI,X       CF    B5 12          with NHI
          STA  N,Y         D1    99 18 ØØ        and in inverse order
          INY              D4    C8
          DEX              D5    CA
          BNE  REMAP2      D6    DØ F1        loop until done

*************** this section converts N into hexadecimal ***************************

HEXCNV    LDY  $#Ø6        Ø2D8  AØ Ø6        for six places
LP1PWR    JSR  PWRGEN      DA    2Ø 6Ø ØØ     generate powers of BASE1
          LDA  N,Y         DD    B9 18 ØØ
          CMP  $#Ø1        EØ    C9 Ø1        N(Y)=Ø1?
          BEQ  RESET3      E2    FØ ØB        if equal, go to RESET3
          BCC  RESET5      E4    9Ø 15        if less, go to RESET5
          STA  MULTP       E6    85 1F        set MULTP=N(Y)
RESET2    TYA              E8    98           put index Y into accumulator
          PHA              E9    48            and push onto stack
          JSR  MULT        EA    2Ø 8Ø ØØ     multiply power by N(Y)
          PLA              ED    68           pull accumulator from stack
          TAY              EE    A8            and restore to Y
RESET3    CLC              EF    18
          LDX  $#Ø3        FØ    A2 Ø3
RESET4    LDA  MULTC,X     F2    B5 1F        add new product
          ADC  HEXCON,X    F4    75 25         to intermediate product
          STA  HEXCON,X    F6    95 25          and store as intermediate product
          DEX              F8    CA
          BNE  RESET4      F9    DØ F7        loop until done
RESET5    DEY              FB    88           for next place
          BEQ  HEX1        FC    FØ 21         if counter=Ø bypass
          DEC  PWR         FE    C6 ØØ        reduce power to be generated
          LDA  PWR         Ø3ØØ  A5 ØØ
          CMP  $#Ø1        Ø2    C9 Ø1        PWR=Ø1?
          BEQ  RESET6      Ø4    FØ Ø2        yes? go to RESET6
          BCS  LP1PWR      Ø6    BØ D2        greater? loop back to new conversion
RESET6    LDA  N,Y         Ø8    B9 18 ØØ
          STA  MULTC3      ØB    85 22        set MULTC=N(Y)
          LDA  $#ØØ        ØD    A9 ØØ
          STA  MULTC1      ØF    85 2Ø
          STA  MULTC2      11    85 21
          LDA  BASE1       13    A5 4A
          STA  MULTP       15    85 1F        set MULTP=BASE1
          LDA  PWR         17    A5 ØØ
          CMP  $#Ø1        19    C9 Ø1        PWR=Ø1?
          BEQ  RESET2      1B    FØ CB        yes? go to RESET2
          BCC  RESET3      1D    9Ø DØ        less? go to RESET3

*************** this section produces result from HEXCON when BASE2=1Ø *************

HEX1      LDA  BASE2       1F    A5 4B
          CMP  $#1Ø        21    C9 1Ø        BASE2=1Ø?
          BCC  ZERO2       23    9Ø 1Ø        no? go to ZERO2
          LDY  $#ØC        25    AØ ØC
          LDX  $#Ø3        27    A2 Ø3
```

```
HEX2     LDA  HEXCON,X     29     B5 25        store HEXCON
         STA  RESULT,Y     2B     99 Ø2 ØØ      into RESULT
         DEY               2E     88
         DEX               2F     CA
         BNE  HEX2         3Ø     DØ F7        loop until done
         JSR  SHORES       32     2Ø 9Ø Ø3     and display result

*************** this section divides HEXCON by BASE2 for crude conversion ***********

ZERO2    STA  DIVIS        Ø335   85 2C        set DIVIS=BASE2
         LDX  $#Ø3         37     A2 Ø3
LP1DIV   LDA  HEXCON,X     39     B5 25        load HEXCON
         STA  DIVD,X       3B     95 28         into DIVD
         DEX               3D     CA
         BNE  LP1DIV       3E     DØ F9        loop until done
         LDY  $#18         4Ø     AØ 18        for 18H places
LP2DIV   JSR  DIVIDE       42     2Ø 1Ø Ø1     execute division
         LDA  RDR          45     A5 3Ø        load RDR
         STA  RSTOR,Y      47     99 3Ø ØØ      into RSTOR
         LDX  $#Ø2         4A     A2 Ø2
TST1QO   LDA  QUO,X        4C     B5 2C
         CMP  $#Ø1         4E     C9 Ø1        QUO(1 or 2)≥Ø1?
         BCS  RESET7       5Ø     BØ Ø9        yes? go to RESET7
         DEX               52     CA
         BNE  TST1QO       53     DØ F7        loop until done
         LDA  QUO3         55     A5 2F
         CMP  DIVIS        57     C5 2C        QUO3=DIVIS?
         BCC  ENDDIV       59     9Ø 15        less? go to ENDDIV
RESET7   LDX  $#Ø3         5B     A2 Ø3
RST7A    LDA  QUO,X        5D     B5 2C        load QUO
         STA  DIVD,X       5F     95 28         into DIVD
         LDA  $#ØØ         61     A9 ØØ
         STA  QUO,X        63     95 2C        zero QUO
         DEX               65     CA
         BNE  RST7A        66     DØ F5        loop until done
         STA  RDR          68     85 3Ø        zero RDR
         DEY               6A     88           decrement place counter
         BEQ  ENDV2        6B     FØ Ø9        if =Ø go to ENDV2
         JMP  LP2DIV       6D     4C 42 Ø3     otherwise back to divide routine
ENDDIV   DEY               7Ø     88           decrement place counter
         LDA  QUO3         71     A5 2F        load QUO3
         STA  RSTOR,Y      73     99 3Ø ØØ      into next RSTOR slot

*************** this section maps RSTOR into RESULT for final result ****************

ENDV2    LDY  $#ØC         76     AØ ØC
         LDX  $#18         78     A2 18
         CLC               7A     18
REMAP3   DEX               7B     CA
         LDA  RSTOR,X      7C     B5 30        left shift alternate bytes
         ASL  ASL          7E     ØA ØA         RSTOR 4 bytes
         ASL  ASL          8Ø     ØA ØA          ...............
         INX               82     E8
         ADC  RSTOR,X      83     75 3Ø          add to next byte RSTOR
         STA  RESULT,Y     85     99 Ø2 ØØ        and store as RESULT
         DEY               88     88
         DEX               89     CA
         DEX               8A     CA
         BNE  REMAP3       8B     DØ EE        loop until done
         JSR  SHORES       8D     2Ø 9Ø Ø3     and display result
```

*1. PWRGEN*

*Subroutine to generate $a^b$ by successive iterations of multiplication subroutine MULT with resetting of counters and intermediate products; allows unsigned binary or decimal arithmetic in 65Ø2 instruction set; maximum result memory allocated $18_H$ bits.*

```
Requires: subroutines: MULT    ØØ8Ø-ØØ9B

          data arrays: BASE1   ØØ4A
                       PWR     ØØØØ
                       PWRS    ØØØ1
                       MULTP   ØØ1F
                       MULTC   ØØ2Ø-ØØ22
```

*Inapplicable to PWR=ØØ,Ø1; calling program must test and bypass.*

```
PWRGEN    LDA  PWR        ØØ6Ø   A5 ØØ     load power
          STA  PWRS         62   85 Ø1     store in counter
          DEC  PWRS         64   C6 Ø1     decrement counter
          LDA  BASE1        66   A5 4A
          STA  MULTP        68   85 1F     set multiplier=base
          STA  MULTC3       6A   85 22     set multiplicand=base
          LDA  $#ØØ         6C   A9 ØØ
          STA  MULTC1       6E   85 2Ø     zero 2 high-order bytes
          STA  MULTC2       7Ø   85 21      of multiplicand
          TYA               72   98        transfer index Y to accumulator
          PHA               73   48         and onto stack
MULTCL    JSR  MULT         74   2Ø 8Ø ØØ  jump to MULT
          DEC  PWRS         77   C6 Ø1     decrement counter
          BNE  MULTCL       79   DØ F9     if ≠Ø return to MULTCL
          PLA               7B   68        pull accumulator from stack
          TAY               7C   A8         and restore to index Y
          RTS               7D   6Ø        return to main program
```

2. MULT

Subroutine multiplies 24-bit number (MULTC) by 8-bit number (MULTP) to yield 24-bit final product (MULTC) by successive iterations of nested addition loops. Intermediate product storage in MIDPRO. Allows unsigned decimal or binary operation in 65Ø2 instruction set.

```
Requires : data arrays : MULTP     ØØ1F
                         MULTC     ØØ2Ø-ØØ22
                         MIDPRO    ØØ23-ØØ25
```

Inapplicable to MULTP less than Ø2; calling program to test and bypass

```
MULT      LDY  MULTP      ØØ8Ø   A4 1F     loop counter=multiplier
          DEY               82   88        decrement loop counter
          LDX  $#Ø3         83   A2 Ø3     set byte counter in loop
REDIST    LDA  MULTC,X      85   B5 1F     set intermediate register
          STA  MIDPRO,X     87   95 22      =multiplier
          DEX               89   CA          for each byte in array
          BNE  REDIST       8A   DØ F9     loop until X=Ø
ADLP2     LDX  $#Ø3         8C   A2 Ø3     set byte counter in loop
          CLC               8E   18        clear carry
ADLP1     LDA  MULTC,X      8F   B5 1F     add multiplicand
          ADC  MIDPRO,X     91   75 22      to intermediate product
          STA  MULTC,X      93   95 1F       store as new multiplicand
          DEX               95   CA            for each byte in array
          BNE  ADLP1        96   DØ F7     loop until X=Ø
          DEY               98   88        decrement loop counter
          BNE  ADLP2        99   DØ F1     another loop if Y≠Ø
          RTS               9B   6Ø        return to main program
```

3. DIVIDE

Subroutine to divide 24-bit dividend (DIVD) by 8-bit divisor (DIVIS) to yield 24-bit quotient (QUO) and 8-bit remainder (RDR) by successive shift and subtraction processes; unsigned binary arithmetic only in 65Ø2 instruction set. Intermediate quotient storage in QUO. Requires initialization of RDR and array QUO to Ø by calling program, DIVIS≠Ø.

```
Requires : data arrays : DIVD      ØØ29-ØØ2B
                         DIVIS     ØØ2C
                         QUO       ØØ2D-ØØ2F
                         RDR       ØØ3Ø
```

```
DIVIDE    LDX  $#19       Ø11Ø   A2 19     load shift counter
LOOP1     ASL  RDR          12   Ø6 3Ø     left shift remainder
          ASL  QUO3         14   Ø6 2F     left shift quotient LSB
LOOP1A    BCS  HIQUO1       16   BØ 28     go to incrementing routine
                                            if carry set
          ASL  QUO2         18   Ø6 2E     left shift quotient mid-byte
          BCS  HIQUO2       1A   BØ 2F     go to incrementing routine
                                            if carry set
          ASL  QUO1         1C   Ø6 2D     left shift quotient MSB
```

```
LOOP2    CLC              1E     18          clear carry
         ASL   DIVD3      1F     Ø6 2B       left shift dividend LSB
         BCS   HIORD1     21     BØ 2F       go to incrementing routine
                                              if carry set
         ASL   DIVD2      23     Ø6 2A       left shift dividend mid-byte
         BCS   HIORD2     25     BØ 36       go to incrementing routine
                                              if carry set
         ASL   DIVD1      27     Ø6 29       left shift dividend MSB
LOOP3    BCS   INCR       29     BØ 39       go to incrementing routine
                                              if carry set
LOOP4    DEX              2B     CA          decrement shift counter
         BEQ   FINIS      2C     FØ 3B       jump to end if X=Ø
         SEC              2E     38          set carry
         LDA   RDR        2F     A5 3Ø       from current remainder
         SBC   DIVIS      31     E5 2C        subtract divisor
         BMI   LOOP1      33     3Ø DD       back to LOOP1 if negative
         STA   RDR        35     85 3Ø       store difference as remainder
         ASL   RDR        37     Ø6 3Ø       left shift remainder
         ASL   QUO3       39     Ø6 2F       left shift quotient LSB
         INC   QUO3       3B     E6 2F       increment quotient LSB
         JMP   LOOP1A     3D     4C 16 Ø1     and go back to LOOP1A
HIQUO1   ASL   QUO2       4Ø     Ø6 2E       left shift quotient mid-byte
         INC   QUO2       42     E6 2E        and increment it
         BCS   HIQUO2     44     BØ Ø5       go to further incrementing
                                              routine if carry
         ASL   QUO1       46     Ø6 2D       left shift quotient MSB
         JMP   LOOP2      48     4C 1E Ø1     and back to LOOP2 (if C=Ø)
HIQUO2   ASL   QUO1       4B     Ø6 2D       left shift quotient MSB
         INC   QUO1       4D     E6 2D       increment quotient MSB
         JMP   LOOP2      4F     4C 1E Ø1     and back to LOOP2
HIORD1   ASL   DIVD2      52     Ø6 2A       left shift dividend mid-byte
         INC   DIVD2      54     E6 2A       increment dividend mid-byte
         BCS   HIORD2     56     BØ Ø5       go to further incrementing
                                              routine if carry
         ASL   DIVD1      58     Ø6 29       left shift dividend MSB
         JMP   LOOP3      5A     4C 29 Ø1     and back to LOOP3 (if C=Ø)
HIORD2   ASL   DIVD1      Ø15D   Ø6 29        left shift dividend MSB
         INC   DIVD1      5F     E6 29        increment dividend MSB
         JMP   LOOP3      61     4C 29 Ø1      and back to LOOP3
INCR     INC   RDR        64     E6 3Ø        increment remainder
         JMP   LOOP4      66     4C 2B Ø1      and back to LOOP4
FINIS    LSR   RDR        69     46 3Ø        right shift remainder to end
         RTS              6B     6Ø           return to main program
```

4. SHOWER & TIMER1

Subroutines to generate error message for display on the KIM-1 6-digit LED readout by successive lighting of appropriate segments of the individual digits using a message lookup table.

```
SHOWER requires: subroutines: TIMER1    ØØDE-ØØE9    timing loop for display
                              SHORES    Ø39Ø-Ø3CF    result display for ERROR=Ø1 or Ø2

                : data arrays: SADD     1741
                               SBDD     1743
                               SAD      174Ø         monitor storage for readout
                               SBD      1742
                               ERROR    ØØØ2
                               MSGERR   ØØD6-ØØDA
                               MSGNUM   ØØDB-ØØDD

SHOWER   LDA   $#7F       ØØAØ   A9 7F
         STA   SADD       A2     8D 41 17    set output directional vector A=7F
         LDA   $#1E       A5     A9 1E
         STA   SBDD       A7     8D 43 17    set output directional vector B=1E
DISP2    LDY   $#Ø8       AA     AØ Ø8       set digit selection counter
         LDX   $#Ø5       AC     A2 Ø5       set loop counter
DISP1    STY   SBD        AE     8C 42 17    select digit
         LDA   MSGERR,X   B1     B5 D5       select segments
         STA   SAD        B3     8D 4Ø 17     to be lit (from lookup table)
         JSR   TIMER1     B6     2Ø DE ØØ      and jump to timing loop
         INY              B9     C8          select next digit
```

```
         INY                   BA    C8
         DEX                   BB    CA              decrement loop counter
         BNE   DISP1           BC    DØ  FØ          if ≠Ø loop again
         LDA   $#12            BE    A9  12
         STA   SBD             CØ    8D  42  17      for sixth digit
         LDX   ERROR           C3    A6  Ø2          set index to error flag
         LDA   MSGNUM,X        C5    B5  DA          and select segments
         STA   SAD             C7    8D  4Ø  17       to be lit (from lookup table)
         JSR   TIMER1          CA    2Ø  DE  ØØ        and jump to timing loop
         LDA   ERROR           CD    A5  Ø2
         CMP   $#Ø3            CF    C9  Ø3          if ERROR=Ø3
         BEQ   DISP2           D1    FØ  D7           loop same display again
         JMP   SHORES          D3    4C  9Ø  Ø3      otherwise jump to show result
```

lookup tables:

```
ØØD6   DØ  DC  DØ  DØ  F9      MSGERR
ØØDB   86  DB  CF              MSGNUM
```

TIMER1 requires: interval timer location 17Ø7

```
TIMER1   LDA   $#FF          ØØDE    A9  FF          set timer for approximately
         STA   17Ø7            EØ    8D  Ø7  17       200 milliseconds per digit
DELAY1   NOP                   E3    EA              do nothing but light segments
         BIT   17Ø7            E4    2C  Ø7  17      time up?
         BPL   DELAY1          E7    1Ø  FA          no? keep lit
         RTS                   E9    6Ø              yes? back to SHOWER for next digit
```

5. SHORES & TIMER2

Subroutines to generate result display on the KIM-1 6-digit LED readout by loading appropriate data into array DISP for display by KIM monitor subroutine SCANDS.

```
SHORES requires: subroutines: TIMER2    Ø3DØ-Ø3E5    timing loop for display
                              SHOWER    ØØAØ-ØØD5    error display for ERROR=Ø1 or Ø2

                : data arrays: ERROR    ØØØ2
                              RESULT    ØØØ3-ØØØE
                              BASE      ØØ4A-ØØ4B
                              NUMBER    ØØ4C-ØØ4E
                              DISP      ØØF9-ØØFA    monitor storage for readout:
                                                        ØØF9     INH
                                                        ØØFA     POINTL
                                                        ØØFB     POINTH

SHORES   LDY   $#Ø1          Ø39Ø    AØ  Ø1          set index for DISP
         LDX   $#Ø3            92    A2  Ø3          set index for NUMBER
LOADN1   LDA   NUMBER,X        94    B5  4B          put NUMBER into DISP
         STA   DISP,Y          96    99  F8  ØØ
         INY                   99    C8              increment DISP index
         DEX                   9A    CA              decrement NUMBER index
         BNE   LOADN1          9B    DØ  F7          loop until DISP is full
         JSR   TIMER2          9D    2Ø  DØ  Ø3       and jump to timing/display loop
         LDA   BASE1           AØ    A5  4A          load BASE1
         STA   POINTH          A2    85  FB           into two highest digits
         LDA   $#BB            A4    A9  BB          load BB
         STA   POINTL          A6    85  FA           into two middle digits
         LDA   BASE2           A8    A5  4B          load BASE2
         STA   INH             AA    85  F9           into two lowest digits
         JSR   TIMER2          AC    2Ø  DØ  Ø3        and jump to timing/display loop
         LDX   $#Ø1            AF    A2  Ø1          set index for RESULT
LOADN3   LDY   $#Ø3            B1    AØ  Ø3          set index for DISP
LOADN2   LDA   RESULT,X        B3    B5  Ø2          put RESULT (3 bytes at a time)
         STA   DISP,Y          B5    99  F8  ØØ       into DISP
         INX                   B8    E8              increment RESULT index
         DEY                   B9    88              decrement DISP index
         BNE   LOADN2          BA    DØ  F7          loop until DISP is full
         TXA                   BC    8A              put RESULT index into accumulator
```

```
        PHA                  BD    48              and push onto stack
        JSR   TIMER2         BE    2Ø  DØ  Ø3        now jump to timing/display loop
        PLA                  C1    68              pull accumulator from stack
        TAX                  C2    AA               and put in RESULT index X
        CPX   $#ØD           C3    EØ  ØD          is X > ØC?
        BCC   LOADN3         C5    9Ø  EA          if not, loop back to load DISP
        LDA   ERROR          C7    A5  Ø2          if yes, does ERROR=ØØ?
        CMP   $#ØØ           C9    C9  ØØ
        BEQ   SHORES         CB    FØ  C3          if yes, loop again for whole display
        JMP   SHOWER         CD    4C  AØ  ØØ      otherwise show error

TIMER2 requires: subroutines: SCANDS      1F1F          monitor display subroutine

                 data arrays: CTLP        ØØ49
                 interval timer location 17Ø7

TIMER2  LDA   $#Ø5         Ø3DØ    A9  Ø5
        STA   CTLP           D2    85  49          set loop counter
DSPN2   LDA   $#FF         Ø3D4    A9  FF          set timer for maximum run
        STA   17Ø7           D6    8D  Ø7  17
DSPN1   JSR   SCANDS         D9    2Ø  1F  1F      and call display subroutine
        BIT   17Ø7           DC    2C  Ø7  17      time up?
        BPL   DSPN1          DF    1Ø  F8          no? maintain display
        DEC   CTLP           E1    C6  49          decrement loop counter
        BNE   DSPN2          E3    DØ  EF          if ≠Ø, reset timer and maintain display
        RTS                  E5    6Ø              otherwise back to SHORES for next entry
```

---

## "THE BEST OF MICRO VOLUME I"

Even though we had extra copies of MICRO printed we could not keep up with the demand for back issues. We have run out of all back issues and all copies of "All of MICRO Volume 1". Since a lot of people who are just finding out about MICRO or are just getting into the 6502 world still want the information which was contained in the first year of MICRO, we have decided to print "The BEST of MICRO Volume 1".

This will contain most of the articles but none of the advertising. A few articles which were topical and are now out-of-date will be dropped and all known microbes will be corrected back in the original articles. The book will be organized by subject. Aside from these minor changes, the content will be identical to that of MICRO numbers 1 through 6. If you already have them, you will not profit by getting the new edition. If you do not have them, then this will be the only way to get the information.

"The BEST of MICRO Volume 1" will be available about the first of November. It will be about 160 page long in an 8 by 11 format, soft cover. The price will be $6.00 (plus $1.00 postage US)

Send your Check or Money Order to:

The BEST of MICRO
P.O. Box 3
So. Chelmsford, MA 01824

## ADVERTISING IN MICRO

It doesn't COST to advertise in MICRO, it PAYS!

MICRO is currently printing 10,000 copies for distribution. 3000+ will go immediately to subscribers and dealers. The remainder will go to new subscribers and to replenish dealer stock throughout the coming year - so you get a lot of coverage for your dollar, into a readership that is eager to know about 6502 oriented products.

DEADLINES for Issue Number 8 - December/January

Ad Reservation by 6 November
Ad Copy by 13 November

The rates are very reasonable for the coverage:

| | | |
|---|---|---|
| Quarter Page | (4 x 5) | $50.00 |
| Half Page | (8 x 5) | $75.00 |
| Full Page | (8 x 10) | $125.00 |

10% discount on six consecutive insertions.

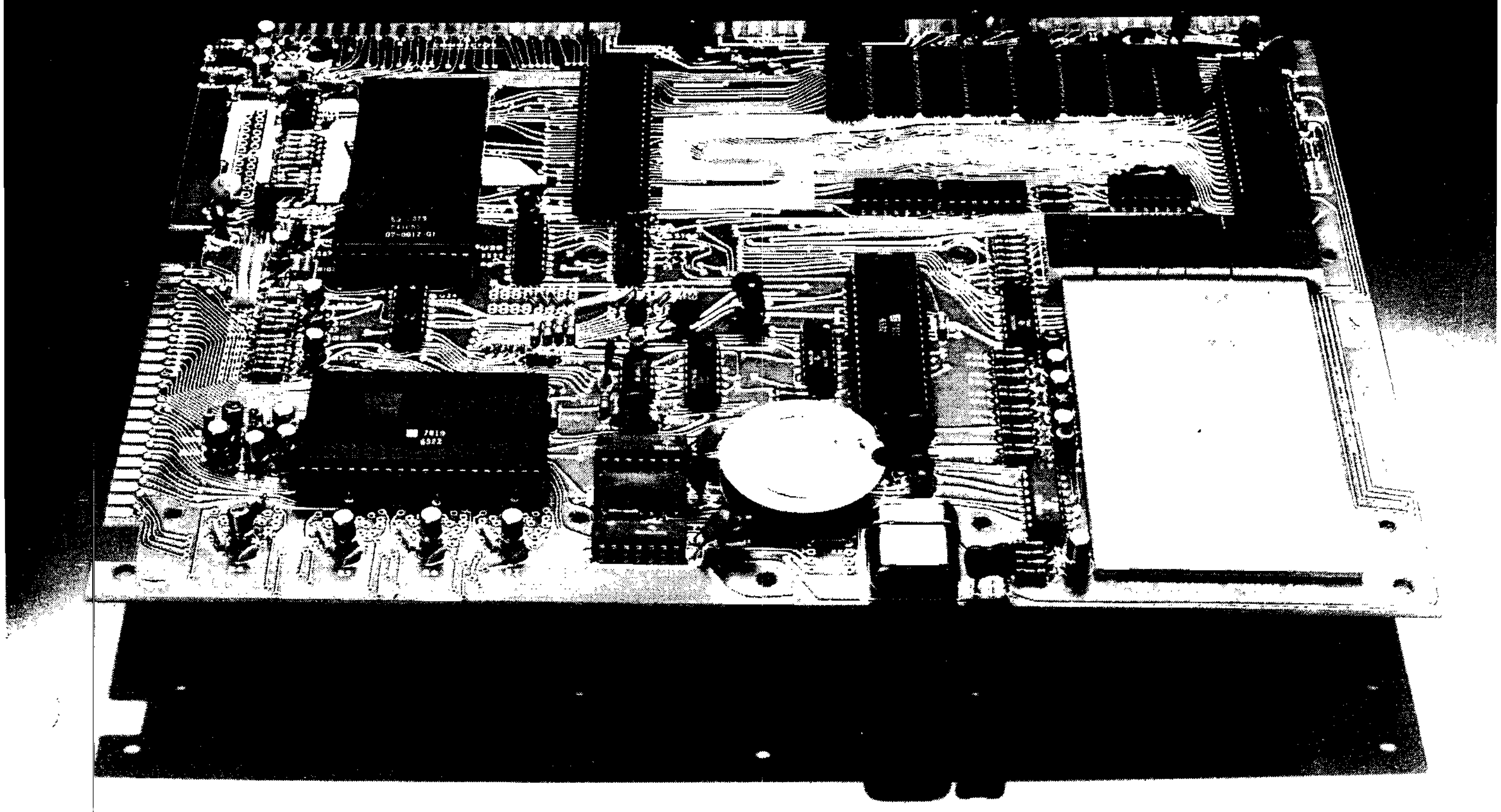
Send Ad copy to:

MICRO, P.O. Box 3, So. Chelmsford, MA 01824

or call for info or Ad reservation:

617/256-3649

# SYM-1.
# Finally, a dependable microcomputer board.



In performance. In quality. In availability. OEMs, educators, engineers, hobbyists, students, industrial users: Our Versatile Interface Module, SYM-1, is a fully-assembled, tested and warranted microcomputer board that's a true single-board computer, complete with keyboard and display. All you do is provide a +5V power supply and SYM-1 gives you the rest—and that includes fast delivery and superior quality.

**Key features include:**

- Hardware compatibility with KIM-1 (MOS Technology) products.
- Standard interfaces include audio cassette with remote control: both 8 bytes/second (KIM) and 185 bytes/second (SYM-1) cassette formats; TTY and RS232; system expansion bus; TV/KB expansion board interface; four I/O buffers; and an oscilloscope single-line display.
- 28 double-function keypad with audio response.
- 4K byte ROM resident SUPERMON monitor including over 30 standard monitor functions and user expandable.
- Three ROM/EPROM expansion sockets for up to 24K bytes total program size.
- 1K bytes 2114 static RAM, expandable to 4K bytes on-board and more off-board.
- 50 I/O lines expandable to 70.
- Single +5V power requirements.
- Priced attractively in single unit quantities: available without keyboard/display, with OEM discounts for larger quantities.

**Synertek Systems Corporation.**
150-160 S. Wolfe Road, Sunnyvale, California 94086
(408) 988-5690.

To place your order now, contact your local area distributor or dealer.

**OEM Distributors**
Kierulff Electronics
Sterling Electronics (Seattle only)
Zeus Components
Century/Bell
Lionex
Hallmark
Intermark Electronics
[illegible]
Technico
General Radio
Western Microtechnology
Future Electronics
Alliance Electronics
Arrow Electronics

**Personal Computer Dealers**
Newman Computer Exchange
Ann Arbor, Michigan
Technico
Columbia, Maryland
Computerland
Mayfield Heights, Ohio
RNB Enterprises
King of Prussia, Pennsylvania
Computer Shop
Cambridge, Massachusetts
Computer Cash
Anchorage, Alaska
Ancrona
Culver City, California
General Radio
Camden, New Jersey
Advanced Computer Products
Santa Ana, California
Computer Comp[illegible]
Van Nuys, Calif[illegible]
Alltronics
San Jose, Ca[illegible]