MICRO™ on the OSI

MICRO

on the

OSI

MICRO INK
P.O. Box 6502
Amherst, New Hampshire 03031

MICRO on the OSI

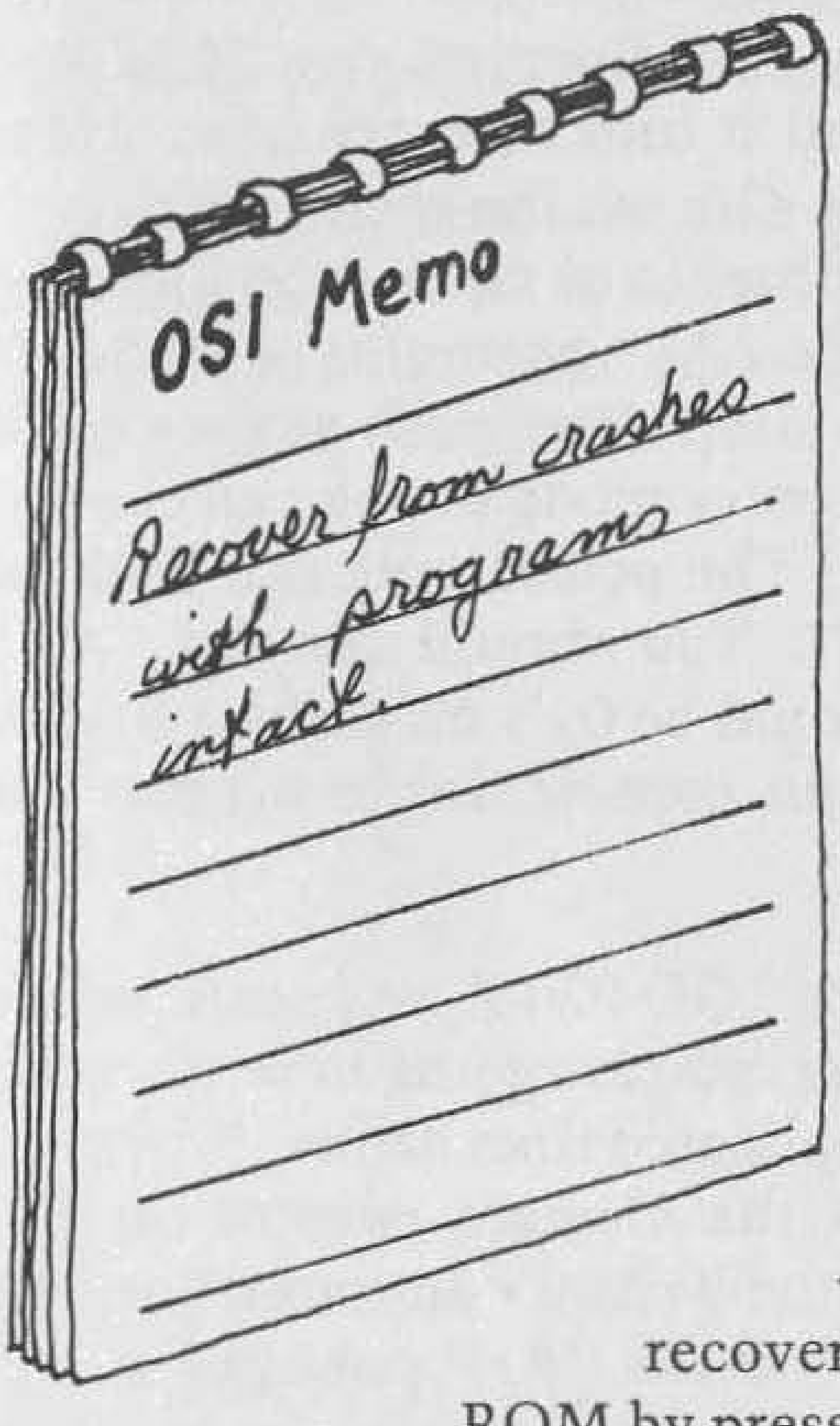
Technical Editor: Kerry Lourash
Contributing authors: Michael J. Alport, Matt Asay, Lester Cain, David Cantrell, Leo Jankowski, Rolf Johannesen, Michael J. Keryan, John Krout, Kerry Lourash, Collin Macauley, Jeff Macauley, Michael M. Mahoney, Yasuo Morishita, Earl Morris, John S. Seybold, Charles L. Stanford, Terry Terrance, Richard L. Trethewey.

MICRO on the OSI ISBN: 0-938222-12-0

Table of Contents

Warm Start Under OS-65D

by Richard L. Trethewey



OS-65D users have had to live with the fact that you can't warm start OS-65D. So if your program suddenly locks up or (with a pre-1981 vintage OSI system) if you accidentally touch the <BREAK> key while typing in your program, you have to start over. Most of the time this problem only means retyping a few lines of code. But if you're like me and prone to programming "on the fly" without periodically saving to disk, it could mean hours of work lost. In this article I will show you a way to recover that lost time with a minimum of effort.

Usually when you touch the <BREAK> key while in BASIC, you can recover your program by entering the monitor ROM by pressing "M" and then "G" (for GO). This can warm start BASIC, but not completely. At this point you can neither run your program nor save it to disk. If all has gone well so far, you can LIST your program to the indirect file and re-boot the system and recover it. This method doesn't always work and does no good if you're using the Assembler/Editor and not BASIC. Also, when you reboot the system, the BEXEC* program writes directly over your old program. Therein lies the key to our success. If the BEXEC* program isn't called into memory, your old program will remain pretty much intact unless you turn off your system.

OS-65D uses a slick method to run the BEXEC* automatically when you boot up. On cold start, the input flag is set for memory input and the memory pointers are set to the command 'RUN"BEXEC*" ', which is called into memory with the rest of OS-65D. Also on cold starting, BASIC checks the I/O flags to see if a console device has been selected. If so, it says "Hello," tells you how much memory you have, and awaits instructions. Should this not be the case, BASIC runs the BEXEC* or executes whatever other instruction was stored on disk. Our task, then, is to change the I/O flags on cold start.

My suggestion for having a reliable method of recovering your programs involves the use of the TRACK 0 Read/Write utility. If you have never used this program, I strongly advise you to consult your manual before proceeding. The prompts in that program are very terse; without further explanation, you won't know what's happening.

To begin, make a duplicate of any OS-65D diskette. If you used the copier program from track 1 (track 13 on mini-floppies) the TRACK 0 utility is still in memory. If you didn't, call it into memory now. Enter "GO 0200" at the "A*" prompt and select #2 from the menu displayed. Now enter "R4200". This will call the contents of track 0 to memory location $4200. Type "E" for exit and at the "A*" prompt type "RE M" to enter the monitor ROM. Now enter ".4321/". The slash at the end of that sequence puts the monitor in the data entry mode so you can change memory. Now type "02<RETURN>02.". The period puts you back in the addressing mode. Now type "2547G". You should see the "A*" prompt again. Note that the 02's above should be 01's on serial systems because you are setting the I/O flags to your console device number on cold start.

Run the TRACK 0 utility again by typing "GO 0200" and again select option 2 from the menu. This time, however, you are going to write track 0 and the instructions are a little more complicated than before. Enter the command "W4200/2200,8". This makes the changes current on the disk. When you boot the disk it won't run the BEXEC* anymore but will start up BASIC as if you had entered "BA" at the "A*" prompt.

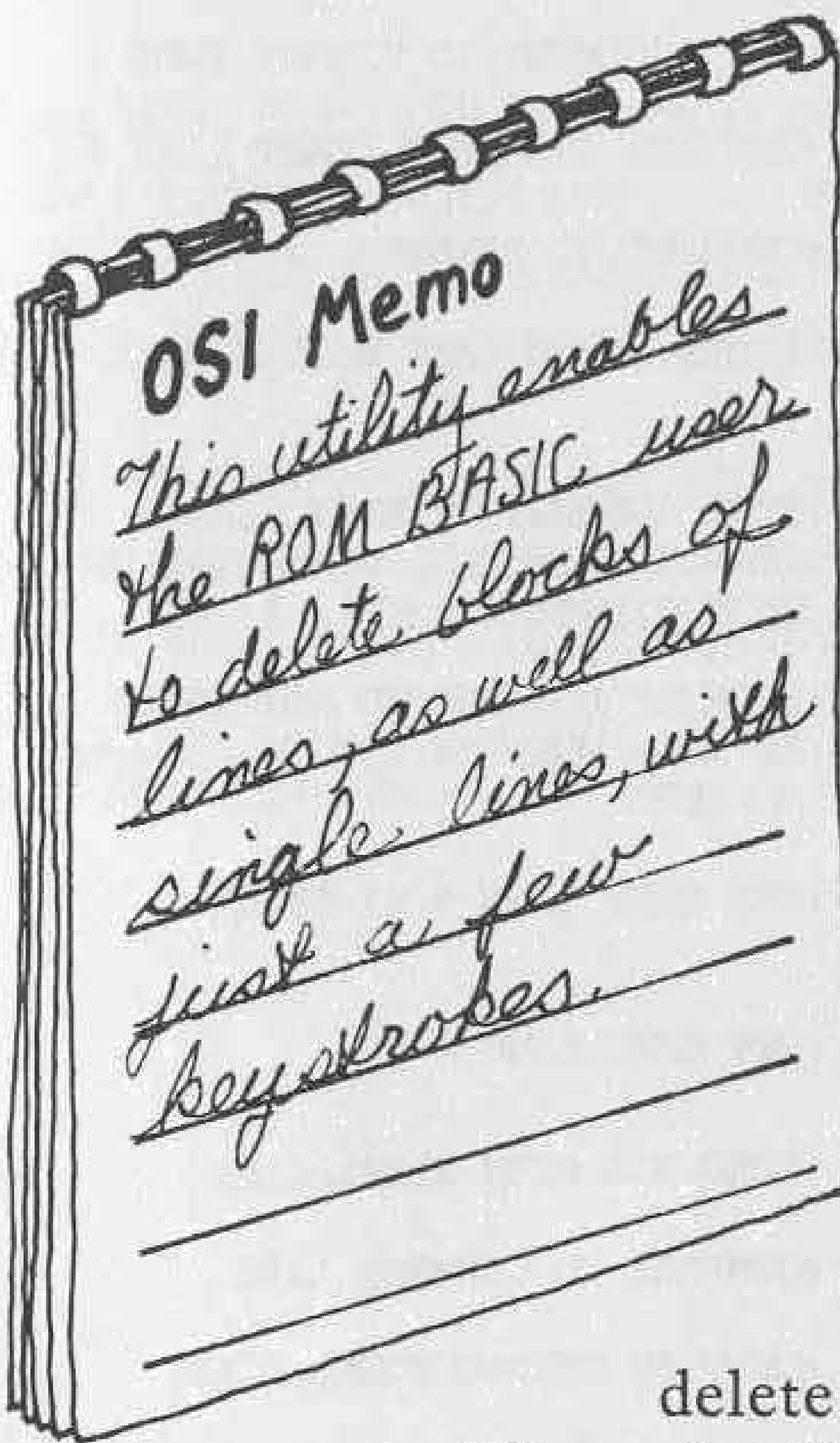
To recover a program press <BREAK>, if you haven't already. Press "M" to enter the monitor ROM and enter ".3A79" for all sizes of OS-65D V3.3. If you are running 3.2 enter ".3179" on 8-inch systems and ".3279" on 5¼-inch systems instead of the above. This is where the file header starts. The file header holds the addresses of where your program starts and ends. This information may not be current if you have altered your program since it was stored on disk, but that won't matter. You need to record the next eight bytes for later so write down the number displayed after the address. Press the "/" key and a <RETURN>. Now write down the number for the next location. Copy down the information through address $??80. Put the diskette just made in the "A" drive and boot it up. BASIC should say "Hello", etc., and "OK". Type "EXIT" and "RE M" as before. Press ".3A79" (or your system's header address as described above) and then the "/" key. Replace the eight bytes of information that you copied down by entering the numbers, followed by a <RETURN>. Now type ".2547G" and you're back at the "A*" prompt. Type "RE BA" to re-enter BASIC.

If the file you are working on is an assembly program, instead of typing "RE M" at the "OK" prompt from BASIC, type "AS" first to invoke the assembler and then "RE M". Replace the eight bytes as described above and type "RE A" instead of "RE BA". List the program to the indirect file. Under BASIC this is done with "LIST<SHIFT>'K' ". With the assembler it's "P<SHIFT>'K' ". Now clear the workspace with a "NEW" or "I" and then enter a <CTRL>'X'. This will reload the entire program into the workspace and update the resident language. Your program is now intact and you can run it and/or save it to disk.

The special recovery disk just made does not need to be devoted to this single purpose. It is still a standard OS-65D disk and you can put whatever you like on it. As you can see, this technique doesn't really cost anything and could save quite a bit of time and effort.

Delete

by Earl Morris and Yasuo Morishita



Normally only a single line of BASIC can be deleted by typing in the line number followed by a carriage return. This is tedious if a large block of lines must be removed; for example, when programs are merged or a utility program is run with another program also in memory. The "DELETE" program creates a USR routine that is called by

Z = USR (first line)(last line)

All the lines of BASIC with line numbers inside the specified range are then deleted.

When OSI ROM BASIC is called to delete a single line, two major routines are used. The code at $A2A2 finds the line to be deleted and then shrinks the program by the number of bytes found in the offending line. Another routine at $A31C is responsible for refixing the pointers that rechain each line to the next. Unfortunately these routines are not written as subroutines and therefore cannot be used by "outside" programs. However, the DELETE program copies these routines from ROM into RAM and creates the needed subroutine. The main line DELETE program accepts the first line to be deleted and calls the copied ROM routine to do the work. Then the line pointers are used to find the line number of the next BASIC line. This is checked for end-of-program and to see if it exceeds the upper limit for deleting. Then the copied routines are called again and the process is repeated until completed. Lines are still deleted one at a time, but the computer, rather than your busy fingers, is doing the work.

The BASIC program listed here will create the DELETE program on page two below the start of BASIC program space. This memory is normally unused in OSI machines. If you are using this space, then the delete program can be relocated by changing the value of "M" in line 14.

Listing 1: Source Code for Main Delete Program

```
                    ;DELETE
                    ;BY MORRIS & MORISHITA
                    ;
                    ;ASSEMBLY LANGUAGE LISTING
                    ;
                         ORG $235

                    ;
0235 20 08 B4            JSR $B408          ;1ST ARGUMENT TO BINARY INTO $
11,12 (START)
0238 20 AD AA            JSR $AAAD          ;GET 2ND ARGUMENT (LAST LINE # )

023B 20 31 B8            JSR $B831          ;CONVERT TO BINARY
023E A5 AF               LDA $AF
0240 85 30               STA $30            ;STORE FINAL LINE # IN $30,31
0242 A5 AE               LDA $AE
0244 85 31               STA $31
0246 20 32 A4       LBLA JSR $A432          ;FIND ADDRESS OF BASIC LINE
0249 B0 1B               BCS LBLB           ;BRANCH IF FOUND, OTHERWISE UP
                                             DATE POINTER AT $11,12
024B A0 01               LDY #$01
024D B1 AA               LDA ($AA),Y        ;LOOK AT POINTER TO NEXT LINE
024F F0 1A               BEQ LBLC           ;IF NULL MUST BE END OF PROGRAM
                                             SO QUIT
0251 A0 03               LDY #$03
0253 B1 AA               LDA ($AA),Y        ;GET NEXT LINE # HI BYTE
0255 85 12               STA $12
0257 88                  DEY
0258 B1 AA               LDA ($AA),Y        ;GET NEXT LINE # LO
025A 85 11               STA $11
025C A6 30               LDX $30            ;LOAD X,A WITH FINAL LINE
025E A5 31               LDA $31
0260 E4 11               CPX $11            ;COMPARE TO CURRENT LINE
0262 E5 12               SBC $12
0264 90 05               BCC LBLC           ;QUIT IF BEYOND FINAL LINE
0266 20 75 02       LBLB JSR $0275
0269 F0 DB               BEQ LBLA           ;ALWAYS BRANCH
026B A9 92          LBLC LDA #$92
026D A0 A1               LDY #$A1           ;$A192 IS ADDRESS OF "OK"
026F 20 C3 A8            JSR $A8C3          ;PRINT "OK"
0272 4C 19 A3            JMP $A319          ;GO BACK TO BASIC
0275                ;
0275                     END
```

Line 16 sets up the USR vector and line 18 builds the main program from the DATA statements. Line 20 moves the "memory close" routine from ROM. Line 22 calculates an absolute JSR address and POKEs it into the main program. Line 24 copies the rechaining routine from ROM and line 26 adds an "RTS" to convert it to a subroutine.

After running the BASIC program, it can delete itself with

Z = USR (10)(44)

Note that the USR function now requires two arguments and will give an "SN" error if both are not present. Everything is deleted by Z = USR(1) (−1), which is the same as a NEW command. The form Z = USR (A)(B) is also helpful to figure out which lines to omit.

The source code for the main program is listed with comments for those readers interested in how the program works. The code is relocatable with the exception of the JSR at $026E. This is a jump to the copied ROM routines. The BASIC set-up program automatically fixes this absolute address.

Listing 2: BASIC Program to Set Up USR Delete Function

```
10 REM  BASIC LINE DELETE
12 REM FORMAT :  Z=USR(START LINE #)(END LINE #)
14 M=565:REM START ADDRESS=$0235 RELOCATABLE
16 A=INT(M/256):POKE12,A:POKE11,M-A*256
18 N=64:FORX=MTOM+N-1:READJ:POKEX,J:NEXT
20 A=41634:M=M+N:N=68:GOSUB28:REM DELETE=$A2A2
22 A=INT(M/256):B=M-256*A:POKEM-13,A:POKEM-14,B
24 A=41756:M=M+N:N=47:GOSUB28:REM REBUILD  =$A31C
26 POKEM+15,96:END: REM "RTS"
28 FORX=0TON-1:J=PEEK(A+X):POKEM+X,J:NEXT:RETURN
30 DATA32,8
32 DATA180,32,173,170,32,49,184,165,175,133
34 DATA48,165,174,133,49,32,50,164
36 DATA176,27,160,1,177,170,240,26,160,3
38 DATA177,170,133,18,136,177,170,133,17
40 DATA166,48,165,49,228,17,229,18
42 DATA144,5,32,125,2,240,219,169,146,160
44 DATA161,32,195,168,76,25,163
```

[illegible]

[illegible]

After entering the BASIC program, it can [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

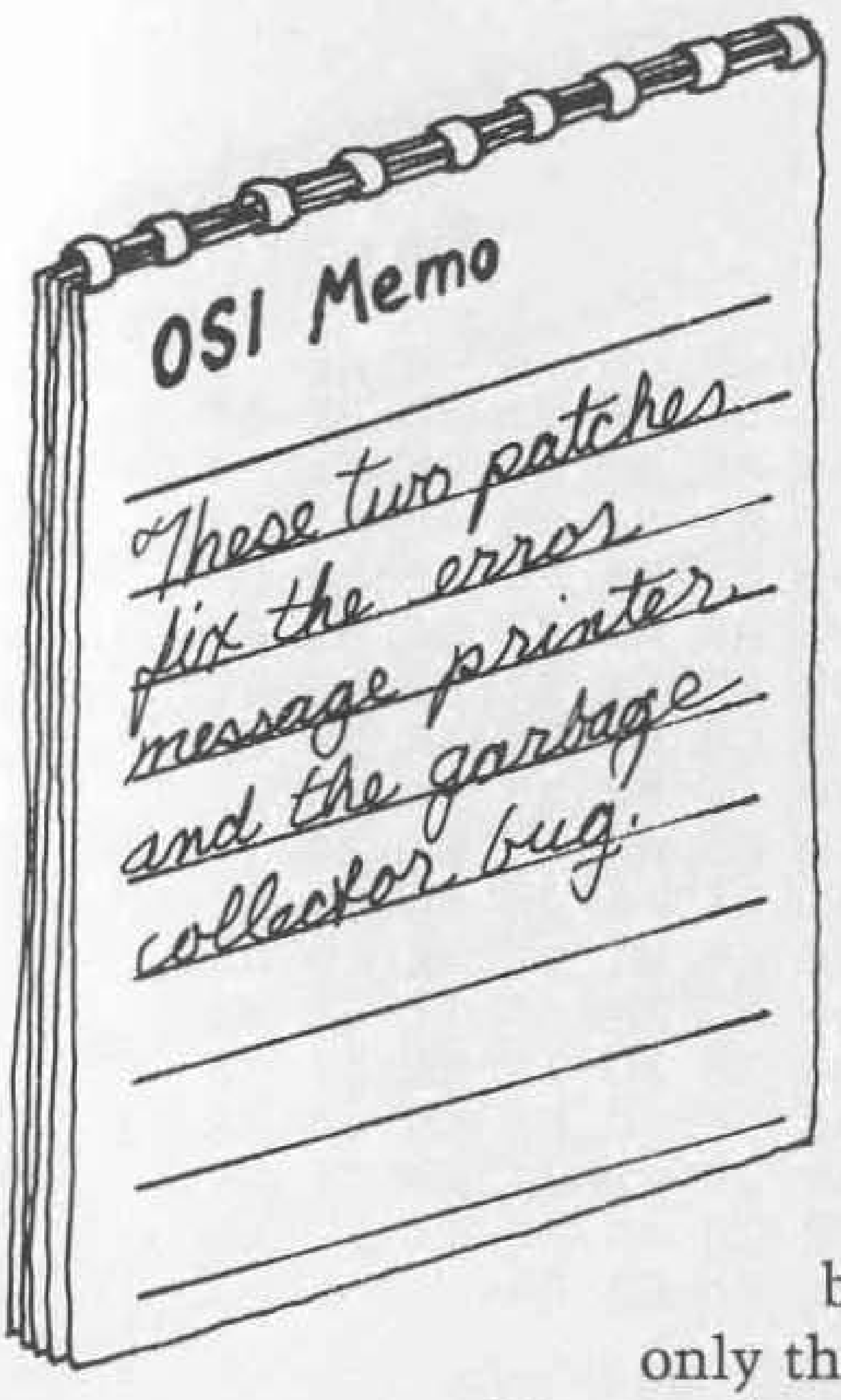
[illegible]

LISTING [illegible] BASIC Program [illegible]

[illegible]

Two Fixes for ROM BASIC

by Earl Morris



Here are two patches for OSI BASIC-in-ROM. The shorter patch fixes the error message printer; the longer one cures the dreaded garbage collector bug. These are not add-on programs, but are direct replacements for the code in the BASIC ROMs. To install these patches you must burn an EPROM replacement for the BASIC ROM.

Error Message Patch

BASIC uses two-character error messages with the high bit of the second character set. Before the graphic chip came along, error messages were printed correctly because the old character ROM decoded only the lower seven bits of a letter. The graphic chip translates the letter as a graphic character, since it decodes all eight bits, and an odd-looking shape appears in the error message. This patch fixes the small but irritating problem.

Garbage Collector Patch

When a string is manipulated, the resultant string is stored at the top of free memory. If enough of these strings are created, they fill the free memory space. At this point, the garbage collector routine is called to find the strings that are still valid and pack them at the top of free memory. Unfortunately, OSI's garbage collector has a bug in it that causes the screen to flash and the computer to "hang" if complicated string manipulation is done. Many partial solutions have been published, but this patch seems to be one of the best answers to the problem.

Listing 1: Error Message Patch

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
A160  49 44 A4 00 4E 46 53 4E 52 47 4F 44 46 43 4F 56
A170  4F 4D 55 53 42 53 44 44 2F 30 49 44 54 4D 4C 53
A180  53 54 43 4E 55 46
```

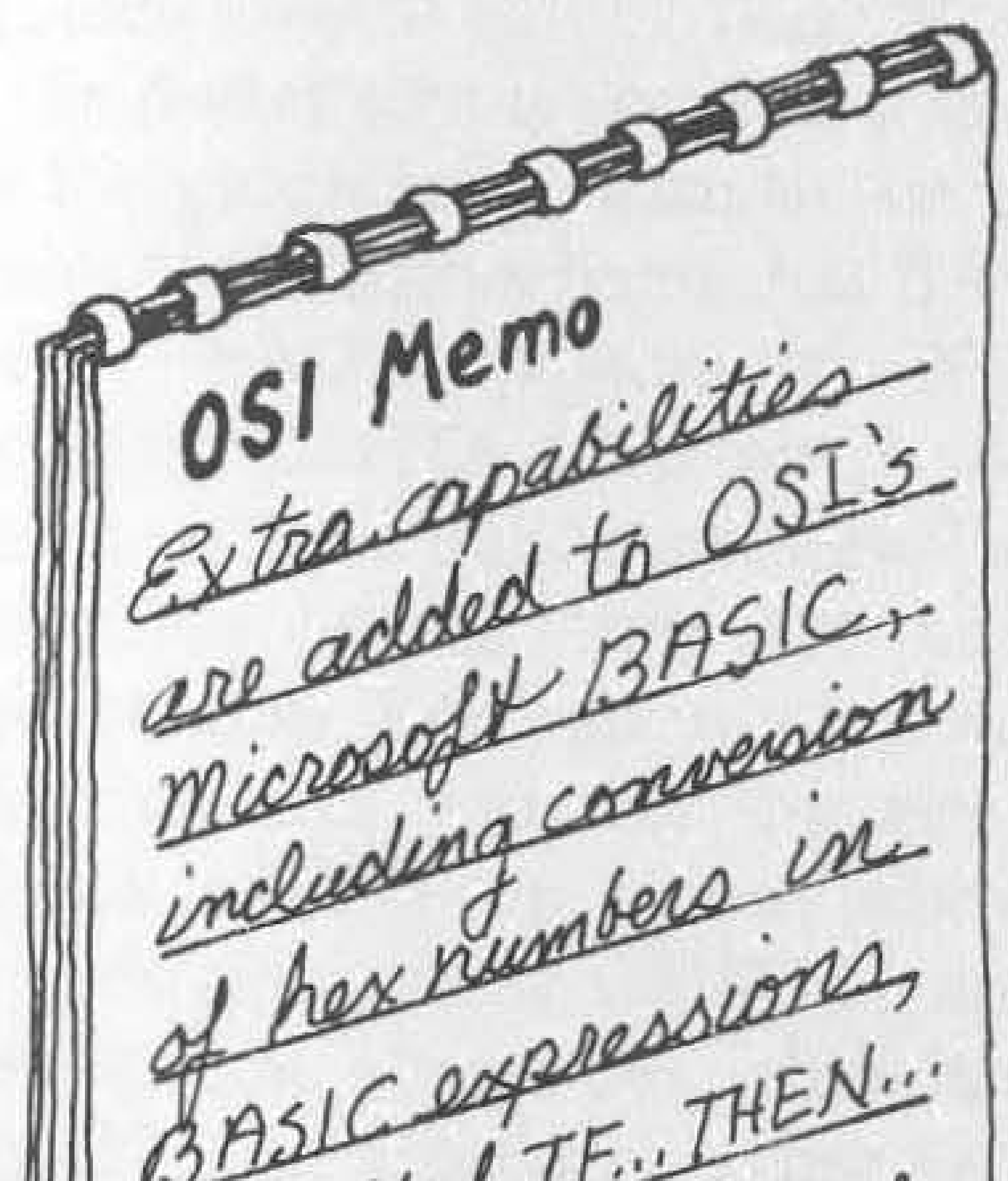
Listing 2: Garbage Collector Patch

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
B140  A9 80 85 60 68 D0 D0 A6 85 A5 86 86 81 85 82 A0
B150  00 84 9D A5 7F A6 80 85 AA 86 AB A9 68 85 71 84
B160  72 C5 65 F0 05 20 D9 B1 F0 F7 A9 06 85 A0 A5 7B
B170  A6 7C 85 71 86 72 E4 7E D0 04 C5 7D F0 05 20 D3
B180  B1 F0 F3 85 A4 86 A5 A9 04 85 A0 A5 A4 A6 A5 E4
B190  80 D0 07 C5 7F D0 03 4C 18 B2 85 71 86 72 A0 01
B1A0  B1 71 08 C8 B1 71 65 A4 85 A4 C8 B1 71 65 A5 85
B1B0  A5 28 10 D7 C8 B1 71 A0 00 0A 69 05 65 71 85 71
B1C0  90 02 E6 72 A6 72 E4 A5 D0 04 C5 A4 F0 C1 20 D9
B1D0  B1 F0 F3 C8 B1 71 10 30 C8 B1 71 F0 2B C8 B1 71
B1E0  AA C8 B1 71 C5 82 90 06 D0 1E E4 81 B0 1A C5 AB
B1F0  90 16 D0 04 E4 AA 90 10 86 AA 85 AB A5 71 A6 72
B200  85 9C 86 9D 88 88 84 A2 A5 A0 18 65 71 85 71 90
B210  02 E6 72 A6 72 A0 00 60 C6 A0 A6 9D F0 F5 A4 A2
B220  18 B1 9C 65 AA 85 A6 A5 AB 69 00 85 A7 A5 81 A6
B230  82 85 A4 86 A5 20 D6 A1 A4 A2 C8 A5 A4 91 9C AA
B240  E6 A5 A5 A5 C8 91 9C 4C 4B B1 EA EA EA
```

Editor's Note: The original version of the garbage collector patch was written by Dick Stibbons and published in the OSI/UK User Group Newsletter, Vol. 1, No. 4. The original code has been modified to correctly collect strings with a zero subscript like A$(0).

Getting BASIC to Behave with OS-65D

by Richard L. Trethewey



The Microsoft BASIC provided on MA/OSI systems was written in 1977 and lacks many of the niceties of newer systems. While the actual source code for BASIC isn't available from either Microsoft or MA/OSI, Aardvark Technical Services in Walled Lake, Michigan, sells a disassembled listing of OS-65D's BASIC. Using this listing as a guide, I was able to make BASIC do some things that otherwise would have been impossible. In under one page of RAM, I was able to make BASIC understand hex (in most places), allow named GOSUBs and GOTOs, and provide a limited IF...THEN... ELSE. This code easily fits behind the "HOOKS into OSI BASIC" I wrote (see MICRO 46:43) and does not interfere with the normal operation of the system. All your old programs will still run with it in place.

Aardvark sells the BASIC source code listing for $24.99 — one of the best bargains around. The 110+-page booklet is well commented and easy to understand. Aardvark also sells listings of OS-65D and ROM BASIC at reasonable prices.

There is a small price to pay for these additions. Since BASIC is an interpreted language it is slow, and adding patches makes it slower. For most applications, the additions I discuss won't affect the timing noticeably. The patch added in the "HOOKS" article costs time only when a variable is assigned a value, but no more so than if you had a dozen extra variables in your program.

The first addition I discuss adds hex capabilities to BASIC. It is accessed whenever BASIC has to deal with a number that appears in your

Listing 1

```
0000           ;**************************************
0000           ;* ADDITIONS TO BASIC UNDER OS-65D V3.3 *
0000           ;*                                     *
0000           ;*      BY RICHARD L. TRETHEWEY        *
0000           ;**************************************
0000                   ;
0000           ;*******LABELS FROM BASIC*******
0000                   ;
0000                   ADDON =$08FC
0000                   ASCII =$1BEE
0000                   CHKCHR=$0E15
0000                   CRDO  =$0A73
0000                   CHRGET=$C0
0000                   CHRGOT=$C6
0000                   FLOAT =$1B44
0000                   FORPNT=$96
0000                   FACEXP=$AE
0000                   FACHI =$AF
0000                   FACMHI=$B0
0000                   FACMLO=$B1
0000                   FACLO =$B2
0000                   FACSGN=$B3
0000                   FRMEVL=$0CCD
0000                   GOTOTK=$88
0000                   GOTO  =$08A6
0000                   INT   =$1BC7
0000                   LINGET=$096C
0000                   OUTDO =$0AEE
0000                   POKER =$19
0000                   PTRGET=$0F2E
0000                   QUINT =$1B96
0000                   REM   =$093C
0000                   REMTK =$8E
0000                   SNERR =$0E1E
0000                   THENTK=$A0
0000                   TXTPTR=$C7
0000                   ;
0000           ;*******OS-65D LABELS*******
0000                   ;
0000                   CASECK=$3A5F
0000                   CHROUT=$2343
0000                   PRBYTE=$2D92
0000                   ;
0000                   ;ROUTINE TO PRINT IN HEX
0000                   ;REPLACES 'H*' COMMAND IN HOOKS
0000                   ;
BD4F           *=$BD4F
BD4F 20C000            JSR CHRGET   THROW AWAY ASTERISK
BD52 20C000            JSR CHRGET   GET NEXT CHARACTER
BD55 20CD0C            JSR FRMEVL   EVALUATE EXPRESSION
BD58 24B3              BIT FACSGN   POS OR NEG?
BD5A 1007              BPL HO       BRANCH IF POS.
BD5C A92D              LDA #'-      PRINT NEG SIGN
BD5E 204323            JSR CHROUT
BD61 46B3              LSR FACSGN   MAKE IT POSITIVE
BD63 2016BF    HO      JSR LIN      MAKE IT AN INTEGER
BD66 A924              LDA #'$      PRINT A '$'
BD68 204323            JSR CHROUT
```

(continued)

program as text (rather than a variable name). In programs that use a lot of numbers without assigned variable names, this speed overhead can be annoying. Using numbers instead of variables in such applications should be avoided, and adding this patch to BASIC may force you to edit some programs. The named GOTOs and new IF code make little difference in speed.

To get BASIC to understand hex, I intercept the code that evaluates numeric expressions. These expressions include equalities and functions. BASIC first looks to see if the term is a variable name or a number in discrete form. If you precede a hex value with a dollar sign, BASIC thinks the character being looked at is a number and not a name. Before BASIC decides how to handle this value, you should interrupt it and check to see if the dollar sign is there. If it is not, execute the instructions written over by the patch and return to the normal code. If it is, you must translate it from ASCII into a form that BASIC understands and then put the number where BASIC expects it. With these additions in place, only the GOSUB/GOTO function in BASIC won't understand hex. With this patch you can do instant hex/decimal calculations and use hex values in programs where they are easier to understand than their decimal equivalents. You can mix hex and decimal in your calculations, too. This addition lets you use the hex form for equalities, FOR/NEXT loops, PEEKs and POKEs, or anywhere you use a number.

Adding named GOSUBs and GOTOs is simple. Instead of always demanding a number, this patch checks to see if the character is a variable name before letting BASIC continue. If you find a name, look up its value and give it to BASIC. That's all!

My version of IF copies the original code up to the point where BASIC decides that the statement is false. Since you can't add the keyword ELSE in the regular table without removing a necessary keyword, I have added an extra function to the REM keyword. With my patch in place, the REM will serve both its original comment function and a new ELSE function. As in normal BASIC operation, a true condition will cause the statement after the THEN to be executed and the REM to be ignored. When the condition is false, though, BASIC will look for a REM in the remainder of the line and execute a simple line-transfer operation placed there. If there is no REM, BASIC will proceed to the next line, as usual. The line transfer is equivalent to GOTO; no other expressions can be evaluated after the REM. Your existing BASIC programs must have their REMs removed from lines with IF...THEN statements.

If you have implemented the hooks into BASIC, I suggest you replace the instructions that interpret the "H*" command with lines 400 to 630 of the new subroutine in listing 1. If you haven't added hooks, you will have to make the first line of your BEXEC* similar to

```
10DISK!"CA BE00=TT,S":POKE133,189:POKE8960,189
```

Listing 1 *(continued)*

```
BD6B A51A            LDA POKER+1 GET HI BYTE
BD6D F003            BEQ H1      IF 0, SKIP IT
BD6F 20922D          JSR PRBYTE  NON-ZERO, SO PRINT
BD72 A519     H1     LDA POKER   GET LOW BYTE
BD74 20922D          JSR PRBYTE  AND PRINT IT
BD77 68              PLA         CANCEL THE 'JSR'
BD78 68              PLA         THAT GOT US HERE
BD79 4C730A          JMP CRDO    DO CR, LF
BD7C 00       RESLO  .BYT 0
BD7D 00       RESHI  .BYT 0
BD7E 0000     INBUF  .DBY 0,0,0
BD80 0000
BD82 0000
BD84                 ;CODE TO ALLOW HEX IN BASIC FORMULAS
BD84                 ;$0DC3 4C7CBE        JMP $BE7C
BD84                 ;
BE7C          *=$BE7C
BE7C C924            CMP #'$     IS IT HEX?
BE7E F00A            BEQ HEXFLT  YES, DO IT!
BE80 C92E            CMP #'.     NO, REPLACE INSTRUCTIONS HERE
BE82 D003            BNE HEX6
BE84 4CEE1B          JMP ASCII
BE87 4CC70D   HEX6   JMP $0DC7
BE8A                 ;
BE8A A004     HEXFLT LDY #4
BE8C A900            LDA #0
BE8E 8D7DBD          STA RESHI   INIZ
BE91 997EBD   HEX0   STA INBUF,Y
BE94 88              DEY
BE95 D0FA            BNE HEX0
BE97 20C000   HEX1   JSR CHRGET  GET NEXT CHARACTER
BE9A F023            BEQ HEX3    CHECK FOR TERMINATOR
BE9C C93A            CMP #':
BE9E F01F            BEQ HEX3
BEA0 C97F            CMP #$7F
BEA2 B01B            BCS HEX3
BEA4 C930            CMP #'0
BEA6 9017            BCC HEX3
BEA8 205F3A          JSR CASECK  MAKE IT UPPER CASE
BEAB 38              SEC
BEAC E900            SBC #0      STRIP OFF ASCII
BEAE C90A            CMP #$A     <10?
BEB0 9002            BCC HEX2
BEB2 E907            SBC #7
BEB4 997EBD   HEX2   STA INBUF,Y SAVE IN BUFFER
BEB7 C8              INY         BUMP CHAR. COUNT
BEB8 C005            CPY #5      TOO MANY?
BEBA D0DB            BNE HEX1    OK, TO HEX1
BEBC 4C1E0E          JMP SNERR
BEBF 88       HEX3   DEY         POINT TO LAST CHAR.
BEC0 B97EBD          LDA INBUF,Y GET LOWEST CHAR.
BEC3 8D7CBD          STA RESLO   SAVE IT
BEC6 C000     HEX5   CPY #0      ONLY ONE DIGIT?
BEC8 F022            BEQ HEX4    YES, WE'RE DONE
BECA 88              DEY         NO, BUMP POINTER
BECB B97EBD          LDA INBUF,Y GET CHARACTER
BECE 0A              ASL A       SHIFT LEFT 4 BITS
BECF 0A              ASL A
BED0 0A              ASL A
BED1 0A              ASL A
```

(continued)

This will call the code into memory and protect it from being overwritten by BASIC or 65D. I have removed the hex-to-decimal conversion since it is replaced by the new code. This version allows the output of any number or variable in hex form. It is limited to numbers less than $FFFF (as are all the other routines here), but at least now you can use both variables and numbers in your conversions.

You will notice that often I do a JSR to a subroutine called CASECK. This OS-65D V3.3 subroutine converts all lower-case letters to upper case. By using the routine here and elsewhere in the "HOOKs into BASIC," you can blind all your commands to upper/lower case. Usually the comments in the code let you know what is happening. If you need more information, I suggest you refer to the books listed at the end of this article. OS-65D V3.2 users should delete the references to CASECK.

The patches to BASIC that implement these changes are simple. To allow hex inputs, change $0DC3 to $4C, $0DC4 to $7C, and $0DC5 to $BE using the monitor ROM. To get named GOSUBs and GOTOs, change $08A7 to $0B and $08A8 to $BF in the same manner. Getting the change for IF...THEN is a little trickier since the jump to the monitor alters this code. Instead of using the monitor ROM, just do POKEs if you have made the above changes and the code is in place. Enter the following line in the immediate mode:

```
POKE$214,$21:POKE$215,$BF
```

(In this case a foible of the 6502 necessitates pointing to one byte before the actual location.) When you have made these changes, save them to disk with the following instructions (consult your manual if you are using a mini-floppy disk):

```
DISK!"SA 02,1 =0200/B":DISK!"SA 03,1 =0D00/B"
```

When changing the ASCII to a floating-point routine, call the code first, as the code at the high end gets overlayed when BASIC is invoked. First, call in the code to high memory with

```
DISK!"CA 4800=04,1"
```

Then do these POKEs:

```
POKE$4BEE,$4C:POKE$4BEF,$51:POKE$4BF0,$BF
```

Finally, save the code with

```
DISK!"SA 04,1 =4800/B"
```

That will make the changes to BASIC permanent.

Listing 1 *(continued)*

```
BED2 18              CLC
BED3 6D7CBD          ADC RESLO      ADD TO PREVIOUS RESULT
BED6 8D7CBD          STA RESLO      AND SAVE IT
BED9 C000            CPY #0         ARE WE DONE?
BEDB F00F            BEQ HEX4       YES, TO HEX4
BEDD 88              DEY            NO, BUMP POINTER
BEDE B97EBD          LDA INBUF,Y    REPEAT PROCESS
BEE1 0A              ASL A
BEE2 0A              ASL A
BEE3 0A              ASL A
BEE4 0A              ASL A
BEE5 18              CLC
BEE6 6D7DBD          ADC RESHI
BEE9 8D7DBD          STA RESHI
BEEC AD7DBD   HEX4   LDA RESHI      TRANSFER RESULT TO
BEEF 85AF            STA FACHI      FLOATING POINT ACC.
BEF1 AE7CBD          LDX RESLO
BEF4 86B0            STX FACMHI
BEF6 A290            LDX #$90
BEF8 38              SEC
BEF9 20441B          JSR FLOAT      CHANGE FROM INT TO F.P.
BEFC AE7CBD          LDX RESLO      SOME FUNCTIONS NEED THIS
BEFF 60              RTS
BF00                 ;
BF00                 ;PATCH TO GOTN TO ALLOW VARIABLES
BF00                 ;IN GOTO'S AND GOSUB'S
BF00                 ;
BF0B          *=$BF0B
BF0B B003            BCS LINE       IT COULD BE A VARIABLE
BF0D 4C6C09          JMP LINGET     NO, IT'S A NUMBER
BF10                 ;
BF10 202E0F   LINE   JSR PTRGET     LOOK UP VARIABLE
BF13 209D1A          JSR $1A9D      PUT IT IN FACC
BF16 20961B   LIN    JSR QUINT      MAKE IT AN INTEGER
BF19 A5B2            LDA FACLO
BF1B 8519            STA POKER
BF1D A5B1            LDA FACMLO
BF1F 851A            STA POKER+1
BF21 60              RTS
BF22                 ;
BF22                 ;DISPATCH TABLE SENDS 'IF' HERE
BF22                 ;$0214=$21   $0215=$BF
BF22                 ;
BF22 20CD0C   NEWIF  JSR FRMEVL     EVALUATE EXPRESSION
BF25 20C600          JSR CHRGOT
BF28 C988            CMP #GOTOTK
BF2A F005            BEQ NEWIF1
BF2C A5A0            LDA THENTK
BF2E 20150E          JSR CHKCHR
BF31 A5AE     NEWIF1 LDA FACEXP
BF33 F003            BEQ FALSE      IF FALSE, CHECK FOR 'ELSE7
BF35 4C4109          JMP $0941      TRUE, DO IT!
BF38 A41E     FALSE  LDY 30
BF3A B1C7     F1     LDA (TXTPTR),Y
BF3C F010            BEQ NOREM      LOOK FOR 'REM7
BF3E C98E            CMP #REMTK
BF40 F003            BEQ F2
BF42 C8              INY
BF43 D0F5            BNE F1         BRANCH ALWAYS
BF45 20FC08   F2     JSR ADDON      UPDATE TXTPTR
```

(continued)

References

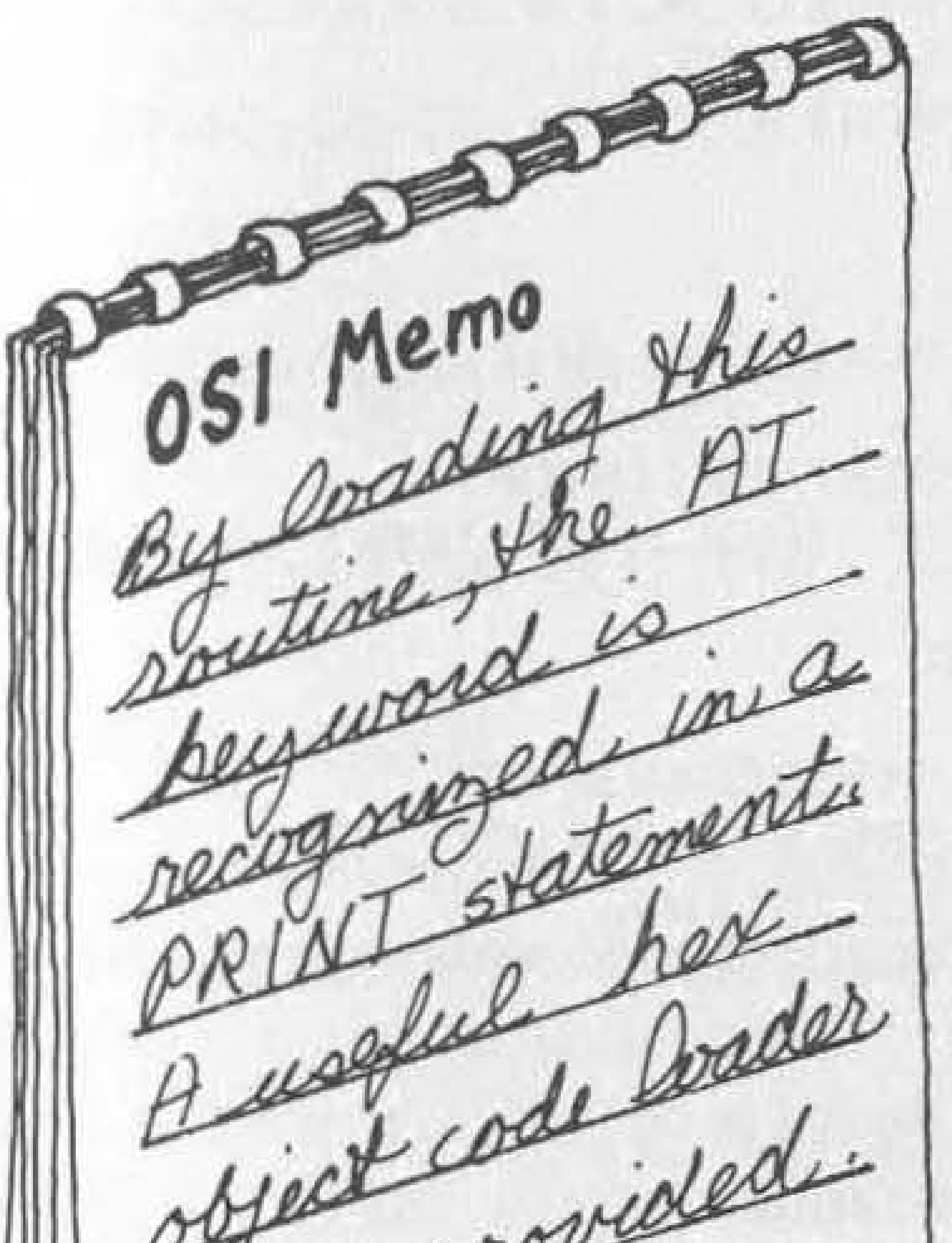
1. *OS-65D V3.2 Source Code*, MA/COMM Office Systems, Inc., Aurora, OH 44202.
2. Barden, William, *How to Program Microcomputers.*
3. *OSI-Microsoft BASIC Assembly Source Listing*, Aardvark Technical Services, Walled Lake, MI.

Listing 1 *(continued)*

```
1770 BF48 20C000          JSR CHRGET  PLUS ONE
1780 BF4B 4CA608          JMP GOTO
1790 BF4E                 ;
1800 BF4E 4C3C09   NOREM  JMP REM     NO ELSE, SO REM
1810 BF51                 ;
1820 BF51                 ;PATCH TO ASCII TO F.P. CONVERSION
1830 BF51                 ;TO ALLOW EITHER HEX OR DECIMAL
1840 BF51                 ; $1BEE 2051BF      JSR $BF51
1850 BF51                 ;
1860 BF51 C924            CMP #'$
1870 BF53 F00A            BEQ VAL1
1880 BF55 20C600          JSR CHRGOT
1890 BF58 A000            LDY #0
1900 BF5A A20A            LDX #$A
1910 BF5C 4CF21B          JMP $1BF2   RE-ENTER NORMAL CODE IF DEC.
1920 BF5F 4C8ABE   VAL1   JMP HEXFLT  NO, IT'S HEX. DO IT!
```

PRINT AT

by Matt Asay



The Microsoft BASIC on an Ohio Scientific C1P has most of the features found on other versions. One feature that is lacking is the ability to print at a selected location on the screen. There are some ways to get around this by using POKE, but you are limited to POKEing one character at a time, which is slow and cumbersome.

I have developed a program to remove these limitations by adding an AT option to the PRINT statement. Once this program is installed you can print anything anywhere on the screen with ease. The program hides itself at the top of your available memory on any size system and uses only 166 bytes of permanent storage. After it has been entered you can write, save, load, and run programs using the new PRINT AT statement. Programs that do not use AT in their PRINTs should function as always.

The syntax of the statement is:

PRINT AT *location; print-list;*

where there are three forms of *location*:

1. A numeric expression. Printing starts at sc + INT(expression), where sc is the address of the screen.
2. Two numeric expressions separated by a comma. Printing starts at sc + INT(expr1) * 32 + INT (expr2). This allows specification of location by row and column.
3. An asterisk ("*"). Printing continues with the position immediately after the last character printed by the last PRINT AT.

print-list is any allowable list of items to be printed, separated by semicolons. The trailing semicolon is necessary since the carriage

Listing 1: BASIC Program to Load, Initialize, and Demonstrate PRINT AT

```
1 REM -----PRINT AT-----
2 REM ---BY MATT ASAY---
3 REM
6 GOSUB 10: GOTO 1000
7 REM
8 REM RELATIVE HEX LOADER SUBROUTINE
9 REM (SEE TEXT FOR A DESCRIPTION)
10 DEF FNA(D)=ASC(MID$(H$,D,1))
20 DEF FNX(D)=FNA(D)-48+(FNA(D)>64)*7
30 DEF FNB(D)=FNX(D)*16+FNX(D+1)
40 DEF FNH(D)=((FNX(D)*16+FNX(D+1))*16+FNX(D+2))*16+FNX(D+3)
45 READ H$: RO=PEEK(134)*256+PEEK(133)-FNH(1)
50 FORH=ROTO32767:READH$:PRINTH$:ONLEN(H$)GOTO 51,52,53,54,55:GOTO54
51 RETURN
52 POKE H,FNB(1):NEXT:STOP
53 RA=RO+FNB(2):GOTO 56
54 POKEH,FNB(1):FORI=3TOLEN(H$)STEP2:H=H+1:POKEH,FNB(I):NEXTI,H:STOP
55 RA=RO+FNH(2)
56 IF LEFT$(H$,1)="H" THEN POKE H,RA/256:NEXT:STOP
57 POKE H,RA AND 255:IF LEFT$(H$,1)="R" THEN H=H+1:POKE H,RA/256
58 NEXT:STOP
100 DATA 00FD: REM SIZE OF CODE IN HEX
105 REM CODE FOR USRX
110 DATA A9,L57,A0,H57,8581848285838484858584 86A207
120 DATA BD,R4F,95C5CA10F8AD1A02AC1B028D,RE6,8C,RE7
130 DATA A9,LE1,A0,HE1,8D1A028C1B02AD1C02AC1D02
140 DATA 8D,RFB,8C,RFC,A9,LF6,A0,HF6,8D1C028C1D02A988A0AE
150 DATA 850B840C60C920F0F34C,R57,00
155 REM CODE FOR PARSER SPLICE (PSPLIC)
160 DATA 24CC1014C941D00E489848A001B1C3C954F013
170 DATA 68A86806CCC997D00285CCC93AB0034CCD0060
175 REM CODE FOR PRINT AT (PR.AT)
180 DATA 46CC68A86820BC0020BC00C9A5D00620BC0038
190 DATA B04120C1AA2008B420C200C92CD023A5110A0A
200 DATA 0A0A26120A8511A5122A48A5114820C9AA2008B4
210 DATA 68186511851168651285 12A5118D,RE9,A5122903
220 DATA 09D08D,REA,20C200C93BD0034CBC00A91C85CC4C4EA2
225 REM CODE FOR OUTPUT SPLICE (OSPLIC)
230 DATA 24CC70034C00008D00D0EE,RE9,D003EE,REA,C60E60
235 REM CODE FOR CTRL C SPLICE (CSPLIC)
240 DATA A90085CC4C0000
245 REM END-OF-DATA MARKER
250 DATA*
260 REM
990 REM INITIALIZE PRINT AT WHILE PRESERVING
995 REM ANY PREVIOUS USR FUNCTION
1000 UL=PEEK(11): UH=PEEK(12)
1020 POKE 11,RO-INT(RO/256)*256: POKE 12,RO/256
1040 X=USR(X)
1060 POKE 11,UL: POKE 12,UH
1100 REM A SHORT DEMO OF THE USE OF PRINT AT
1200 PRINT:PRINT: PRINT" TEST PROGRAM"
1220 FOR I=1 TO 20: PRINT:NEXT
1230 PRINT AT 10*32+5;"PRINT";
1240 PRINT AT *;" AT";
1250 PRINT AT 12,5;"HAS BEEN";
1260 PRINT "WORKS !!!";
1270 PRINT AT *;" LOADED...";
1280 A$="AND IT"
1290 PRINT AT 14,20-LEN(A$);A$;
1300 FOR I=1 TO 500: NEXT
```

return and linefeed that BASIC tags on will print as their corresponding graphics characters. This was done intentionally to allow the printing of all graphics characters using CHR$( ).

Examples

```
PRINT AT 200;CHR$(248);" < - A tank";

PRINT AT X,Y; "PRINT AT ROW X, COLUMN Y";

PRINT AT 15,7; "PRINT AND ";

A$ = "ADD"

PRINT AT *; A$ + " MORE";

PRINT "PRINT ON BOTTOM AND SCROLL"
```

How to Install

Once I developed this program I needed an easy way to install it on a system. I considered and rejected making a tape that the monitor could read. It would be difficult to modify, error-prone on input, and would work only if loading to a fixed absolute address. I did not want to use a BASIC program that POKEd in several DATA statements of decimal values since I think in hex when programming in assembly. For this reason I created a BASIC program that reads hex strings, converts them to binary, and loads them into memory. To be adaptable it calculates a starting load address from the size of the program and the address of the top of memory.

Enter the program shown in listing 1, save it to tape, and then run it. After it is through loading (about 15 seconds) it will print "PRINT AT HAS BEEN LOADED... AND IT WORKS !!!" across several lines of the screen. Then you may type NEW and enter or LOAD any program you like using PRINT AT.

If an error occurs in the middle of a PRINT AT statement the "AT flag" can be turned off by typing any valid BASIC statement (i.e., LIST or "?" for PRINT, etc.) at the keyboard.

Relative Hexadecimal Loader

The loader reads strings from data statements and loads a program into high memory. The program consists of four parts:

Program size:
A four-digit hex number. This value is subtracted from the end-of-memory address at $0085 to get the starting address for the program.

Listing 2: Assembly Listing of PRINT AT Routine

```
0000            ;*******************
0000            ;*                 *
0000            ;*     PRINT AT    *
0000            ;*                 *
0000            ;*   BY MATT ASAY  *
0000            ;*                 *
0000            ;*******************
0000            ;
0000             ATFLG=$CC          STATUS BYTE FOR 'PRINT AT'
0000             ASTOK=$A5         '*' TOKEN FOR MULTIPLICATION
0000            CHRGET=$00BC        GET NEXT CHAR IN BASIC LINE
0000            CHRGOT=$00C2        GET SAME CHAR AGAIN
0000             PRTOK=$97          TOKEN FOR PRINT COMMAND
0000            ;
2100                   *=$2100
2100 A957       USRX   LDA #PSPLIC*256/256 USR INITIALIZATION
2102 A021              LDY #PSPLIC/256
2104 8581              STA $81      RESERVE MEMORY FOR SPLICES
2106 8482              STY $82
2108 8583              STA $83
210A 8484              STY $84
210C 8585              STA $85
210E 8486              STY $86
2110 A207              LDX #7       PUT SPLICE INTO PARSER
2112 BD4F21     USRX1  LDA PATCH,X
2115 95C5              STA $C5,X
2117 CA                DEX
2118 10F8              BPL USRX1
211A AD1A02            LDA $021A    GET OLD OUTPUT VECTOR
211D AC1B02            LDY $021B
2120 8DE621            STA OS.0+1   STORE INTO OUTPUT SPLICE
2123 8CE721            STY OS.0+2
2126 A9E1              LDA #OSPLIC*256/256 SPLICE INTO OUTPUT
2128 A021              LDY #OSPLIC/256
212A 8D1A02            STA $021A
212D 8C1B02            STY $021B
2130 AD1C02            LDA $021C    GET OLD CTRL-C VECTOR
2133 AC1D02            LDY $021D
2136 8DFB21            STA CS.0+1   STORE INTO CTRL-C SPLICE
2139 8CFC21            STY CS.0+2
213C A9F6              LDA #CSPLIC*256/256 SPLICE INTO CTRL-C
213E A021              LDY #CSPLIC/256
2140 8D1C02            STA $021C
2143 8C1D02            STY $021D
2146 A988              LDA #$88     RESTORE DEFAULT USR VECTOR
2148 A0AE              LDY #$AE
214A 850B              STA $0B
214C 840C              STY $0C
214E 60                RTS
214F            ;
214F            ;****************************************
214F C920       PATCH  CMP #$20     PATCH PUT AT $C5-$CC
2151 F0F3              BEQ *-11
2153 4C5721            JMP PSPLIC
2156 00         .BYTE 0             ATFLG AT $CC
2157            ;BIT 0 SET- PRINT TOKEN FOUND ON LAST FETCH
2157            ;BIT 1 SET- 'PRINT AT' CURRENTLY ACTIVE
2157            ;****************************************
```

(continued)

Non-relocatable hex data:
  A string of any number of bytes in hex form.

Relocatable addresses:
  A prefix character R, H, or L followed by two or four hex characters. The hex number is added to the starting address of the program. The resulting address is stored as follows:
    R: Store both bytes (low, high form)
    H: Store high byte
    L: Store low byte

End of program marker:
  Any single character ("*" is used here).

You can use the loader program for your own machine-language routines. Use lines 1-58 as shown. Replace 100-999 with DATA statements for your code in the format shown. When the program has finished loading it will jump to 1000 with R0 set to the starting load address. Your statements here should protect your program and perform any other initialization needed.

How the Program Works

The program has four parts: a USR call for initialization, "splices" into the BASIC parse, output, and control-C routines. The USR routine changes the top of memory address to protect the permanent part of the program (not including this initialization). It patches the other three pieces into their respective vectors. The code at line 1000 saves and restores the previous USR address, so this routine can be loaded after another USR routine without messing it up.

The second piece is spliced into the parse routine at $BC-$D3. This routine fetches the program for the BASIC interpreter a character/token at a time. When not in a PRINT statement this routine works normally; otherwise it checks for an AT following the PRINT token. If it is found, the routine collects and interprets the location specification. It then returns the character following the first semicolon to the print routine as if the "AT location;" had not been there.

The third piece is spliced into the output vector. Any time the "AT flag" (bit 1 of $CC) is on, instead of going to the normal print routine it outputs to the current screen location and then increments the location. It then decrements the character count (which the routine that calls it increments) to prevent overflow and returns to the caller.

Listing 2 *(continued)*

```
2157              ;
2157              ;******PARSER SPLICE******
2157  24CC        PSPLIC BIT ATFLG    PRINT TOKEN FOUND?
2159  1014               BPL SPL1     BRANCH IF NOT
215B  C941               CMP #'A      CHECK FOR 'AT'
215D  D00E               BNE SPL0     BRANCH IF NOT FOUND
215F  48                 PHA          SAVE A & Y REGISTERS
2160  98                 TYA
2161  48                 PHA
2162  A001               LDY #1
2164  B1C3               LDA ($C3),Y
2166  C954               CMP #'T      NO BLANKS ALLOWED BETWEEN A&T
2168  F013               BEQ PR.AT    BRANCH IF 'AT' FOUND
216A  68                 PLA          RESTORE A & Y
216B  A8                 TAY
216C  68                 PLA
216D  06CC        SPL0   ASL ATFLG    CLEAR 'PRINT FOUND' BIT
216F  C997        SPL1   CMP #PRTOK   IS CHAR A PRINT TOKEN?
2171  D002               BNE SPL2     NO, BRANCH
2173  85CC               STA ATFLG    SET PRINT FOUND, CLR AT FOUND
2175  C93A        SPL2   CMP #':      SET STATUS & RETURN CHAR
2177  B003               BCS SPL3
2179  4CCD00             JMP $00CD
217C  60          SPL3   RTS
217D              ;
217D              ;******PRINT AT FOUND******
217D  46CC        PR.AT  LSR ATFLG    CLEAR PRINT FLAG, SET AT FLAG
217F  68                 PLA          RESTORE A & Y
2180  A8                 TAY
2181  68                 PLA
2182  20BC00             JSR CHRGET   SKIP OVER 'T'
2185  20BC00             JSR CHRGET   GET NEXT CHAR
2188  C9A5               CMP #ASTOK   '*' TOKEN?
218A  D006               BNE PR.A0    NO, BRANCH
218C  20BC00             JSR CHRGET   GET NEXT CHAR
218F  38                 SEC
2190  B041               BCS PR.A3    BRANCH ALWAYS
2192              ;
2192  20C1AA      PR.A0  JSR $AAC1    COLLECT EXPRESSION 1
2195  2008B4             JSR $B408    CONVERT TO INTEGER
2198  20C200             JSR CHRGOT   FOLLOWED BY COMMA?
219B  C92C               CMP #',
219D  D023               BNE PR.A2    NO, BRANCH
219F  A511               LDA $11      PUSH INT(EXPR1)*32 ON STACK
21A1  0A                 ASL A
21A2  0A                 ASL A
21A3  0A                 ASL A
21A4  0A                 ASL A
21A5  2612               ROL $12
21A7  0A                 ASL A
21A8  8511               STA $11
21AA  A512               LDA $12
21AC  2A                 ROL A
21AD  48                 PHA
21AE  A511               LDA $11
21B0  48                 PHA
21B1  20C9AA             JSR $AAC9    COLLECT 2ND EXPRESSION
21B4  2008B4             JSR $B408    CONVERT TO INTEGER
21B7  68                 PLA          ADD INT(EXPR1)*32
21B8  18                 CLC
```

(continued)

The last piece is spliced into the control-C vector. This vector is called at the end of each statement (to check if control-C is depressed). The spliced routine unconditionally resets the "AT flag" before going to the normal control-C routine. This prevents an error, control-C, or END of the program from leaving the "PRINT AT" on when control returns to the user.

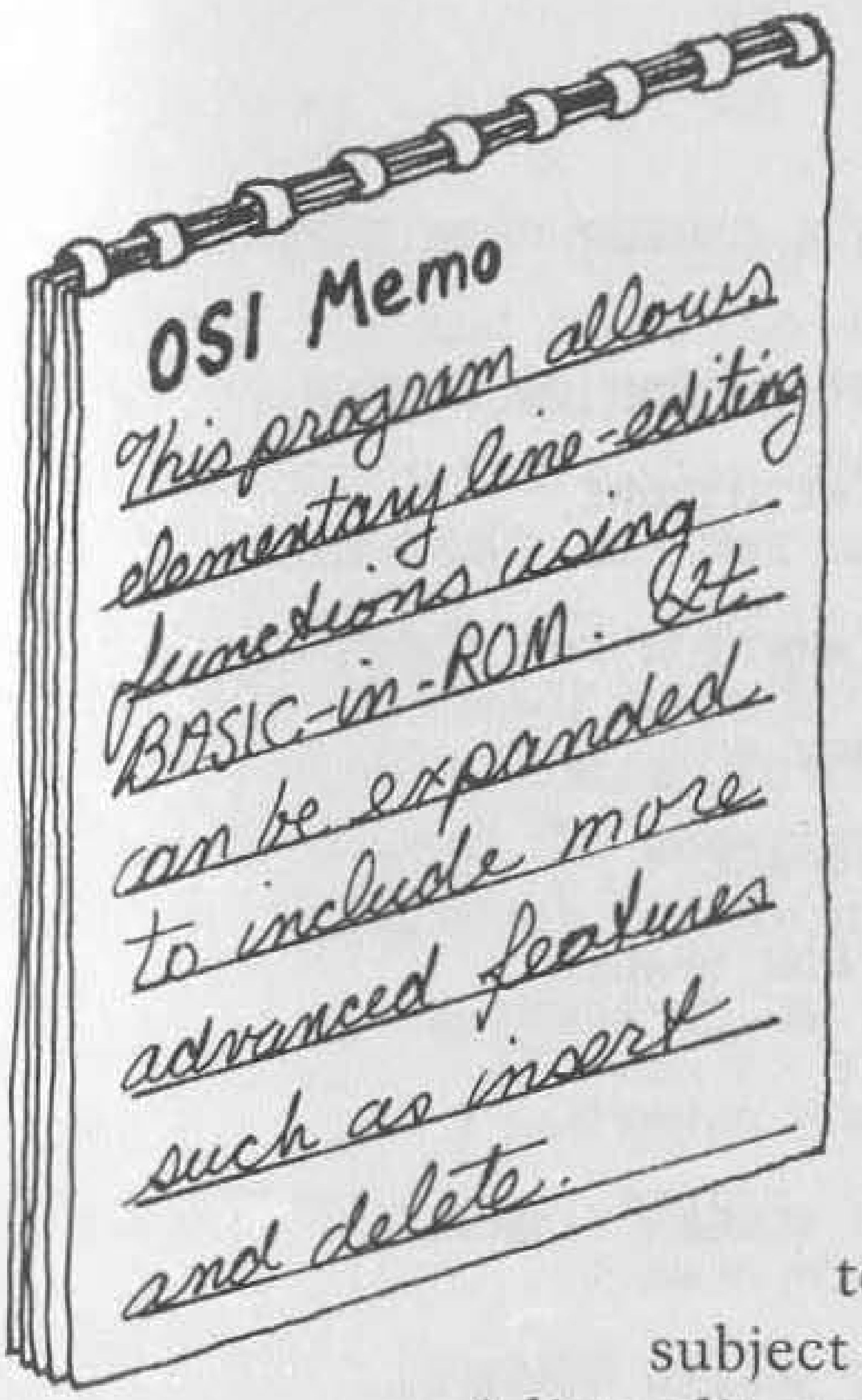
This program takes 253 bytes to load; but after initialization it requires only 166 bytes. If you wish to preserve the initialization code also, just change the "L57" in line 110 to "L00".

Listing 2 *(continued)*

```
1190 21B9 6511          ADC $11
1200 21BB 8511          STA $11
1210 21BD 68            PLA
1220 21BE 6512          ADC $12
1230 21C0 8512          STA $12
1240 21C2 A511   PR.A2  LDA $11      ADD $D000, STORE AS 'AT' LOC.
1250 21C4 8DE921        STA OS.1+1
1260 21C7 A512          LDA $12
1270 21C9 2903          AND #03
1280 21CB 09D0          ORA #$D0
1290 21CD 8DEA21        STA OS.1+2
1300 21D0 20C200        JSR CHRGOT   GET CHARACTER
1310 21D3 C93B   PR.A3  CMP #';      MUST BE SEMICOLON
1320 21D5 D003          BNE BOOBOO   ERROR IF NOT
1330 21D7 4CBC00        JMP CHRGET   GET CHAR & GOTO PRINT ROUTINE
1340 21DA        ;
1350 21DA A91C   BOOBOO LDA #28      LOAD OFFSET OF 'ST' ERR MSG.
1360 21DC 85CC          STA ATFLG    RESET 'PRINT' & 'AT' FLAGS
1370 21DE 4C4EA2        JMP $A24E    PRINT ERROR MESSAGE
1380 21E1        ;
1390 21E1        ;******OUTPUT VECTOR SPLICE******
1400 21E1 24CC   OSPLIC BIT ATFLG    'AT' FLAG SET?
1410 21E3 7003          BVS OS.1     YES, BRANCH
1420 21E5 4C0000 OS.0   JMP $0000    DO NORMAL OUTPUT & RETURN
1430 21E8 8D00D0 OS.1   STA $D000    STORE CHAR ON SCREEN
1440 21EB EEE921        INC OS.1+1   INCREMENT SCREEN ADDRESS
1450 21EE D003          BNE OS.2
1460 21F0 EEEA21        INC OS.1+2
1470 21F3 C60E   OS.2   DEC $0E      DON'T LET CHAR COUNT OVERFLOW
1480 21F5 60            RTS
1490 21F6        ;
1500 21F6        ;******CTRL-C VECTOR SPLICE******
1510 21F6 A900   CSPLIC LDA #0       END OF STATEMENT,
1520 21F8 85CC          STA ATFLG    SO RESET PRINT, AT FLAGS
1530 21FA 4C0000 CS.0   JMP $0000    DO NORMAL CTRL-C STUFF
```

Line Editor for OSI 540 Board

by Earl Morris



OSI users are painfully aware that if a mistake is discovered in the 63rd character of a BASIC line, the entire line must be retyped. I have watched in awe as PET owners zip the cursor across the screen and correct the offending character in a few keystrokes. OSI machines lack this useful feature as standard equipment. But don't despair. This article describes a software patch using the 540 video board and BASIC-in-ROM to allow line editing on OSI machines. The program provides the basic editing functions, but you can add additional features as you wish. The technique also can be applied to the C1P, subject to limitations I will discuss later.

A line editor must perform three functions: it must find the line to be edited, make the changes, and then put the line back into the BASIC program. Finding the line is easy — just LIST it. The data is then on the screen. The line editor can read a character from the screen and copy it exactly whenever a designated key is hit. If any other character is typed, that character is inserted into the new line instead of the screen character. Now comes the hard part: How do you get the line back into BASIC?

The new line must be inserted at the proper location, moving the rest of the program and refixing all the pointers. This is exactly what the BASIC input routines do. The line editor can be much simpler if BASIC can be fooled into believing that you re-typed the entire line.

First examine the BASIC input routines. After cold starting BASIC, type in the following line:

```
10ABCDE
```

Listing 1

```
0000                ;*********************************
0000                ;*                               *
0000                ;*  LINE EDITOR FOR OSI 540 BOARD *
0000                ;*                               *
0000                ;*       BY E.D. MORRIS          *
0000                ;*                               *
0000                ;*********************************
0000                ;
0240                *=$240
0240 A920                  LDA #$20    CLEAR BOTTOM OF SCREEN
0242 A280                  LDX #$80
0244 9DC0D6         CLR    STA $D6C0,X
0247 CA                    DEX
0248 10FA                  BPL CLR
024A E8             CUR    INX
024B A920           C1     LDA #$20    REMOVE CURSOR FROM SCREEN
024D 9D80D6                STA $D680,X
0250 9D82D6                STA $D682,X
0253 A95E                  LDA #$5E    PRINT CURSOR ON SCREEN
0255 9D81D6                STA $D681,X
0258 20EBFF                JSR $FFEB   GET KEYSTROKE
025B C920                  CMP #$20    SPACE BAR FOR SHORT LINE
025D F019                  BEQ COPY
025F C921                  CMP #'!     EXCLAMATION FOR SHORT LINE
0261 F010                  BEQ LONG
0263 C90D                  CMP #$0D    RETURN ?
0265 F020                  BEQ DONE
0267 C95F                  CMP #$5F    BACKSPACE ?
0269 F017                  BEQ BACK
026B C923                  CMP #'#     '#' FOR SPACE
026D D00C                  BNE WSCR    MUST BE CORRECTION
026F A920                  LDA #$20    SPACE
0271 D008                  BNE WSCR    BRANCH ALWAYS
0273                ;
0273 BD01D6         LONG   LDA $D601,X READ SCREEN (LONG)
0276 D003                  BNE WSCR    BRANCH ALWAYS
0278                ;
0278 BD41D6         COPY   LDA $D641,X READ SCREEN (SHORT)
027B 9DC1D6         WSCR   STA $D6C1,X PRINT CHAR ON SCREEN
027E 9513                  STA $13,X   STORE CHAR IN BUFFER
0280 D0C8                  BNE CUR     BRANCH ALWAYS
0282                ;
0282 CA             BACK   DEX         BACKSPACE
0283 30C5                  BMI CUR     LIMIT BACKSPACE
0285 10C4                  BPL C1      BRANCH ALWAYS
0287                ;
0287 A900           DONE   LDA #0      PUT NULL INTO BUFFER
0289 9513                  STA $13,X
028B A992                  LDA #$92    DISPLAY 'OK' MESSAGE
028D A0A1                  LDY #$A1
028F 20C3A8                JSR $A8C3
0292 A212                  LDX #$12
0294 A000                  LDY #0
0296 4C80A2                JMP $A280   BACK TO BASIC
```

If you press RETURN, this line will be entered into the BASIC text. However, instead of RETURN, press the BREAK key and jump to the machine-monitor mode. Examine the data stored at locations $0013 to $0019. You should find

Location	Data	ASCII
$0013	31	1
$0014	30	0
$0015	41	A
$0016	42	B
$0017	43	C
$0018	44	D
$0019	45	E

The data at these locations is the hex representation of the ASCII characters you just typed. Locations $0013 through $005A are the input buffer. Thus, to simulate keyboard input the line editor must store the corrected line in this buffer. The next trick is to get BASIC to accept this data. First the "X" and "Y" registers must be set to point at the input buffer and then a jump made to the proper location in BASIC.

Try the following experiment. Cold start BASIC and jump to the machine monitor. Using the monitor, fill locations $0013 to $0019 with the hex data from the above example, adding a $00 at location $001A. Again using the machine monitor, write the following program at $0250.

```
$0250 A2 12    LDX #$12
$0252 A0 00    LDY #$00
$0254 4C 80 A2JMP $A280
```

Then execute the program starting at $0250. The pointers are set to the input buffer and a jump is made into ROM. There will be no indication that anything happened, but you are now back in BASIC. Type LIST and

```
10ABCDE
```

will appear. This technique has convinced BASIC to accept a line of data stored in the input buffer as if it had been typed in. Try using this method to input other lines of data, remembering to make the final character a null or $00.

Here is the final link to writing a line editor. Listing 1 is an editor assembled at address $0240. The program assumes that the line to be edited has been listed previously and now appears on the screen starting at $D641. The line editor is called through the USR function. After clearing several screen locations, the program displays an up arrow ($5E) as a cursor immediately below the line to be edited. The subroutine at $FFEB gets a character from the keyboard. If this character is a space bar ($20), one character is copied from the old line into the input buffer and displayed on the screen below the cursor. The cursor will move backwards on a backspace or $5F input. A RETURN or $0D indicates that you are finished editing that line. Since the space bar is used for direct copying, something else must be used for a space. I have chosen the # sign or $23. Any other character typed is assumed to be corrected input and is stored in the buffer and on the screen.

The RETURN key causes the program to display "OK" and places a null at the end of the input line. The pointers are set as described above, and a jump made back into BASIC. If the program is moved to reside in a different memory location, the jump absolute instructions at lines $0282 and $0288 must be changed.

For those of you who do not use machine code, I have included a BASIC program to set up this patch and then erase itself. Once the line editor is entered, either by BASIC or *via* machine code, load the program you want to edit. Then add the following line to your BASIC program:

```
1 POKE 11,64: POKE 12,2:Z = USR(1)
```

LIST the line you want to edit, then type RUN. This calls the line editor and displays the cursor directly under the listed line. The valid commands were listed above. To run your program, either delete line one or enter RUN 10 (assuming your first line is 10). Before you save the corrected program, delete line one.

Listing 2

```
10 PRINT "LINE EDITOR FOR OSI"
20 PRINT " C1P OR SUPERBOARD"
30 FOR I=576 TO 669:READ J:POKE I,J:NEXT
40 PRINT:PRINT "EDITOR LOADED":NEW
50 DATA 169,32,141,37,211,141,38,211,162,0
60 DATA 169,32,157,5,211,157,7,211,169,94
70 DATA 157,6,211,32,235,255,201,32,240,22
80 DATA 201,13,240,34,201,95,240,26,201,35
90 DATA 208,2,169,32,157,38,211,149,19,76
100 DATA 124,2,189,230,210,157,38,211,149,19
110 DATA 232,76,74,2,202,76,74,2,169,79
120 DATA 141,69,211,169,75,141,70,211,169,32
130 DATA 141,71,211,169,0,149,19,162,18,160
140 DATA 0,76,128,162
```

Now for the limitations of this simple editor. The line to be corrected must appear at a fixed position on the video screen. This is determined by the screen read instruction LDA $D641,X. The editor will not work if the line is not exactly at this position. For example, if a line is longer than 64 characters, the screen will scroll, moving the text up one line. A similar problem occurs when you attempt to edit the last line of a program — the listed line appears too low on the video screen. In this case, simply hit a RETURN to scroll up one line and then type RUN to enter the editor.

Lines longer than 64 characters can be edited by changing the screen read instruction from LDA $D641,X to LDA $D601,X. This is accomplished by using different keys for the "copy" function, depending on the length of the line being edited. Lines shorter than 64 characters are copied by pressing the space bar. Longer lines are copied with the exclamation (!) key.

This editor can be modified to run on a C1P or Superboard by changing the appropriate screen locations. A BASIC listing of a C1P version is shown in listing 2. The editor is limited to a single video line, which, in the case of the C1P, is only 25 characters. To edit multiple lines, the editor must be able to skip over the unused bytes on the edges of the C1P video screen.

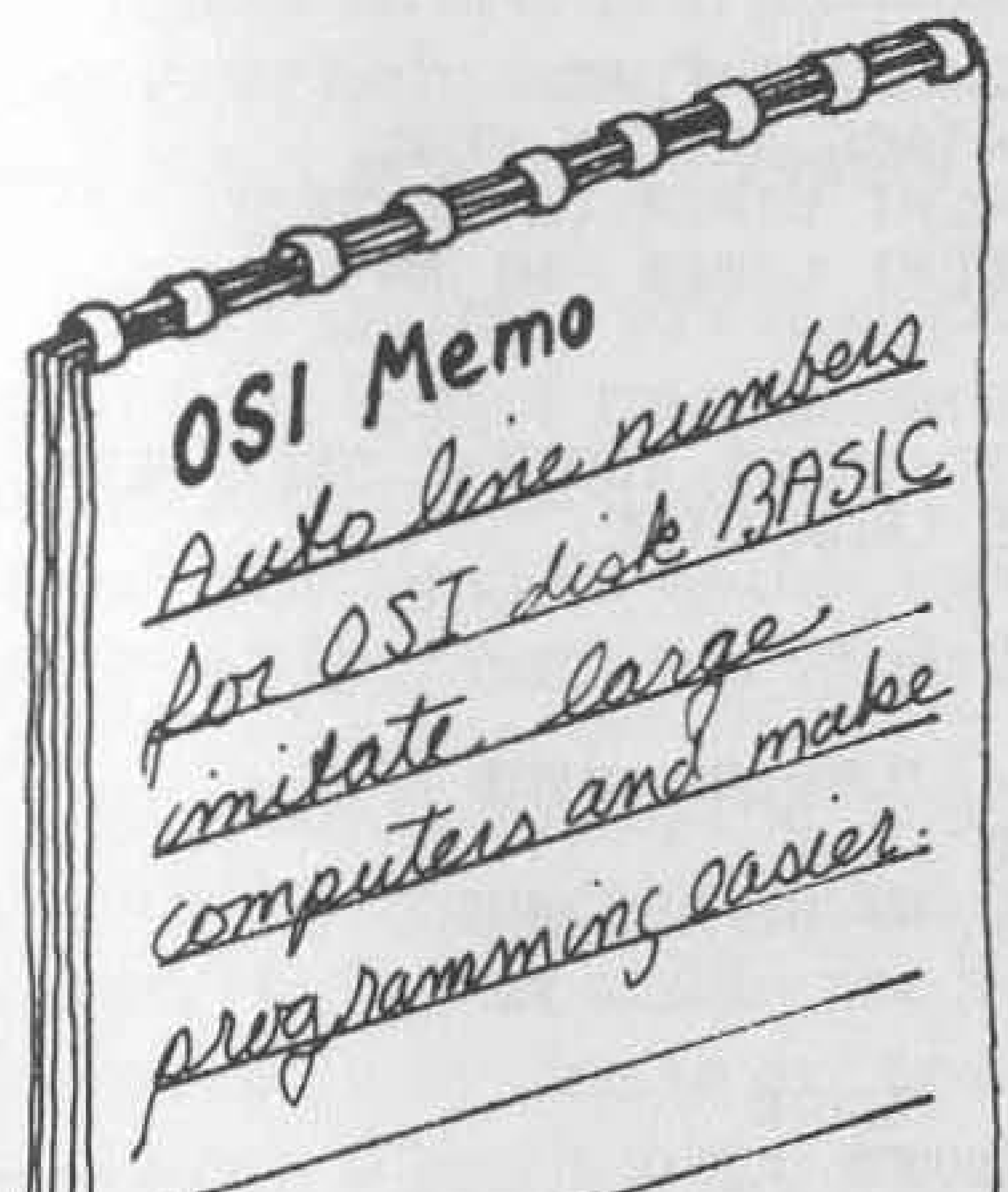
Listing 3 is the source code for 65D3.2. Assemble the program somewhere (for example at $XXXX) and go back into BASIC. The editor is set up by DISK!"GO XXXX". This set-up POKEs the word "EDIT" in place of "WAIT" in the instruction table and changes the dispatch table to point to the edit routine. The first NOP must be left due to the way the dispatch table works. LIST a line, then call the editor by entering EDIT. Otherwise this routine works the same as the 540 ROM version.

Listing 3

```
 10 0000            ;******************************
 20 0000            ;* LINE EDITOR 65D 3.2 VERSION *
 30 0000            ;*                            *
 40 0000            ;*        BY EARL MORRIS      *
 50 0000            ;******************************
 60 0000            ;
 70 0000                    ;JUMP HERE TO SET UP EDITOR BY "EDIT"
 80 E800            *=$E800
 90 E800 A945       SET     LDA #'E
100 E802 8DC902             STA $02C9
110 E805 A944               LDA #'D
120 E807 8DCA02             STA $02CA
130 E80A A915               LDA #GO*256/256
140 E80C 8D2402             STA $0224
150 E80F A9E8               LDA #GO/256
160 E811 8D2502             STA $0225
170 E814 60                 RTS
180 E815            ;
190 E815                    ;START OF EDIT ROUTINE
200 E815             BUFF=$1B
210 E815 EA         GO      NOP
220 E816 A920               LDA #$20
230 E818 A280               LDX #$80
240 E81A 9DC0D6     CLR     STA $D6C0,X CLEAR SCREEN BOTTOM
250 E81D CA                 DEX
260 E81E 10FA               BPL CLR
270 E820 A200               LDX #0
280 E822 A920       CUR     LDA #$20    REMOVE CURSOR
290 E824 9D40D6             STA $D640,X
300 E827 9D42D6             STA $D642,X
310 E82A A95E               LDA #$5E    CURSOR
320 E82C 9D41D6             STA $D641,X PLACE CURSOR
330 E82F 20EDFE             JSR $FEED   GET KEYSTROKE
340 E832 C920               CMP #$20    SPACE BAR FOR SHORT LINE
350 E834 F019               BEQ COPY
360 E836 C921               CMP #'!     EXCLAMATION FOR LONG LINE
370 E838 F010               BEQ LONG
380 E83A C90D               CMP #$D     RETURN?
390 E83C F022               BEQ DONE
400 E83E C95F               CMP #$5F    BACKSPACE?
410 E840 F019               BEQ BACK
420 E842 C923               CMP #'#     # FOR SPACE
430 E844 D00C               BNE WSCR    MUST BE CORRECTION
440 E846 A920               LDA #$20    SPACE
450 E848 D008               BNE WSCR    BRANCH ALWAYS
460 E84A BDC1D5     LONG    LDA $D5C1,X READ SCREEN (LONG)
470 E84D D003               BNE WSCR    ALWAYS
480 E84F BD01D6     COPY    LDA $D601,X READ SCREEN (SHORT)
490 E852 9D81D6     WSCR    STA $D681,X WRITE SCREEN
500 E855 951B               STA BUFF,X  INPUT BUFFER
510 E857 E8         L1      INX
520 E858 4C22E8             JMP CUR
530 E85B CA         BACK    DEX         BACKSPACE
540 E85C 30F9               BMI L1      LIMIT BACKSPACE
550 E85E 10C2               BPL CUR
560 E860 A900       DONE    LDA #0      NULL INTO BUFFER
570 E862 951B               STA BUFF,X
580 E864 A992               LDA #$92
590 E866 A003               LDY #3
600 E868 200300             JSR $0003   DISPLAY 'OK' MESSAGE
610 E86B A21A               LDX #$1A
620 E86D A000               LDY #0
630 E86F 4C8004             JMP $0480   BACK TO BASIC
```

Auto Line Numbers for OSI Disk BASIC

by Lester Cain



Software support for the OSI is improving but is still minimal, and users have to develop many of their own programs. Actual programming with flow charts and algorithms is part of the pleasure of developing your own program. But when it's time to input to the machine some of the fun flies out the window. With all the necessary keying, line numbers are an added detriment and detract from the pleasure of writing programs.

Some of you are familiar with large mainframe computers, which have an AUTO function and put out line numbers for you. This function is definitely a plus and should be available to everyone. I explain here a simple, easy-to-use program that gives you an AUTO function to relieve some of the tedious burden of typing. There are two listings — one in assembly language and the other in BASIC, which should work on the C1P disk BASIC also. The logic is easy to follow and could be put to use on ROM machines with different hooks. But I will leave that as an exercise for persons with ROM.

Listing 1 is the assembly-language routine necessary to develop the program. In OSI disk BASIC, the routine to get a character from the keyboard and incorporate it into the BASIC Source begins at $558, which is LDX #$0. At the next address, or $55A, there is a hook to make BASIC jump to the AUTO program. This is accomplished in line 310 of listing 2 and forces information to go through the code before BASIC can do anything with the keyboard information.

Now you are at routine START in the assembly routine. Since there is a hook here to make BASIC jump, you will have to perform the routine

Listing 1

```
0000             ;* AUTO LINE NUMBERS *
0000             ;*  FOR OSI C1P-C8P  *
0000             ;*  WITH DISK BASIC  *
0000             ;*                   *
0000             ;*    BY LES CAIN    *
0000             ;*                   *
0000             ;
0000               SCL=$6C          CURSOR'S HOME POINTER (LO)
0000               SCH=$6D          CURSOR'S HOME POINTER (HI)
0000               BUF=$1A          START OF BASIC BUFFER
0000             ;
0000             BASIC=$055D        INPUT EXIT POINT
0000             INPUT=$0587        BASIC INPUT ROUTINE
0000              LINE=$1CDC        HEX-DECIMAL CONVERT ROUTINE
0000             ;
0000                TH=$D8          AUTONUMBER FLAG
0000                FH=TH+1         CARRIAGE RETURN FLAG
0000                LO=TH+2         CURRENT LINE# (LO BYTE)
0000                HI=TH+3         CURRENT LINE# (HI BYTE)
0000             ;
8000             *=$8000
8000 208705      START   JSR INPUT  BASIC INPUT ROUTINE SENT HERE
8003 48                  PHA        SAVE CHARACTER
8004             ;
8004 C906        AUTON   CMP #6     CTRL F ?
8006 D006                BNE AUTOFF
8008 A900                LDA #0     YES, TURN ON AUTO
800A 85D8                STA TH     AUTO FLAG
800C 85D9                STA FH     FLAG TO BYPASS AUTO
800E             ;
800E C91B        AUTOFF  CMP #$1B   ESC
8010 D004                BNE BACK   TEST FLAGS
8012 E6D8                INC TH     TURN OFF AUTO FLAGS
8014 E6D9                INC FH
8016 A5D8        BACK    LDA TH     GET AUTO FLAG
8018 D004                BNE BK     NOT A 0 - BACK TO BASIC
801A A5D9                LDA FH     CR FLAG MADE 0 WITH A CR
801C F004                BEQ AUTO   IF 0 THEN CONTINUE WITH LINE #
801E 68          BK      PLA        RESTORE SAVED CHARACTER
801F 4C5D05      BK1     JMP BASIC  BACK TO BASIC WITH CHAR
8022             ;
8022 68          AUTO    PLA        PULL SAVED CHAR FROM STACK
8023 A940                LDA #$40   LO BYTE OF SCREEN ADDRESS
8025                                ;**** #$65 FOR C1P ****
8025 856C                STA SCL    INITIALIZE POINTER LO BYTE
8027 A9D7                LDA #$D7   HI BYTE OF SCREEN ADDRESS
8029                                ;**** #$D3 FOR C1P ****
8029 856D                STA SCH    INITIALIZE POINTER HI BYTE
802B A5DA                LDA LO     LO BYTE OF LINE #
802D 18                  CLC
802E 690A                ADC #10    ADD LINE INCREMENT
8030 85DA                STA LO     SAVE LO BYTE
8032 9002                BCC ASOUT  SKIP INCR HI IF NO CARRY
8034 E6DB                INC HI     INCREMENT HI IF NECESSARY
8036 A6DA        ASOUT   LDX LO     GET LO BYTE OF LINE #
8038 A5DB                LDA HI     GET HI BYTE OF LINE #
803A             ;
803A             ;CONVERTS BINARY NUMBER TO ASCII STRING & OUTPUTS
```

(continued)

that was originally there, getting a key from the keyboard. At AUTON you test for a control 'F'. If this key is encountered, the two Auto flags are set to zero and the program will fall through to the AUTO routine. If there is no control 'F', then test for an ESC at AUTOFF. If there is an ESC, turn off Auto flags TH and FH and go back to BASIC with the character in the accumulator. If no ESC is found, test Auto flag TH. If TH is not zero then test the secondary flag FH. This flag is turned off in the SCR routine so constant line numbers are not output. If FH is zero then you are ready for a new line number and fall through to the AUTO routine.

AUTO is a simple addition and increments the line number by 10 at every pass. AUTO also initializes the indirect screen pointers. This needs to be done only once, but why take chances? BASIC might decide to stick something at these addresses.

One of the keys to the whole program is the ASOUT routine. The line number is loaded into the accumulator and the X index. A JSR to the BASIC routine LINE ($1CDC) outputs an ASCII string from the binary values in LO and HI to the screen at cursor level. BASIC uses this routine to output line numbers when listing.

This brings you to the most important segment of the program — getting BASIC to accept the line number you have created. It must be in an acceptable format and in the input buffer. Use the Y index for LINE, and decrement it by one to get you to the cursor. Here storage is started into the buffer. After the line number is in, the X index is decremented and you write on top of the cursor with a space. BASIC uses X to point into the buffer. From here it's back to the keyboard with a space after the last digit of the line number. Here you also turn off the CR flag FH, simply by incrementing it.

Now for the last segment of the assembly program — the CR routine. You have put a hook into BASIC with the statement in line 270 of listing 2. BASIC jumps here when it finds a carriage return. Turn to the back of flag FH; if the main Auto flag TH is on, the AUTO process continues until an ESC turns off both flags. To end the program, jump to $A6D. This puts the buffer pointer into the CHARGET routine and checks the syntax to determine if what you just did was an immediate command or a line number. Since it is a line number, all pointers will be reset and the line is entered into the BASIC Source.

The BASIC program as shown is all that is necessary to have the AUTO function on your system. Line 170 determines the highest page of RAM on your system and sets the high end of BASIC work space to protect the object code. Statement 220 POKEs the code into the appropriate area of memory by reading the data and POKEing it to I. Statement 270

Listing 1 *(continued)*

```
803A             ;USED TO OUTPUT LINE #'S WHEN LISTING
803A             ;
803A 20DC1C            JSR LINE     BASIC ROUTINE FOR THE LINE #
803D 98                TYA          GET Y-REG FROM OUTPUT ROUTINE
803E AA                TAX          SAVE IT IN X-REG
803F 48               PHA
8040 88                DEY          BYPASS SPACE AFTER CURSOR
8041             ;
8041 B16C        SCR   LDA (SCL),Y GET CHARACTER FROM SCREEN
8043 991A00            STA BUF,Y    PUT IT IN BASIC BUFFER-1
8046 88                DEY
8047 D0F8              BNE SCR      NOT AT END OF LINE# ON SCREEN
8049 68                PLA          GET Y-REG BACK
804A A8                TAY          RESTORE Y FOR DISPLAY PURPOSES
804B CA                DEX          BYPASS CURSOR, X IS BUFFER IND
804C E6D9              INC FH       TURN OFF CR FLAG
804E A920              LDA #$20     LOAD A SPACE
8050 D0CD              BNE BK1      TO BASIC WITH SPACE IN ACC.
8052             ;
8052             ;PATCH FROM BASIC POKES TO RESTORE AUTO FLAG
8052             ;AFTER A CR IS RECEIVED BY INPUT ROUTINE
8052             ;
8052 A900        CR    LDA #0       TURN AUTO FLAG BACK ON
8054 85D9              STA FH       SET CR FLAG
8056 4C6D0A            JMP $0A6D    BACK TO BASIC ADDRESS PATCHED
```

puts in the intercept jump to reset the secondary Auto flag. Statement 310 puts the hook for getting characters into the original BASIC routine, for the test routine. Since the machine code is completely relocatable, the only variable is P, which BASIC puts in 8960 on boot in, indicating the highest page in RAM.

The REM statement in the data indicates the location of the beginning line number. This can be changed if you don't want to start a line number as 100.

The listings included here allow you to choose how you want to implement the AUTO routine. The assembly method can be used in the free area before BASIC workspace on the mini-disks. A note of caution: some of the new software has a revised keyboard routine in this area. This way the program is available all the time and not used as free RAM. Or, the BASIC program could be run from BEXEC*. The BASIC listing was made using the AUTO function.

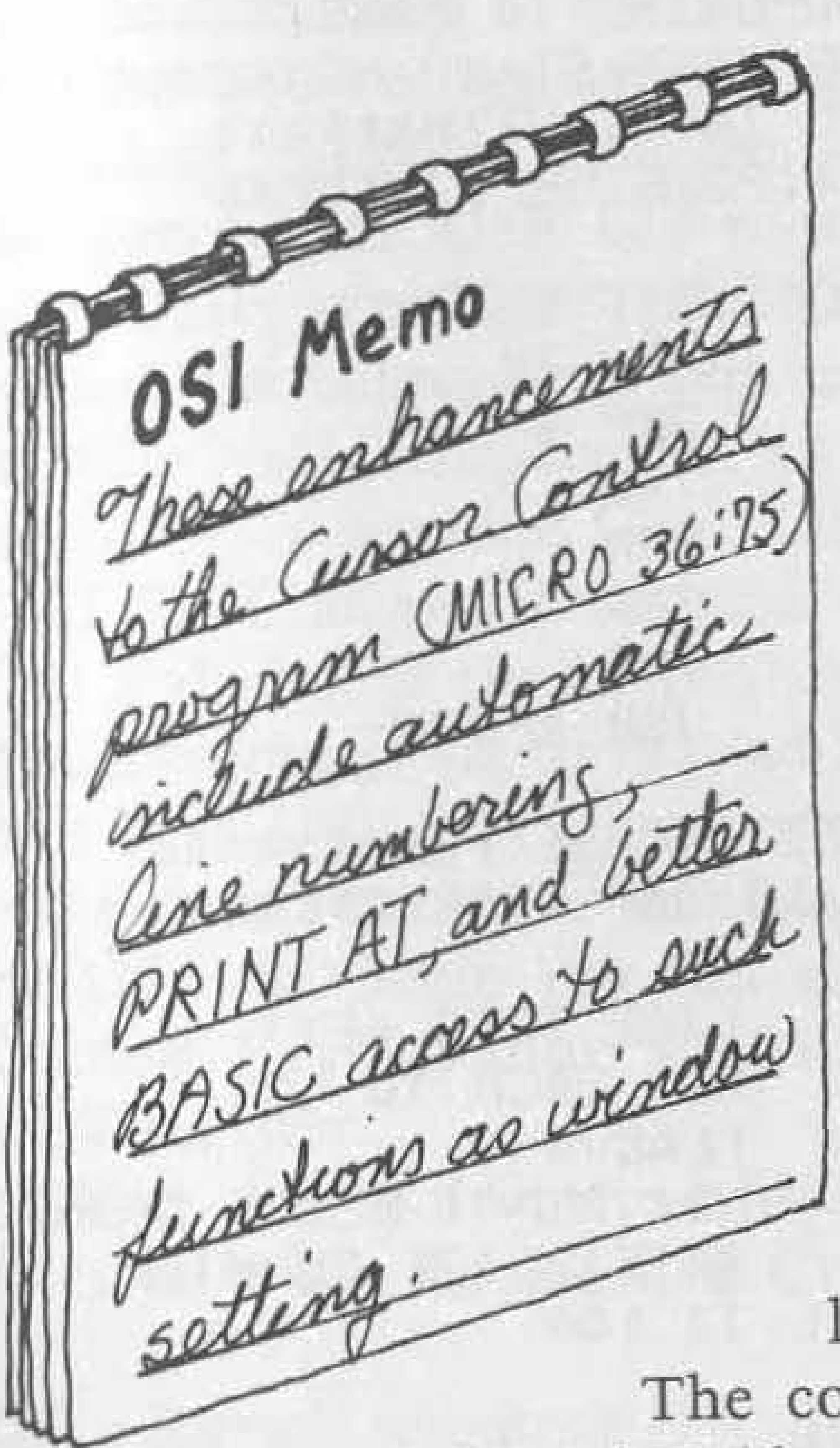
A few words here on using the finished program: the two flags are turned off at first and must be turned on with a Control-F. After the program is on, it will continue to output line numbers until it encounters an ESC. The ESC can be either in the line or before another line is output. Simply press the space bar to continue after each carriage return. This is certainly more convenient than typing in line numbers!

Listing 2

```
10 REM AUTO LINE NUMBERS
20 REM FOR OSI 1P-8P DISK SYSTEMS
30 REM WORKS FOR ANY SIZE MEMORY
40 REM
50 REM POKE NEW HIGH MEMORY TO SAVE CODE
60 S=PEEK(8960):POKE 132,143:POKE 133,S: RUN 70
70 P=PEEK(8960)
80 REM
90 REM X IS BEGIN ADDRESS TO POKE CODE
100 X=P*256+144:FORI=X TO X+88:READ A:POKE I,A:NEXT
110 REM
120 REM POKE A JUMP TO MACHINE CODE AT $0584
130 REM P IS THE HIGH BYTE
140 POKE 1412,76:POKE 1414,P:POKE 1413,226
150 REM
160 REM POKE JUMP TO MACHINE CODE AT $055A
170 POKE 1370,76:POKE 1371,144:POKE 1372,P
180 REM
190 PRINT:PRINT"READY":PRINT
200 REM
210 REM SET BEGINNING LINE = TO 90
220 POKE 218,90:POKE 219,0
230 REM
240 REM DATA FOR MACHINE LANGUAGE CODE
250 DATA 32,135,5,72,201,6,208,6,169,0,133,216,133,217,201,27
260 DATA 208,4,230,216,230,217,165,216,208,4,165,217,240,4,104,76
270 DATA 93,5,104,169
280 DATA 64:REM CHANGE TO 101 FOR C1P
290 DATA 133,108,169
300 DATA 215:REM CHANGE TO 211 FOR C1P
310 DATA 133,109,165,218,24,105
320 DATA 10:REM THIS IS THE AUTONUMBER INCREMENT
330 DATA 133,218,144,2,230,219,166,218,165,219,32,220,28,152,170,72
340 DATA 136,177,108,153,26,0,136,208,248,104,168,202,230,217,169,32
350 DATA 208,205,169,0,133,217,76,109,10
```

Autonumber Plus for Cursor Control

by Kerry Lourash



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

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

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

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

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

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

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

Listing 1

```
0000               ;**********************************
0000               ;*  AUTONUMBER FOR CURSOR CONTROL  *
0000               ;*                                 *
0000               ;*         BY KERRY LOURASH        *
0000               ;**********************************
0000               ;
0000               COUNTL=$F1          AUTO COUNTER LO BYTE
0000               COUNTH=$F2          AUTO COUNTER HI BYTE
0000                 FLAG=$206         AUTO RESET FLAG
0222                    *=$222
0222               ;
0222               ;******ADD THIS ROUTINE TO MAKE******
0222               ;******AUTONUMBER FREE-STANDING******
0222               ;******PATCH INTO INPUT WITH******
0222               ;******POKE536,34:POKE 537,2******
0222               ;
0222              ;0222 2C0302  INPUT  BIT $203
0222              ;0225 1003           BPL IN
0222              ;0227 4CBFFF         JMP $FFBF
0222              ;022A 8A      IN     TXA
0222              ;022B 48             PHA
0222              ;022C 98             TYA
0222              ;022D 48             PHA
0222              ;022E 2000FD         JSR $FD00
0222               ;
0222               ;******CHANGE 'QUIT' CODE FROM******
0222               ;******JMP $1E12 TO JMP $FDB7******
0222               ;
0222 C90A          AUTONM CMP #$A      LINE FEED KEY?
0224 D03D                 BNE  QUIT    NO, BACK TO CC
0226 AE0602               LDX  FLAG    FLAG=0 ?
0229 D008                 BNE  ZERO    NO, DON'T RESET COUNTER
022B A964                 LDA #100     INITIALIZE COUNTER
022D 85F1                 STA  COUNTL  TO 100
022F A900                 LDA #0
0231 85F2                 STA  COUNTH
0233 A900          ZERO   LDA  #0
0235 8D0602               STA  FLAG
0238 A6F1                 LDX  COUNTL
023A A5F2                 LDA  COUNTH
023C 205EB9               JSR  $B95E   PRINT A LINE #
023F 20E0A8               JSR  $A8E0   PRINT A SPACE
0242 A2FF                 LDX #$FF
0244 E8            LOOP   INX          PUT LINE # IN BUFFER
0245 BD0101               LDA  $101,X  GET DIGIT
0248 9513                 STA  $13,X   PUT DIGIT IN BUFFER
024A D0F8                 BNE  LOOP
024C 18            INCRMT CLC          INCREMENT AUTO COUNTER
024D A90A                 LDA #10      BY 10
024F 65F1                 ADC  COUNTL
0251 85F1                 STA  COUNTL
0253 9002                 BCC  DONE
0255 E6F2                 INC  COUNTH
0257 8E0602        DONE   STX  FLAG    SET FLAG
```

(continued)

screen. The space after the line number is printed by another BASIC-in-ROM routine.

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

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

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

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

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

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

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

PRINAT - Print at (4)

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

To make window setting easier, I developed:

WINSET - Set window boundaries (5)

Listing 1 *(continued)*

```
025A 68              PLA             PULL BUFFER INDEX (X)
025B A8              TAY             FROM STACK AND REPLACE
025C 68              PLA             WITH NEW CHAR COUNT
025D 8A              TXA
025E 48              PHA
025F 98              TYA
0260 48              PHA
0261 A901            LDA #1          NON-PRINTING CHAR
0263 4C121E  QUIT    JMP $1E12       BACK TO CURSOR CONTROL
```

Listing 2

```
0000         ;********************
0000         ;*  BASIC ACCESS TO  *
0000         ;*  CURSOR CONTROL   *
0000         ;********************
0000         ;
0000         CURSOR=$E0
0000         ALTWIN=$E6
0000          PATCH=$1E0F
0000         ESCAPE=$1E5C
0000           HOME=$1E72
0000         RUBOUT=$1E80
0000         PCURSR=$1F14
0000          PRINT=$1F1F
0000         ;
0264                *=$0264
0264         ;
0264 2008B4  USRGO   JSR  $B408    CONVERT TO 2-BYTE No.
0267 C900            CMP  #0       IS HI BYTE=0?
0269 F010            BEQ  ESC      YES, TO CC SUBS
026B 6C1100          JMP  ($11)    JUMP TO ADDRESS
026E         ;
026E 201AA7  CLR     JSR  $A71A    FIND END OF LINE
0271 C8              INY           PLUS 1
0272 98              TYA
0273 18              CLC           UPDATE PARSER POINTER
0274 65C3            ADC  $C3
0276 9002            BCC  CL1
0278 E6C4            INC  $C4
027A 60      CL1     RTS
027B         ;
027B 88      ESC     DEY           SWITCH WINDOWS
027C D005            BNE  RUB
027E 48              PHA
027F 48              PHA
0280 4C601E          JMP  ESCAPE+4
0283         ;
0283 88      RUB     DEY           CLEAR WINDOW
0284 D005            BNE  HOM
0286 48              PHA
```

(continued)

The command format is X = USR(5) top boundary, bottom boundary. The boundaries are expressed as line numbers: 1 = top to 32 = bottom. See figure 2 in the Cursor Control article for a map of the window lines. A typical command is: X = USR(5)24,30. This command sets the alternate window to the bottom quarter of the screen. To use the window, call the ESC routine: X = USR(1).

CLR Subroutine

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

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

Listing 2 *(continued)*

```
0287 48              PHA
0288 4C841E          JMP  RUBOUT+4
028B          ;
028B 88       HOM    DEY            HOME CURSOR
028C D005            BNE  PRINAT
028E 48              PHA
028F 48              PHA
0290 4C6F1E          JMP  HOME-3
0293          ;
0293 88       PRINAT DEY            PRINT AT
0294 D016            BNE  WINSET
0296 201F1F          JSR  PRINT     ERASE CURSOR
0299 20C1AA          JSR  $AAC1     GET OFFSET
029C 2008B4          JSR  $B408     CONVERT TO 2-BYTE #
029F 84E0            STY  CURSOR    ADD OFFSET TO $D000
02A1 18              CLC
02A2 69D0            ADC  #$D0
02A4 85E1            STA  CURSOR+1
02A6 20141F          JSR  PCURSR  PRINT CURSOR
```

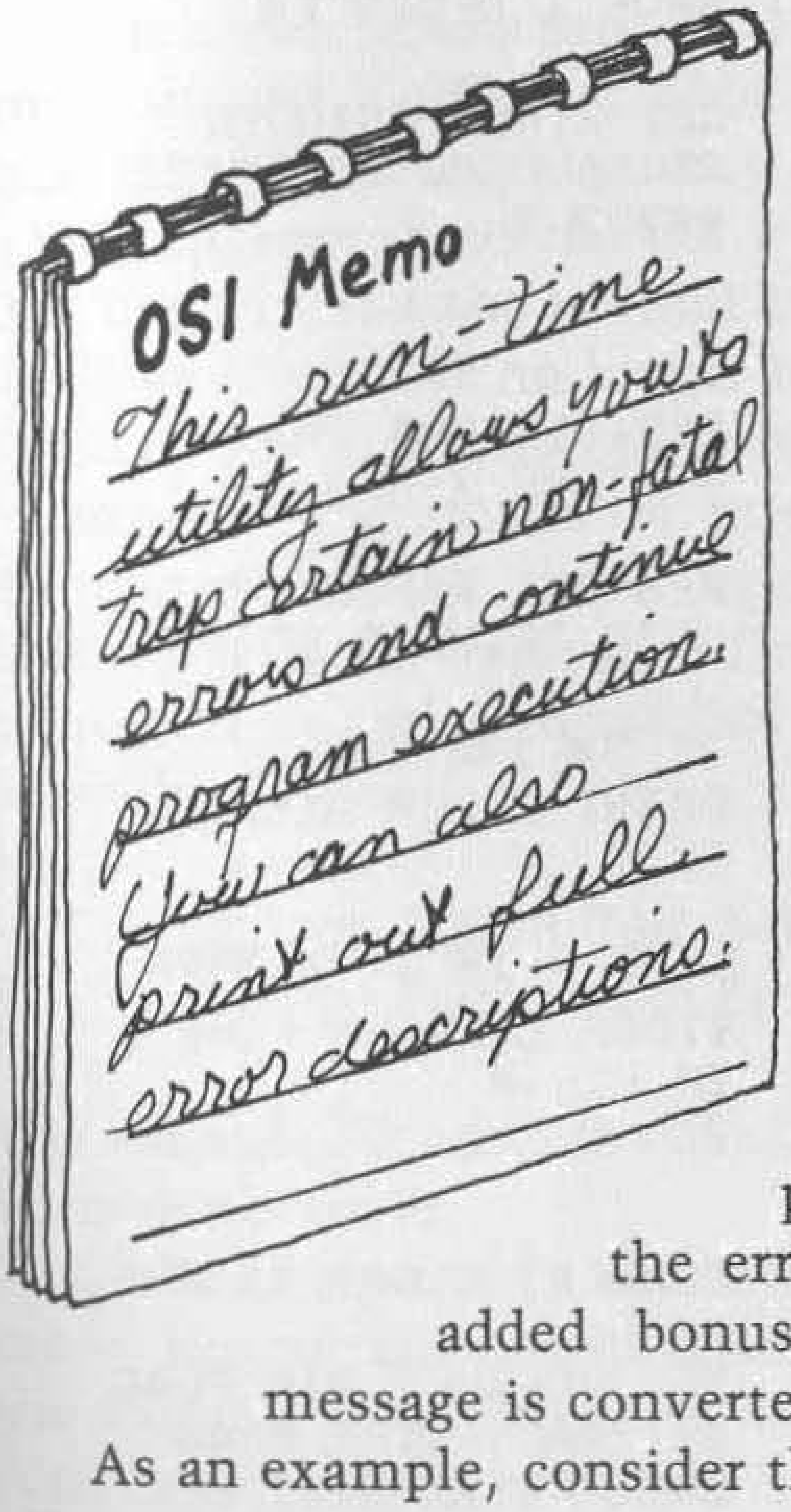
(continued)

Listing 2 *(continued)*

```
590 02A9  4C6E02             JMP  CLR      GOTO END OF LINE
600 02AC            ;
610 02AC  88        WINSET  DEY            SET ALT. WINDOW
620 02AD  D032              BNE  ERR
630 02AF  20C302            JSR  WINGET+3 GET START OF WINDOW
640 02B2  20D502            JSR  STOR       STORE IT
650 02B5  20C002            JSR  WINGET     GET END OF WINDOW
660 02B8  A202              LDX  #2
670 02BA  20D502            JSR  STOR      STORE IT
680 02BD  4C6E02            JMP  CLR       TO END OF LINE
690 02C0            ;
700 02C0  2001AC    WINGET  JSR  $AC01     FIND COMMA ELSE ERROR
710 02C3  20C1AA            JSR  $AAC1     GET VALUE
720 02C6  2005AE            JSR  $AE05     CONVERT TO 2-BYTE #
730 02C9  C6AF              DEC  $AF       MINUS 1
740 02CB  A205              LDX  #5        #6 FOR 2K CONVERSIONS
750 02CD  06AF      W1      ASL  $AF       MULTIPLY BY 32
760 02CF  26AE              ROL  $AE
770 02D1  CA                DEX
780 02D2  D0F9              BNE  W1
790 02D4  60                RTS
800 02D5            ;
810 02D5  A5AF      STOR    LDA  $AF         STORE WINDOW VALUES
820 02D7  95E6              STA  ALTWIN,X
830 02D9  18                CLC
840 02DA  A9D0              LDA  #$D0
850 02DC  65AE              ADC  $AE
860 02DE  95E7              STA  ALTWIN+1,X
870 02E0  60                RTS
880 02E1  4C88AE    ERR     JMP  $AE88       FUNCTION CALL ERR
```

ON ERROR GOTO for OSI ROM BASIC

by Earl Morris and Kerry Lourash



When OSI ROM BASIC encounters an error, program execution is halted and the screen displays the dreaded

? S* ERROR IN LINE xx

where the * is a graphics character rather than the correct letter. The following programs add an "ON ERR GOTO" function to your machine so that errors are detected and a jump is made to program line 50000. The line number where the error occurred is stored in the variable XX and the type of error is stored in X. At line 50000 the programmer can print out the expanded error message, fix the error, or jump back to the program. As an added bonus, the graphics character in the error message is converted to the correct alphabetic letter.

As an example, consider the program

```
10 INPUT "NUMBER"; A
20 PRINT:PRINT 1/A
30 GOTO 10
```

If a zero is input, the program halts with a divide-by-zero error in line 20. With the error-trap program in place, the following can be added:

```
50000 PRINT: IF XX < > 20 THEN END
50010 PRINT:PRINT "CAN'T DIVIDE BY ZERO — TRY AGAIN"
50020 GOTO 10
```

Listing 1: 1P Version

```
0000               ;*********************************
0000               ;*  ON ERROR ROUTINE, 1P VERSION  *
0000               ;*********************************
0000               ;
0000               ;GOES TO LINE 50000 ON ERROR WITH
0000               ;LINE NUMBER IN XX, ERROR TYPE IN X.
0000               ;
0000               ;SET UP $021A=$22,  $021B=$02
0000               ;
0222                   *=$0222
0222               ;
0222 C90D                 CMP #$0D      IS OUTPUT A CR?
0224 D015                 BNE  BYE      NO, EXIT TO NORMAL
0226 8A                   TXA           SAVE X REGISTER
0227 48                   PHA
0228 BA                   TSX           GET STACK POINTER
0229 BD0601               LDA  $106,X   IS CALLING ADDRESS
022C C952                 CMP #$52      $A252 ?
022E D007                 BNE  A1
0230 BD0701               LDA  $107,X
0233 C9A2                 CMP #$A2
0235 F007                 BEQ  ERRTRP   YES, TO ERR TRAP
0237 68            A1     PLA           RESTORE X-REG.
0238 AA                   TAX
0239 A90D                 LDA #$0D      RESTORE A-REG.
023B 4C69FF        BYE    JMP  $FF69    GOTO REGULAR OUTPUT
023E               ;
023E A588          ERRTRP LDA  $88      IF IN IMM. MODE
0240 C9FF                 CMP #$FF      PRINT ERROR MESSAGE
0242 F04C                 BEQ  ERROR
0244               ;
0244               ;******STORE CURRENT LINE # IN XX ******
0244 A487                 LDY  $87      STORE CURRENT LINE #
0246 85AD                 STA  $AD      IN F.P.A
0248 84AE                 STY  $AE
024A A290                 LDX #$90
024C 38                   SEC
024D 20E8B7               JSR  $B7E8    CONVERT LINE# TO F.P.
0250 A900                 LDA #0
0252 855E                 STA  $5E      SET DEFAULT DIM FLAG
0254 855F                 STA  $5F      SET VAR. TYPE FLAG
0256 A958                 LDA #'X       SPECIFY XX VARIABLE
0258 8593                 STA  $93      NAME
025A 8594                 STA  $94
025C 2049AD               JSR  $AD49    FIND OR CREATE XX
025F 8597                 STA  $97
0261 8498                 STY  $98
0263 2074B7               JSR  $B774    STORE F.P.A IN XX
0266               ;
0266               ;******STORE ERROR #/2 IN X ******
0266 68                   PLA           PULL ERROR #
0267 48                   PHA           SAVE IT AGAIN
0268 4A                   LSR  A        HALVE IT
0269 A8                   TAY
026A A900                 LDA #0
026C 20C1AF               JSR  $AFC1    STORE ERR # IN F.P.A
026F A900                 LDA #0
0271 8594                 STA  $94
```

(continued)

If an error occurs in line 20, the error trap program prints a message and continues program execution. Other errors will still end the program. The error trap resets the stack, effectively clearing all loops and subroutines. The jump back to the main program cannot enter within a FOR-NEXT loop or go directly to a subroutine.

Two versions of the ON ERR routine are listed: 1P and 540. Use the version appropriate for your machine. The method used to detect errors is different for each type of computer. The 1P version uses the output vector on page two. On every carriage return, the ON ERR program searches the stack to determine which routine is writing to the screen. If a $A252 is found on the stack, then the error routine is outputting and the ON ERRor program is triggered.

Machines other than the 1P do not have the output vector in RAM, and must use a different hook into BASIC. The ON ERR program hooks into the OK message printer at $0003. The routine looks for the "?", which appears above the OK whenever an error occurs. A disadvantage of this hook is that the normal error message has already been printed and the type of error is no longer in memory. Thus, the 540 version stores a value in XX (line number) but not in X (error type).

In both programs, after an error is detected, location $88 is inspected. If it contains a $FF, the computer is in the immediate mode and the ON ERRor routine is bypassed. Then the normal error message (corrected) is printed. If you wish to use ON ERRor in the immediate mode, change the following location:

```
 1P — Change $0243 from $4C to $00
540 — Change $0259 from $EE to $00
```

The variable XX contains 65xxx as a line number if the error occurs in the immediate mode.

If the computer is not in immediate mode, or if the above patch is made, the current line number is converted to floating point and stored in the variable XX. The error index contained in the X register is halved, converted to floating point, and stored in the variable X.

Next a search is made for line 50000. If it is found, the parser pointer is set to the start of line 50000 and the program jumps to the start of the BASIC execution loop. If no line 50000 is found, the normal error message is output and execution is halted.

Notes on 1P Version

Whenever the BREAK key is pressed, the 1P's vectors are reset to the original. The output vector again must be pointed to ON ERRor after every break. This can also be done with

```
POKE 538,34 : POKE 539,2
```

Listing 1 *(continued)*

```
590 0273 2049AD          JSR  $AD49     FIND OR CREATE X
600 0276 8597           STA  $97
610 0278 8498            STY  $98
620 027A 2074B7          JSR  $B774     STORE F.P.A IN X
625 027D          ;
630 027D          ;******FIND LINE 50000******
640 027D A950            LDA #$50       HEX 50000 IN $11,12
650 027F 8511           STA  $11
660 0281 A9C3            LDA #$C3
670 0283 8512            STA  $12
680 0285 2032A4          JSR  $A432     LOOK FOR LINE
690 0288 9006            BCC  ERROR     BRANCH IF NO LINE
700 028A 20D9A6          JSR  $A6D9     SET PARSER AT 50000
710 028D 4CC2A5          JMP  $A5C2     GOTO BASIC EXEC. LOOP
720 0290          ;
730 0290          ;******PRINT ERROR MESSAGE******
740 0290 68       ERROR  PLA            PULL ERROR INDEX
750 0291 AA              TAX
760 0292 20E3A8          JSR  $A8E3     PRINT '?'
770 0295 BD64A1          LDA  $A164,X   GET FIRST CHARACTER
780 0298 20E5A8          JSR  $A8E5     PRINT IT
790 029B BD65A1          LDA  $A165,X   GET SECOND CHARACTER
800 029E 297F            AND #$7F       ZERO HI BIT OF CHAR
810 02A0 4C5FA2          JMP  $A25F     TO REG. ERR ROUTINE
```

For the 1P version, the error type is contained in the variable X. Table 1 lists the error types. A program can be written to print out the full error descriptions if you have trouble remembering what "T*" means.

Notes on 540 Version

On error can also be set up using

```
POKE 4,64 : POKE 5,2
```

The first command in line 50000 should be PRINT. This scrolls the error message up one line to prevent retriggering ON ERRor. The 540 version does not put the error type into X, but the error type is displayed on the screen at $D741 and $D742. The ON ERRor program could be extended to read these locations and do a table look-up to get the error index.

Table 1: Error Types

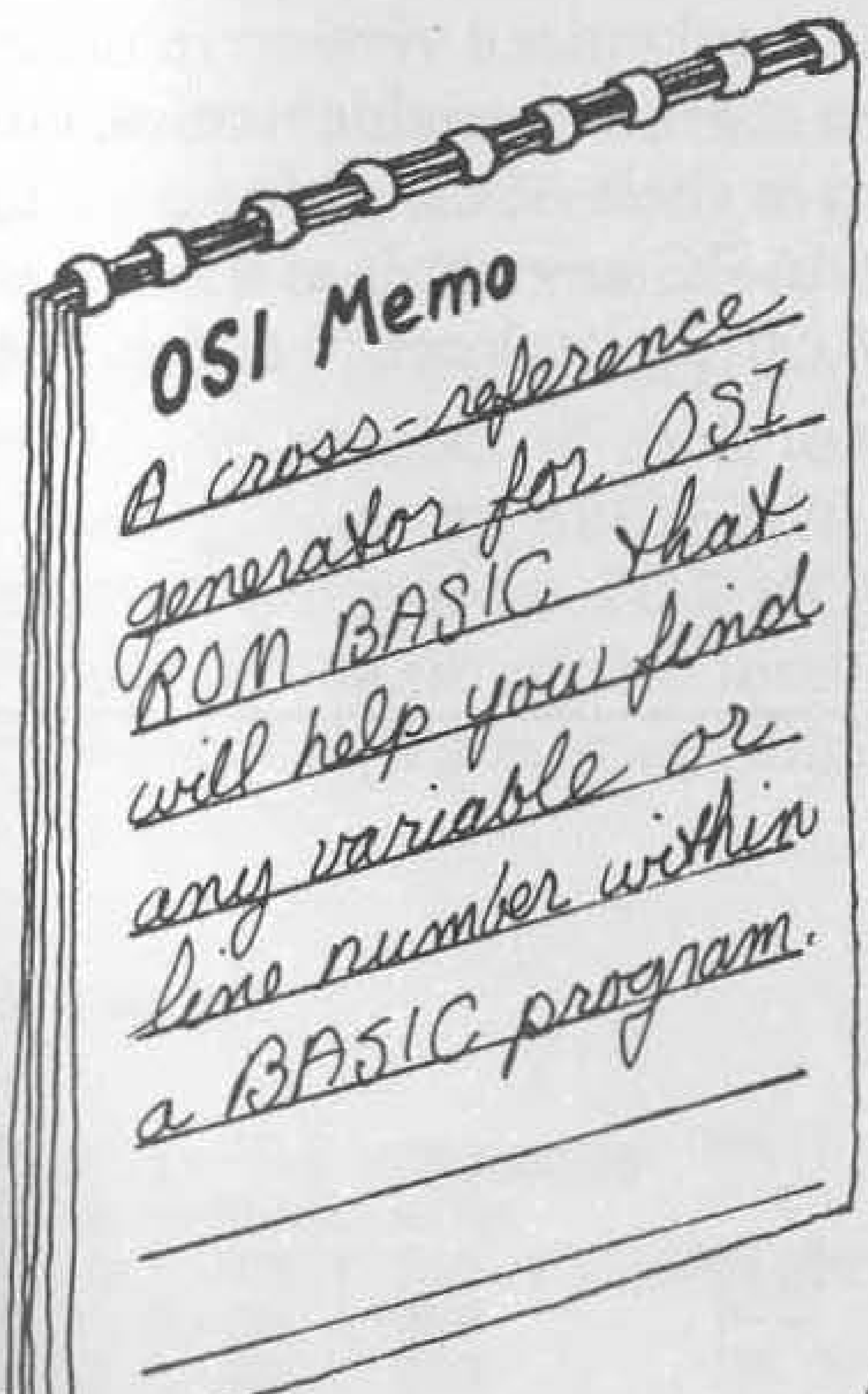
Index	Error Message
0	Next Without For
1	Syntax Error
2	Return Without Gosub
3	Out Of Data
4	Function Call — argument out of range
5	Overflow
6	Out Of Memory
7	Undefined Statement GOTO non-existent line
8	Bad Subscript Subscript greater than dimension
9	Double Dimension
10	Division By Zero
11	Illegal Direct Can't use in immediate mode
12	Type Mismatch
13	Long String
14	String Temporaries
15	Continue Error
16	Undefined Function

Listing 2: 540 Video Version

```
10 0000           ;***********************************
20 0000           ;*  ON ERROR ROUTINE FOR 540 VIDEO  *
30 0000           ;***********************************
40 0000           ;
50 0000           ;GOES TO LINE 50000 ON ERROR
60 0000           ;WITH LINE NUMBER IN XX.
70 0000           ;
80 0000           ;SET UP  $0004=$40  $0005=$02
90 0000           ;
100 0240                *=$0240
110 0240          ;
120 0240 48             PHA
130 0241 AD40D7         LDA $D740   READ CHAR FROM SCREEN
140 0244 C93F           CMP #'?     IS IT A QUESTION MARK?
150 0246 F004           BEQ J1      IF YES, THEN ERROR OCCURED
160 0248 68       J2    PLA         NORMAL MESSAGE OUTPUT
170 0249 4CC3A8         JMP $A8C3   MESSAGE PRINTER
180 024C          ;
190 024C AD42D7   J1    LDA $D742   GET GRAPHICS CHARACTER
200 024F 297F           AND #$7F    CLEAR HI BIT
210 0251 8D42D7         STA $D742   RETURN CHAR TO SCREEN
220 0254 A588           LDA $88
230 0256 C9FF           CMP #$FF    IN IMMEDIATE MODE?
240 0258 F0EE           BEQ J2      YES, GO TO BASIC
250 025A 68             PLA
260 025B A487           LDY $87     PUT CURRENT LINE #
270 025D 84AE           STY $AE     IN $AD, $AE
280 025F A588           LDA $88
290 0261 85AD           STA $AD
300 0263 A290           LDX #$90
310 0265 38             SEC
320 0266 20E8B7         JSR $B7E8   CONVERT HEX TO FLOATING
330 0269 A900           LDA #0
340 026B 855E           STA $5E     SET DIM DEFAULT FLAG
350 026D 855F           STA $5F     SET VARIABLE TYPE
360 026F A958           LDA #'X     SPECIFY XX VARIABLE
370 0271 8593           STA $93
380 0273 8594           STA $94
390 0275 2049AD         JSR $AD49   FIND OR CREATE XX VAR.
400 0278 8597           STA $97
410 027A 8498           STY $98
420 027C 2074B7         JSR $B774   PUT VALUE INTO XX
430 027F A950           LDA #$50    PUT HEX 50000 INTO $11,$12
440 0281 8511           STA $11
450 0283 A9C3           LDA #$C3
460 0285 8512           STA $12
470 0287 2032A4         JSR $A432   LOOK FOR LINE 50000
480 028A 9006           BCC J3      BRANCH IF LINE NOT FOUND
490 028C 20D9A6         JSR $A6D9   POINT TO LINE 50000
500 028F 4CC2A5         JMP $A5C2   GOTO BASIC EXEC LOOP
510 0292          ;
520 0292 A992     J3    LDA #$92    NO LINE 50000- PRINT 'OK'
530 0294 A0A1           LDY #$A1
540 0296 4CC3A8         JMP $A8C3   MESSAGE PRINTER
```

Cross-Reference Generator for OSI BASIC-in-ROM

by John Krout



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

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

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

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

Listing 1

```
60010 T=PEEK(123)+256*PEEK(124)
60100 FOR I=768 TO T
60110 X=PEEK(I)
60160 PRINT I;CHR$(X);X
60170 NEXT I: END
```

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

Table 1

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

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

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

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

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

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

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

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

Listing 2

```
60120 IF X=0 GOTO60500
60500 REM NEW LINE
60510 LINE=PEEK( I+3 )+256*PEEK( I+4 )
60520 PRINT LINE
60530 I=I+5
60540 GOTO 60110
```

If BASIC can translate new text lines to tokens and, during a LIST, *vice versa*, then there should be a dictionary of BASIC vocabulary and corresponding tokens somewhere in ROM. In fact, the dictionary resides in addresses 41092 through 41314 (see MICRO 24:25, 23:65). Listing 3 takes a look at the dictionary, and the results of listing 3 appear in table 2. The items are placed in the dictionary in numerical order of their corresponding tokens. The last character of each item has its most significant digit set to 1 to tell BASIC that the end of the item has been reached. In listing 3, X represents a byte of data in the dictionary and is used in line

61040 to build a string, B$, of consecutive bytes. Line 61050 branches to avoid incrementing the token number, variable TK, and printing and clearing B$, if the item is not yet complete; i.e., if the most significant bit of X is cleared. While assembling B$, use Boolean logic in line 61040 to clear the most significant bit of every character, not just the last one. This may be overkill, but it is also compact code and serves the need to conserve RAM. Now combine listings 1 through 3. This enables you to search for any string, or token corresponding to a dictionary item, that you need to find.

Listing 3

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

Table 2

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

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

Listing 4

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

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

Listing 5

```
60030 INPUT"KEYWORD OR STRING";A$:PRINT
60040 IF ASC(A$)=75 GOTO 61000
60170 NEXT I:PRINT:GOTO60030
61005 INPUT"WHICH KEYWORD";A$:PRINT
61015 L=LEN(A$)
61070 IFA$=LEFT$(B$,L)THENA$=CHR$(TK):GOTO60070
61200 PRINTA$;"NOT FOUND":PRINT:GOTO60030
```

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

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

The problem of distinguishing subscripted, string, and numeric variables is easier to solve. If references to a numeric variable are sought, the program should reject any it finds that are followed by either a (or a $. If references to a string variable are sought, the program should ignore any followed by a (character. These suffix rejection rules for numeric and string variables suggest that you can eliminate erroneous references embedded in larger strings (illustrated above by the searches for 7 and T) by implementing a set of suffix and prefix rejection rules. The prefix rule for all strings is rejection of references preceded by a numeric or upper-case alphabetic character. The suffix rule for constants, line references, and numeric variables is as stated above for numeric variables, with the additional rejection of numeric and upper-case alphabetic suffixes.

Listing 6 incorporates these rules into the cross-reference generator, utilizing three defined Boolean functions in a single IF/GOTO statement. The functions are defined in lines 60005 through 60007. The argument in each is the ASCII value of a character. FNA returns a true value if the character is numeric or upper-case alpha. FNB returns true if the character is neither (nor $. FNC, utilizing FNA and FNB in its definition, returns true if the character is either numeric, upper-case alpha, (, or $. Line 60070 is modified to set new variable A equal to the ASCII value of the first byte of A$. Lines 60080 and 60135 skip over the rules implementation if A indicates that A$ represents a token. Line 60090 sets new variable B equal to the ASCII value of the last byte of B$, to decide later if the string to be found is a subscripted or string variable.

Listing 6

```
60005 DEF FNA(X)=(X>47ANDX<58) OR (X>64ANDX<91)
60006 DEF FNB(X)=X<>36 AND X<>40
60007 DEF FNC(X)=NOT FNB(X) OR FNA(X)
60070 L=LEN(A$):B$="":A=ASC(A$)
60080 IF A>127 GOTO 60100
60090 B=ASC(RIGHT$(A$,1))
60135 IF A>127 GOTO 60160
60140 IF A$<>RIGHT$(B$,L) GOTO 60170
60145 Y=PEEK(I+1):IFLEN(B$)>LTHENW=ASC(RIGHT$(B$,L+1))
60150 IFFNA(W)OR(B=36ANDY=40)OR(FNB(B)ANDFNC(Y))GOTO60170
60535 W=0
```

Since the program doesn't need the rules unless a potential reference is located, line 60140 jumps past the rules until that condition is met. In line 60145, Y is the ASCII value for the reference suffix and, if the reference is not the first item in the text line, then W is the ASCII value of the reference prefix. Line 60535 sets W to zero whenever a new line number is decoded. Line 60150 skips the line number printing statement if any of the prefix or suffix rejection rules are met when a potential reference is found. This is one easy way to read the line:

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

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

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

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

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

Listing 7

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

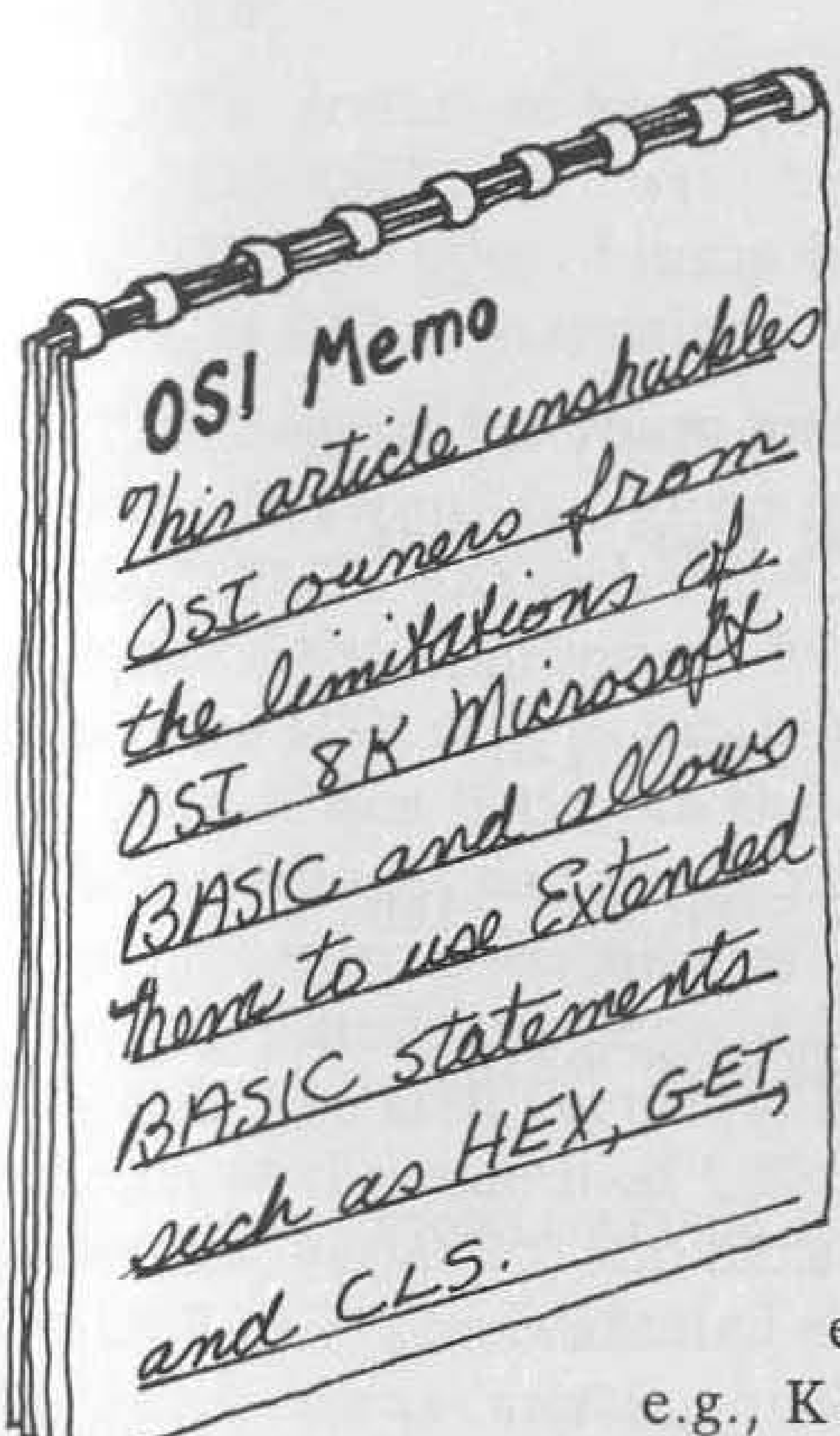
Have you been wondering about the need for the next-line pointers? They are essential to BASIC's execution of branching statements. An understanding of this process will help you improve execution speed of your own programs as well as the cross-reference generator. When a branch token such as a GOTO is executed, BASIC first translates the string of digits following the token into the low-high line-number format. The speed of this operation clearly depends on the length of the string, so it always helps to use small line numbers, even though this may be impractical in large programs. If line references were stored in low-high format when tokenized, it would save memory and speed things up. I suspect Microsoft shares my conclusion that it is difficult to distinguish constants and line references.

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

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

Extended OSI BASIC

by Collin Macauley and Jeff Macauley



Attempts to emulate an Extended BASIC have been undertaken but they have never been user transparent. Ed Carlson, in a MICRO article (25:15), altered the parser routine (CHRGET). Michael Mahoney continued this theme in a follow-up article (MICRO 46:51). Unfortunately the parser routine is the most used subroutine in BASIC and the loss in speed may be unacceptable to many programmers. Additionally, the use of "#C" in a program does not look like a CLS command. A different approach was taken by Yasuo Morishita in PEEK(65) Vol. 2, No. 11, where an extended USR(X) statement was used; e.g., K=USR(0)KY designated a GET command. Again this solution did not use accepted syntax.

Our program is a development of the Morishita program and uses an adaptation of that program when the Extended BASIC commands are called. Originally the program was developed for use with Synertek 8K BASIC where an enhanced USR(X) statement is available.

The program will recognize any user-designated statements; the only limitation is the ability of the programmer to define code to support these statements. A jump and keyword table are readily expanded when further additions are developed.

The program is divided into three sections, described as follows:

INPUT. The input vector ($218, $219) is pointed to this routine and converts the user's keywords into their correct USR calls When a carriage return is detected the routine checks the input buffer for each keyword; e.g., CLS. If a keyword is located it is converted into a user call of one of the following types:

a. O0=USR(0)$1CXY - standalone statement; e.g., CLS
b. USR(0)$1CXY - equate statement; e.g., X=HEX($AAAA)
where $1CXY is the appropriate address in the keyword jump table

Listing 1: Input Routine

```
0000                ;*************************
0000                ;*  EXTENDED OSI BASIC   *
0000                ;*                       *
0000                ;* COLIN & JEFF MACAULEY *
0000                ;*************************
0000                ;
0000                ;          INPUT
0000                ;
0000                        EA  =$E2
0000                        EX  =$E4
0000                        LB  =$13
0000                        LF  =$203
0000                        PB  =$E3
0000                        TE  =$E0
0000                        TF  =$200
0000                        TOK=$1B00
0000                        TW  =$E6
0000                        UF  =$F7        INPUT FLAG
0000                        US  =$1B14
0000                        XB  =$0E        LINE CHAR. COUNTER
0000                ;
1E00                        *=$1E00
1E00                ;
1E00 A5F7           TZ      LDA UF          SET INPUT FLAG
1E02 0940                   ORA #$40
1E04 85F7                   STA UF
1E06 20BAFF                 JSR $FFBA       GET CHAR FROM KYBD
1E09 C90D                   CMP #$D         END OF INPUT?
1E0B F001                   BEQ T
1E0D 60                     RTS             RETURN TO BASIC
1E0E                ;
1E0E A900           T       LDA #0
1E10 860E                   STX XB          SAVE BUFFER LENGTH
1E12 9513                   STA LB,X
1E14 AA                     TAX
1E15 86E0                   STX TE
1E17 A000                   LDY #0
1E19 B513           TO      LDA LB,X        CHECK BUFFER
1E1B E40E                   CPX XB          END OF BUFFER?
1E1D F009                   BEQ T8
1E1F D9001B                 CMP TOK,Y       NO, CHECK FOR KEYWORD
1E22 F020                   BEQ T1
1E24 E8             T4      INX             NO, LOOP BACK
1E25 4C191E                 JMP TO
1E28 A900           T8      LDA #0          RESET COUNTER
1E2A 85E0                   STA TE
1E2C AA                     TAX
1E2D B9001B         T5      LDA TOK,Y       FIND END OF KEYWORD
1E30 3004                   BMI T2
1E32 C8                     INY
1E33 4C2D1E                 JMP T5
1E36                ;
1E36 C8             T2      INY             SKIP TO NEXT KEYWORD
1E37 C8                     INY
1E38 C8                     INY
1E39 C8                     INY
1E3A C9FF                   CMP #$FF        END OF KEYWORDS?
1E3C D0DB                   BNE TO          NO, LOOP BACK
```

(continued)

The input buffer will be expanded to accommodate the conversion. Thus the program line "10 CLS : X = 1" would be expanded to "10 O0 = USR(0)$1C03 : X = 1" in the input buffer. With this expansion, care must be taken not to overflow the input buffer as an error will be flagged if multistatement lines are too long. Also note that the variable O0 cannot be used and has been chosen specifically because it is unlikely that a programmer would use it in view of the letter O/zero confusion.

Each keyword has four parts in the keyword table. For example, consider CLS:

1. 'CL' letters of keyword less one
2. $D3 ASCII "S" with highest bit set
3. '03' low byte of jump table address ($1C03)
4. $00 $00 = standalone statement, else $03

The program has room for additional keywords, which can be inserted into the keyword and jump tables.

OUTPUT. The output vector ($21A, $21B) points to this routine and will print the appropriate Extended BASIC statements rather than the converted USR call. In this manner the USR call conversion is invisible to the user. When listing to the screen or tape, the routine searches for the USR statements and prints them only when no match is found; i.e., an actual program USR call was made, as opposed to a keyword USR call. If a match is found, the keyword, rather than the USR call is printed.

USR. The USR vector ($0B, $0C) points to this routine and allows execution of the redefined USR call. This new type of USR call allows expressions, variables, and hexadecimal values to be evaluated and used by the USR call. The evaluated expressions, etc., are stored as integers in the low/high format starting at $E0. Any Extended BASIC routine can then access these locations when required. Because of the revised form of USR call, any non-keyword USR calls *must* be of the following type:

USR(0)XX,A,...F
where XX is the call address
A-F are up to 7 data values

XX and A-F may be expressions, numbers, or hexadecimal numbers (if preceded by a "$" sign).

This change makes the USR call easier to decipher, as you are freed from continually changing locations $0B, $0C before calling a USR routine with the USR address always being identifiable. The changes to Yasuo Morishita's program were to allow parentheses to be used in defining a statement; e.g., AUTO (start, inc) for an auto line-number command. The parentheses cause the BASIC expression handler ($AAAD) to flag an error and must be skipped. The routine checks for the open parenthesis and, if found, replaces the close parenthesis with a colon. After all expressions are evaluated, the open parenthesis is easily skipped and the

Listing 1 *(continued)*

```
1E3E 18            CLC          YES, EXIT TO BASIC
1E3F A60E          LDX XB
1E41 A90D          LDA #$D
1E43 60            RTS
1E44          ;
1E44 E6E0     T1   INC TE
1E46 C8            INY
1E47 E8            INX
1E48 B9001B        LDA TOK,Y
1E4B 297F          AND #$7F
1E4D D513          CMP LB,X     MATCH FOR NEXT CHAR OF KYWD.
1E4F D008          BNE T3       NO, CHECK BUFFER AGAIN
1E51 B9001B        LDA TOK,Y
1E54 300C          BMI T6       KEYWORD FOUND
1E56 4C441E        JMP T1       KEEP CHECKING
1E59          ;
1E59 88       T3   DEY          RESET COUNTER
1E5A CA            DEX
1E5B C6E0          DEC TE
1E5D D0FA          BNE T3
1E5F 4C241E        JMP T4
1E62          ;
1E62 C8       T6   INY
1E63 E8            INX
1E64 84E2          STY EA
1E66 86E3          STX PB
1E68 B9021B        LDA TOK+2,Y STANDALONE KEYWORD?
1E6B F005          BEQ K1       YES, USE 00=USR(0)$1CXY
1E6D A90B          LDA #$B      NO, USE USR(0)$1CXY
1E6F 4C741E        JMP T17
1E72 A90E     K1   LDA #$E
1E74 38       T17  SEC
1E75 E5E0          SBC TE
1E77 85E4          STA EX
1E79 C6E4          DEC EX
1E7B 18            CLC
1E7C 650E          ADC XB
1E7E C947          CMP #$47     BUFFER OVERFLOW?
1E80 9005          BCC K2
1E82 A20A          LDX #$A      YES, FLAG ERROR
1E84 4C4EA2        JMP $A24E
1E87 A50E     K2   LDA XB       SET UP TO EXPAND BUFFER
1E89 38            SEC
1E8A E5E3          SBC PB
1E8C AA            TAX
1E8D E8            INX
1E8E A900          LDA #0
1E90 85E7          STA TW+1
1E92 A50E          LDA XB
1E94 18            CLC
1E95 6913          ADC #LB
1E97 85E6          STA TW
1E99 A000     EP   LDY #0       EXPAND BUFFER
1E9B B1E6          LDA (TW),Y
1E9D A4E4          LDY EX
1E9F 91E6          STA (TW),Y
1EA1 C6E6          DEC TW
1EA3 CA            DEX
1EA4 D0F3          BNE EP
1EA6 A50E          LDA XB
```

(continued)

colon subsequently replaced by a close parenthesis before execution to the appropriate machine-language subroutine.

For an 8K RAM system, the Cold Start MEMORY SIZE? Prompt should be answered with "6900" to protect the program from being overwritten by BASIC. The program uses zero-page locations $DF-$EF and $F2-$F8 and care must be taken if your monitor or machine-code programs use these locations. To initialize the program the following POKEs are required:

```
POKE 247,0 : POKE 11,64 : POKE 12,28 - UF flag; USR vector
POKE 538,0 : POKE 539,29 : POKE 536,0 : POKE 537,30 - Output/Input vectors
```

To demonstrate the program, three keywords have been included in the listings:

1. CLS - clear screen
2. GET - wait for a keyboard response and save ASCII value of key hit
3. HEX($XXXX) - converts hexadecimal value XXXX into decimal

In use, the following short program

```
10 CLS
20 PRINT HEX($2000)
30 PRINT GET
```

will in actual fact be stored as

```
10 O0 = USR(0)$1C03)
20 PRINT USR(0)$1C00($2000)
30 PRINT USR(0)$1C06
```

in memory, but this will not be visible to the programmer.

The program may be relocated and transferred to EPROM to save your valuable RAM. With this program you will need to thumb through your back issues of MICRO to locate those routines for PRINTAT, AUTO, PLAY, etc., which may be readily incorporated.

Listing 1 *(continued)*

```
1EA8 18                 CLC
1EA9 65E4               ADC  EX
1EAB 850E               STA  XB
1EAD A4E2               LDY  EA
1EAF B9021B             LDA  TOK+2,Y  STANDALONE KEYWORD?
1EB2 F004               BEQ  T18
1EB4 A003               LDY  #3
1EB6 D002               BNE  T19
1EB8 A000        T18    LDY  #0
1EBA A5E3        T19    LDA  PB
1EBC 38                 SEC
1EBD E5E0               SBC  TE
1EBF AA                 TAX
1EC0 CA                 DEX
1EC1 B9141B      EP3    LDA  US,Y     SHIFT USR INTO BUFFER
1EC4 9513               STA  LB,X
1EC6 E8                 INX
1EC7 C8                 INY
1EC8 C00C               CPY  #$C
1ECA D0F5               BNE  EP3
1ECC A4E2               LDY  EA
1ECE B9001B             LDA  TOK,Y    PUT USR ADDRESS INTO BUFFER
1ED1 9513               STA  LB,X
1ED3 B9011B             LDA  TOK+1,Y
1ED6 E8                 INX
1ED7 9513               STA  LB,X
1ED9 A900               LDA  #0
1EDB 85E0               STA  TE
1EDD A8                 TAY
1EDE 4C241E             JMP  T4       CHECK FOR MORE KEYWORDS
```

Listing 2: Output Routine

```
0000                ;*************************
0000                ;*  EXTENDED OSI BASIC   *
0000                ;*                       *
0000                ;* COLIN & JEFF MACAULEY *
0000                ;*************************
0000                          ;
0000                ;         OUTPUT
0000                ;
0000                          PA1   =$F3
0000                          PA2   =$F4
0000                          PRINT=$FF69
0000                          PX    =$F8
0000                          TOCS =$F5
0000                          SPX   =$F2
0000                          TOK   =$1B00
0000                          UF    =$F7       INPUT FLAG
0000                          US    =$1B14
0000                          V1    =$F6
0000                          ;
1D00                          *=$1D00
1D00                ;
1D00 24F7     BASP      BIT UF         CHECK INPUT FLAG
1D02 7013               BVS BA3        YES, PRINT CHARACTER
1D04 1003               BPL BA1        USR ALREADY DETECTED?
1D06 4C391D             JMP FS
1D09 C94F     BA1       CMP #'O        NO, CHECK FOR USR
1D0B D003               BNE BA2
1D0D 4C221D             JMP OST
1D10 C955     BA2       CMP #'U        NOT FOUND, PRINT CHAR
1D12 D003               BNE BA3
1D14 4C331D             JMP UST
1D17                ;
1D17 48       BA3       PHA
1D18 A5F7               LDA UF
1D1A 29BF               AND #$BF       CLEAR INPUT FLAG
1D1C 85F7               STA UF
1D1E 68                 PLA
1D1F 4C69FF             JMP PRINT      PRINT CHAR
1D22                ;
1D22 48       OST       PHA            MAYBE OO=USR( O )$1CXY
1D23 A901               LDA #1
1D25 85F8     OST1      STA PX
1D27 85F2               STA SPX
1D29 A980               LDA #$80       SET UWR FLAG
1D2B 85F7               STA UF
1D2D 68                 PLA
1D2E A964               LDA #100
1D30 850F               STA $0F
1D32 60                 RTS            RETURN TO BASIC
1D33                ;
1D33 48       UST       PHA            MAYBE USR( O )$1CXY
1D34 A904               LDA #4
1D36 4C251D             JMP OST1
1D39                ;
1D39 85F6     FS        STA V1         SAVE REGISTERS
1D3B 48                 PHA
1D3C 8A                 TXA
1D3D 48                 PHA
```

(continued)

Listing 2 *(continued)*

```
1D3E 98                 TYA
1D3F 48                 PHA
1D40 A5F7               LDA UF        CLEAR INPUT FLAG
1D42 29BF               AND #$BF
1D44 85F7               STA UF
1D46 A5F6               LDA V1
1D48 A6F8               LDX PX
1D4A E00E               CPX #14       00=USR(0) OR USR(0) FOUND
1D4C F03C               BEQ P50       CHECK FOR REST OF ADDRESS
1D4E D01A               BNE P2
1D50              ;
1D50 A6F2         P4    LDX SPX       NO MATCH
1D52 CA                 DEX
1D53 BD141B       P44   LDA US,X      PRINT CHARS HELD BACK
1D56 2069FF             JSR PRINT
1D59 E8                 INX
1D5A E4F8               CPX PX
1D5C D0F5               BNE P44
1D5E A900               LDA #0        CLEAR USR FLAG
1D60 85F7               STA UF
1D62 68                 PLA
1D63 A8                 TAY
1D64 68                 PLA
1D65 AA                 TAX
1D66 68                 PLA
1D67 4C69FF             JMP PRINT     PRINT & RETURN TO BASIC
1D6A              ;
1D6A E00C         P2    CPX #12       SAVE X OF $1CXY
1D6C F012               BEQ P5
1D6E E00D               CPX #13       SAVE Y OF $1CXY
1D70 F013               BEQ P6
1D72 DD141B             CMP US,X      CHECK CHARS
1D75 D0D9               BNE P4        NO MATCH, PRINT CHARS
1D77 E8           P7    INX
1D78 86F8               STX PX        YES, HOLD PRINT
1D7A 68           RETR  PLA           RETURN TO BASIC
1D7B A8                 TAY
1D7C 68                 PLA
1D7D AA                 TAX
1D7E 68                 PLA
1D7F 60                 RTS
1D80              ;
1D80 85F3         P5    STA PA1
1D82 4C771D             JMP P7
1D85              ;
1D85 85F4         P6    STA PA2
1D87 4C771D             JMP P7
1D8A              ;
1D8A A200         P50   LDX #0
1D8C 86F5         P36   STX TOCS
1D8E E8           P33   INX
1D8F BD001B             LDA TOK,X     CHECK FOR KEYWORD
1D92 10FA               BPL P33
1D94 E8                 INX
1D95 BD001B             LDA TOK,X     FOUND
1D98 C5F3               CMP PA1       CHECK ADDRESS-1ST PART
1D9A D027               BNE P31       NO, NEXT KEYWORD
1D9C E8                 INX
1D9D BD001B             LDA TOK,X
1DA0 C5F4               CMP PA2       CHECK ADDRESS-2ND PART
```

(continued)

Listing 2 *(continued)*

```
1DA2 D020                BNE  P32
1DA4 A6F5                LDX  TOCS
1DA6 BD001B     P35      LDA  TOK,X
1DA9 3007                BMI  P34
1DAB 2069FF              JSR  PRINT
1DAE E8                  INX
1DAF 4CA61D              JMP  P35      CONTINUE PRINTING
1DB2 297F       P34      AND  #$7F
1DB4 2069FF              JSR  PRINT
1DB7 A900                LDA  #0       RETURN TO BASIC
1DB9 85F7                STA  UF
1DBB 68                  PLA
1DBC A8                  TAY
1DBD 68                  PLA
1DBE AA                  TAX
1DBF 68                  PLA
1DC0 4C69FF              JMP  PRINT
1DC3            ;
1DC3 E8         P31      INX
1DC4 E8         P32      INX
1DC5 E8                  INX
1DC6 BD001B              LDA  TOK,X    CHECK NEXT KEYWORD
1DC9 3003                BMI  P38
1DCB 4C8C1D              JMP  P36
1DCE 4C0CAC     P38      JMP  $AC0C
```

Listing 3: USR Routine

```
0000            ;************************
0000            ;*  EXTENDED OSI BASIC   *
0000            ;*                       *
0000            ;* COLIN & JEFF MACAULEY *
0000            ;************************
0000            ;
0000            ;        USR
0000            ;
0000            ; ADAPTED FROM
0000            ; "EXTENDED USR(X) REVISITED"
0000            ; BY YASUO MORISHITA
0000            ; IN PEEK(65) VOL. 2, #11
0000            ;
1B00             *=$1B00
1B00            ;
1B00                    ;KEYWORD TOKENS
1B00 48          TOK    .BYT 'HE',$D8,'00',3
1B01 45
1B02 D8
1B03 30
1B04 30
1B05 03
1B06 43                 .BYT 'CL',$D3,'03',0
1B07 4C
1B08 D3
1B09 30
1B0A 33
1B0B 00
1B0C 47                 .BYT 'GE',$D4,'06',3
1B0D 45
1B0E D4
1B0F 30
1B10 36
1B11 03
1B12                    ;KEYWORD END MARKER
1B12 FF                 .BYT $FF,$FF
1B13 FF
1B14                    ;USR CALL
1B14 4F          US     .BYTE 'O0=USR(0)$1C'
1B15 30
1B16 3D
1B17 55
1B18 53
1B19 52
1B1A 28
1B1B 30
1B1C 29
1B1D 24
1B1E 31
1B1F 43
1C00             *=$1C00
1C00            ;
1C00                    ;KEYWORD JUMP TABLE
1C00 4CDB1C             JMP HEX
1C03 4CE21C             JMP CLS
1C06 4CF61C             JMP GET
1C40             BAS    *=*+55
1C40            ;
```

(continued)

Listing 3 *(continued)*

```
1C40                    CP     =$E0
1C40                    CHRGET=$BC
1C40                    CHRGOT=$C2
1C40             ;
1C40  A200       EXTUSR LDX #0
1C42  865A              STX $5A       RESET DATA COUNTER
1C44  86DF              STX $DF       CLEAR BRACKET FLAG
1C46  A000              LDY #0
1C48  B1C3       CH4    LDA ($C3),Y
1C4A  C928              CMP #'(       BRACKETS USED?
1C4C  D006              BNE CH1
1C4E  A901              LDA #1        YES, SET FLAG
1C50  85DF              STA $DF
1C52  D008              BNE CH2
1C54  C900       CH1    CMP #0        END OF LINE?
1C56  F007              BEQ CH3       YES, CHECK USR ADDRESS
1C58  C93A              CMP #':       END OF STATEMENT?
1C5A  F003              BEQ CH3
1C5C  C8         CH2    INY
1C5D  D0E9              BNE CH4       LOOP BACK
1C5F  A5DF       CH3    LDA $DF       CHECK BRACKET FLAG
1C61  F00D              BEQ CH5       CLEAR, GET ADDRESS
1C63  88         CH6    DEY
1C64  F02F              BEQ SER
1C66  B1C3              LDA ($C3),Y
1C68  C929              CMP #')       SET, FIND CLOSE BRACKET
1C6A  D0F7              BNE CH6
1C6C  A93A              LDA #':       REPLACE ')' WITH ':'
1C6E  91C3              STA ($C3),Y
1C70  20C200     CH5    JSR CHRGOT    GET CURRENT CHAR
1C73  20A81C            JSR CD        GET USR ADDRESS
1C76  A5E0              LDA CP
1C78  05E1              ORA CP+1
1C7A  F019              BEQ SER
1C7C  20C200     JE4    JSR CHRGOT    END OF LINE?
1C7F  F017              BEQ JEOUT
1C81  C928              CMP #'(       NO, CHECK FOR BRACKET
1C83  D006              BNE CB
1C85  20BC00            JSR CHRGET    SKIP BRACKET
1C88  4C8E1C            JMP XX
1C8B  2001AC     CB     JSR $AC01     COMMA?
1C8E  20A81C     XX     JSR CD        GET DATA
1C91  E011              CPX #17       MORE THAN 7 DATA ITEMS?
1C93  30E7              BMI JE4       NO, GET MORE DATA
1C95  4C0CAC     SER    JMP $AC0C
1C98             ;
1C98  A5DF       JEOUT  LDA $DF       CHECK BRACKET FLAG
1C9A  F009              BEQ JE1
1C9C  A000              LDY #0
1C9E  A929              LDA #')       YES, REPLACE BRACKET
1CA0  91C3              STA ($C3),Y
1CA2  20BC00            JSR CHRGET    SKIP BRACKET
1CA5  6CE000     JE1    JMP (CP)      GOTO ML CODE ROUTINE
1CA8             ;
1CA8  C924       CD     CMP #'$       HEX EXPRESSION?
1CAA  D01E              BNE JD1
1CAC  A299       CD1    LDX #$99      YES, EVALUATE HEX
1CAE  A905              LDA #5
1CB0  8559              STA $59
1CB2  20BC00     JD3    JSR CHRGET
```

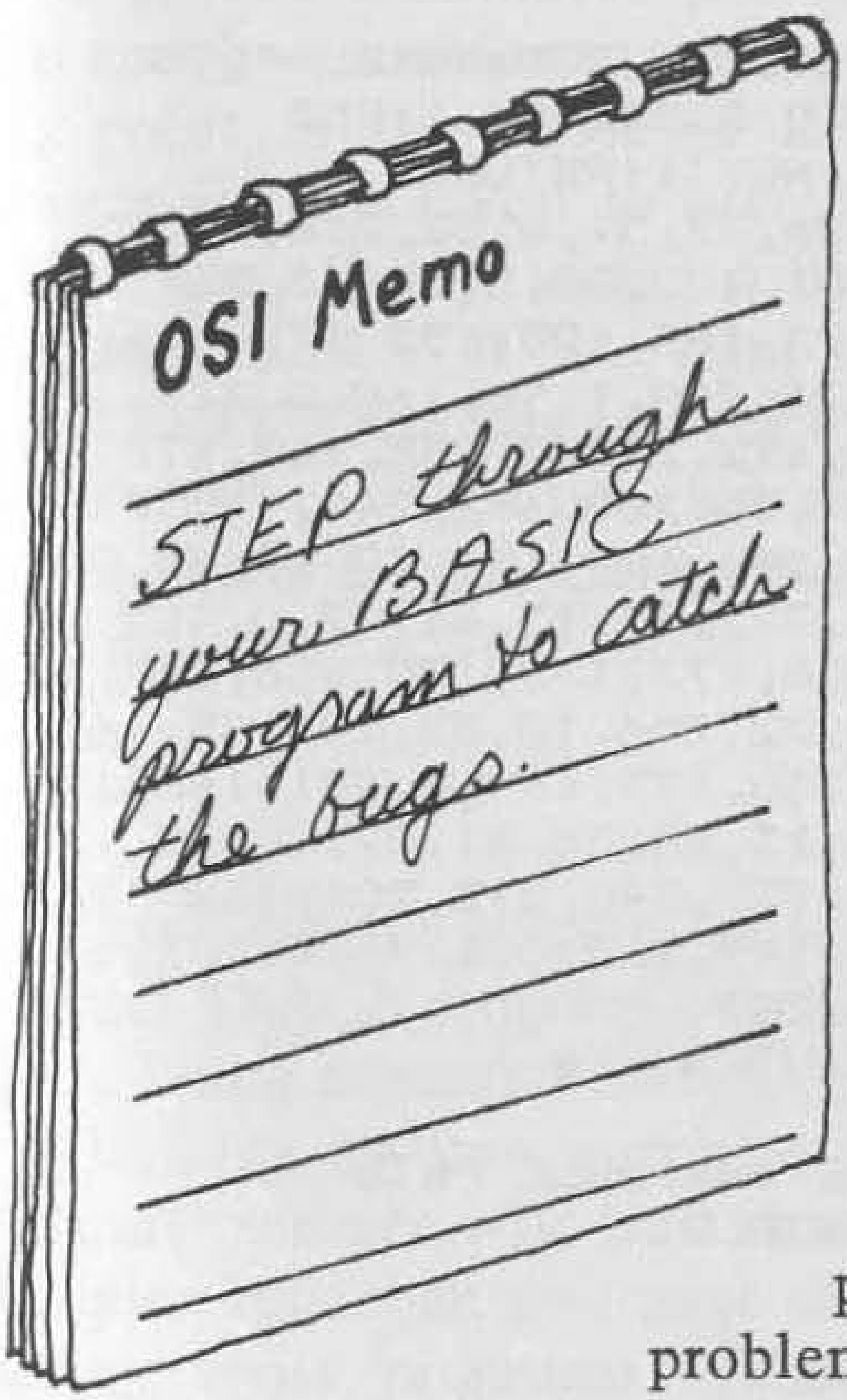
(continued)

Listing 3 *(continued)*

```
1CB5 C659            DEC $59
1CB7 F00A            BEQ JD2
1CB9 2093FE          JSR $FE93
1CBC 30D7            BMI SER
1CBE 20DAFE          JSR $FEDA
1CC1 F0EF            BEQ JD3
1CC3 A495     JD2    LDY $95
1CC5 A596            LDA $96
1CC7 18              CLC
1CC8 9006            BCC SX
1CCA 20ADAA   JD1    JSR $AAAD    EVALUATE EXPRESSION
1CCD 2008B4          JSR $B408
1CD0 A65A     SX     LDX $5A      STORE EVALUATION
1CD2 94E0            STY CP,X
1CD4 E8              INX
1CD5 95E0            STA CP,X
1CD7 E8              INX
1CD8 865A            STX $5A
1CDA 60              RTS
1CDB          ;
1CDB A5E3     HEX    LDA CP+3     HEX-DEC CONVERSION
1CDD A4E2            LDY CP+2
1CDF 4CC1AF          JMP $AFC1
1CE2          ;
1CE2 A000     CLS    LDY #0       CLEAR SCREEN
1CE4 A920            LDA #$20
1CE6 9900D0   CL     STA $D000,Y
1CE9 9900D1          STA $D100,Y
1CEC 9900D2          STA $D200,Y
1CEF 9900D3          STA $D300,Y
1CF2 C8              INY
1CF3 D0F1            BNE CL
1CF5 60              RTS
1CF6          ;
1CF6 2000FD   GET    JSR $FD00    GET A KEYSTROKE
1CF9 A8              TAY
1CFA A900            LDA #0
1CFC 4CC1AF          JMP $AFC1
```

BASIC STEP and TRACE

by Richard L. Trethewey



Debugging BASIC programs is always a chore, especially if you didn't write the program in the first place. If you don't have a printer, or if you do have one and don't want to pencil-check the program, the only alternative has been the standard "TRACE" program provided by M/A-OSI with all versions of OS-65D. That program prints out the line number of every new line as it is executed. For many purposes that is fine, but unfortunately the way this trace is implemented, the line numbers are not followed by a carriage return; you can easily get lost between these numbers and any output from the program being traced. This problem only gets worse if there are FOR-NEXT loops involved; you may find your output being scrolled off the screen because TRACE doesn't halt program execution — it just interrupts it. I think I have a simple solution.

I have written my own trace program that displays the line of program text before BASIC executes it and optionally displays all non-subscripted variables and their values. My program also waits for a keystroke before executing the line, or halts execution if the user presses the <RETURN> key. The tracing function allows you to halt execution even if the program being traced has disabled <CTRL> 'C' checking.

The BASIC program I have listed here POKEs the machine-code routine that does the tracing into memory and protects it from getting overwritten by BASIC. This code assumes you have 48K of memory on board. If you don't, you will have to re-assemble the machine code at a lower location using the source code I have included here. You will also have to change the routine starting at line 100, which does the POKEing into memory. It probably would be easier to change this routine to a call from disk to memory rather than compute the bytes that require changing

Listing 1

```
10 POKE133,175:REM- SET HIGH MEMORY TO $AFFF
20 GOSUB100:REM- POKE TRACE CODE INTO MEMORY AT $B000
30 INPUT"ENABLE OR DISABLE TRACE (E/D)";A$
40 L=2011:IFLEFT$(A$,1)="E"THENGOSUB260:GOTO70
50 IFLEFT$(A$,1)="D"THEN90
60 PRINT:PRINT"ENTER 'E' OR 'D' ONLY, PLEASE.":PRINT:GOTO30
70 POKEL,32:POKEL+1,0:POKEL+2,176:POKEL+3,234:POKEL+4,234
80 PRINT"TRACE ENABLED.":END
90 POKEL,24:POKEL+1,144:POKEL+2,2:POKEL+3,230:POKEL+4,200
95 M=PEEK(8960):POKE133,M:PRINT"TRACE DISABLED.":END
100 FORX=45056TO45273:READY:POKEX,Y:NEXTX:RETURN
110 DATA165,134,133,25,165,135,133,26,32,51,6,32,218,6,32
120 DATA115,10,32,33,176,32,64,35,201,13,208,5,169,3,76
130 DATA33,8,96,160,0,165,122,133,172,165,123,133,173,166,173
140 DATA228,125,208,7,165,172,197,124,208,1,96,160,0,177,172
150 DATA133,146,41,127,32,67,35,209,172,240,3,238,168,176,200
160 DATA177,172,133,147,41,127,32,67,35,209,172,240,19,174,168
170 DATA176,208,3,76,169,176,169,37,140,168,176,32,67,35,76
180 DATA113,176,32,138,15,32,157,26,32,115,45,61,32,0,165
190 DATA146,16,17,172,168,176,200,200,177,172,170,136,177,172,32
200 DATA220,28,76,146,176,32,236,28,32,204,10,32,106,45,165
210 DATA172,24,105,7,133,172,144,2,230,173,169,0,141,168,176
220 DATA76,43,176,0,140,168,176,32,115,45,36,61,32,0,172
230 DATA168,176,200,177,172,141,168,176,240,212,206,168,176,200,177
240 DATA172,133,148,200,177,172,133,149,160,0,177,148,32,67,35
250 DATA204,168,176,240,187,200,208,243
260 INPUT"DID YOU WANT VARIABLES PRINTED";Y$
270 IFLEFT$(Y$,1)="Y"THENRETURN
280 POKE45073,44:RETURN:REM- DISABLE VARIABLE PRINT
290 REM- POKE 45073 WITH 32 TO RE-ENABLE
```

in the DATA statements. I used POKEs to save a track on my disk and to make the program easier to transport to other disks.

To enable STEP/TRACE, run the program and respond with "E" to the prompt "ENABLE OR DISABLE TRACE (E/D) ?". You can then select whether or not to have the variables printed during the tracing. Now load and run the program to be debugged. You will see the first line to be executed displayed just as if you had entered LIST LN#. You will also see the variables that have been encountered on subsequent lines at this point, if you chose to do so from the TRACE program. Now the system waits for you to press a key before executing the line you see before you. If you want to continue, I suggest you simply press the <SPACE BAR>. If you want to stop before this line is executed, press the <RETURN> key and the system will display a <BREAK> message. If you need to check on a subscripted variable or do a PEEK you could do so now from the immediate mode. Also you can enter "CONT" now and continue program execution.

This program gives me a lot more control while debugging than the original TRACE program ever could. It's also nice to actually see the line that's being executed instead of having a program listing in front of me and looking up line numbers all the time. I'm sure that those of you without printers will find this handy too. The code for STEP/TRACE occupies less than one page of RAM, so it shouldn't prevent you from tracing most programs. When you disable STEP/TRACE your full workspace is returned to you.

Listing 2

```
0000           ;*****************
0000           ;****************************
0000           ;* BASIC SINGLE LINE STEPPER *
0000           ;*                          *
0000           ;*  BY RICHARD L. TRETHEWEY  *
0000           ;****************************
0000           ;
0000                    ;BASIC EXTERNALS
0000                    ;
0000            POKER=$19
0000            VARTAB=$7A          START OF VAR. TABLE
0000            ARRTAB=$7C          START OF ARRAYS
0000            ENDTAB=$7E          END OF ARRAYS
0000            EXLINE=$86          CURRENT LINE NUMBER
0000            VARNAM=$92          ASCII NAME OF VAR.
0000            VARPNT=$94          ADDRESS OF VARIABLE
0000            VARPTR=$AC
0000            FNDLIN=$0633        FIND A BASIC LINE
0000            DISLIN=$06D8        DISPLAY A BASIC LINE
0000            ZCFL  =$0821        CTRL C CHECK
0000            CRDO  =$0A73        DO LF, CR
0000            BASPRT=$0ACC        PRINT NUMBER
0000            GETVAR=$1A9D        PUT VAR. IN F.P.A
0000            ASCII =$1CEC        CONVERT F.P.A TO ASCII
0000            PNUMBR=$1CDC        PRINT INTEGER VARIABLE
0000           ;
0000                    ;OS-65D EXTERNALS
0000           ;
0000            CRLF  =$2D6A        PRINT LF,CR
0000            INCH  =$2340        GET KEYSTROKE
0000            CHROUT=$2343        PRINT CHARACTER
0000            STROUT=$2D73
0000           ;
B000                    *=$B000
B000 A586               LDA EXLINE     GET CURRENT LINE #
B002 8519               STA POKER      MOVE IT
B004 A587               LDA EXLINE+1
B006 851A               STA POKER+1
B008 203306             JSR FNDLIN     FIND LINE IN WORKSPACE
B00B 20DA06             JSR DISLIN+2   DISPLAY IT ON SCREEN
B00E 20730A             JSR CRDO       CLEAN UP WITH CR, LF
B011 2021B0             JSR VIEWIT     PRINT NON-SBSCRPTD. VAR'S
B014 204023             JSR INCH       GET A CHARACTER FROM KYBD.
B017 C90D               CMP #$D        IS IT A CR?
B019 D005               BNE CONT       NO, CONTINUE
B01B A903               LDA #3         YES, LOAD A CTRL C
B01D 4C2108             JMP ZCFL       AND EXECUTE IT
B020 60         CONT    RTS            BACK TO BASIC
B021           ;
B021 A000       VIEWIT  LDY #0
B023 A57A               LDA VARTAB     LOAD START OF VAR. TABLE
B025 85AC               STA VARPTR     PUT IT IN POINTER
B027 A57B               LDA VARTAB+1
B029 85AD               STA VARPTR+1
B02B A6AD       V0      LDX VARPTR+1   CHECK MSB OF POINTER
B02D E47D               CPX ARRTAB+1   SAME AS MSB OF END?
B02F D007               BNE V1         NO, PRINT VARIABLE
B031 A5AC               LDA VARPTR     YES, CHECK LSB, TOO
```

(continued)

Listing 2 *(continued)*

```
B033 C57C            CMP ARRTAB
B035 D001            BNE V1          NOT SAME, PRINT VAR.
B037 60              RTS             YES IT IS, QUIT
B038          ;
B038 A000     V1     LDY #0          INIZ INDEX
B03A B1AC            LDA (VARPTR),Y GET VAR. NAME 1ST BYTE
B03C 8592            STA VARNAM      SAVE IT
B03E 297F            AND #$7F        ZERO HI BIT
B040 204323          JSR CHROUT      AND PRINT IT
B043 D1AC            CMP (VARPTR),Y IS THIS AN INTEGER?
B045 F003            BEQ V2          NO, SKIP A BIT
B047 EEA8B0          INC STRFLG      YES, SHOW IT
B04A C8       V2     INY             BUMP INDEX
B04B B1AC            LDA (VARPTR),Y GET 2ND BYTE OF NAME
B04D 8593            STA VARNAM+1    SAVE IT
B04F 297F            AND #$7F        MASK AS BEFORE
B051 204323          JSR CHROUT      PRINT IT
B054 D1AC            CMP (VARPTR),Y IS THIS A SPECIAL VAR?
B056 F013            BEQ V3          NO, ITS AN F.P. TYPE
B058 AEA8B0          LDX STRFLG      CHECK IF AN INTEGER
B05B D003            BNE V5          FLAG SET! INTEGER=>V5
B05D 4CA9B0          JMP STRING      FLAG CLEAR STRING=>
B060          ;
B060 A925     V5     LDA #'%         LOAD '%'
B062 8CA8B0          STY STRFLG      SAVE INDEX
B065 204323          JSR CHROUT      PRINT THE '%'
B068 4C71B0          JMP V6          SKIP A BIT
B06B 208A0F   V3     JSR $0F8A       SET POINTERS TO VAR.
B06E 209D1A          JSR GETVAR      PUT VAR. IN F.P.A
B071 20732D   V6     JSR STROUT
B074 3D              .BYTE '= ',0    PRINT'= '
B075 20
B076 00
B077 A592            LDA VARNAM      CHECK VAR TYPE
B079 1011            BPL V4          F.P ?=> V4
B07B ACA8B0          LDY STRFLG      RECOVER INDEX
B07E C8              INY             BUMP IT 2
B07F C8              INY
B080 B1AC            LDA (VARPTR),Y GET VAR. FROM MEMORY
B082 AA              TAX
B083 88              DEY
B084 B1AC            LDA (VARPTR),Y
B086 20DC1C          JSR PNUMBR      PRINT THE INTEGER
B089 4C92B0          JMP NEXT        MOVE TO NEXT VARIABLE
B08C          ;
B08C 20EC1C   V4     JSR ASCII       CONVERT TO ASCII
B08F 20CC0A          JSR BASPRT      PRINT IT
B092 206A2D   NEXT   JSR CRLF        DO A CR, LF
B095 A5AC            LDA VARPTR      BUMP POINTER TO NEXT
B097 18              CLC             SPOT ON TABLE
B098 6907            ADC #7
B09A 85AC            STA VARPTR
B09C 9002            BCC NX1
B09E E6AD            INC VARPTR+1
B0A0 A900     NX1    LDA #0          CLEAR FLAG
B0A2 8DA8B0          STA STRFLG
B0A5 4C2BB0          JMP V0          LOOP UNTIL DONE
B0A8          ;
B0A8 00       STRFLG .BYTE 0
B0A9 8CA8B0   STRING STY STRFLG      SAVE INDEX
```

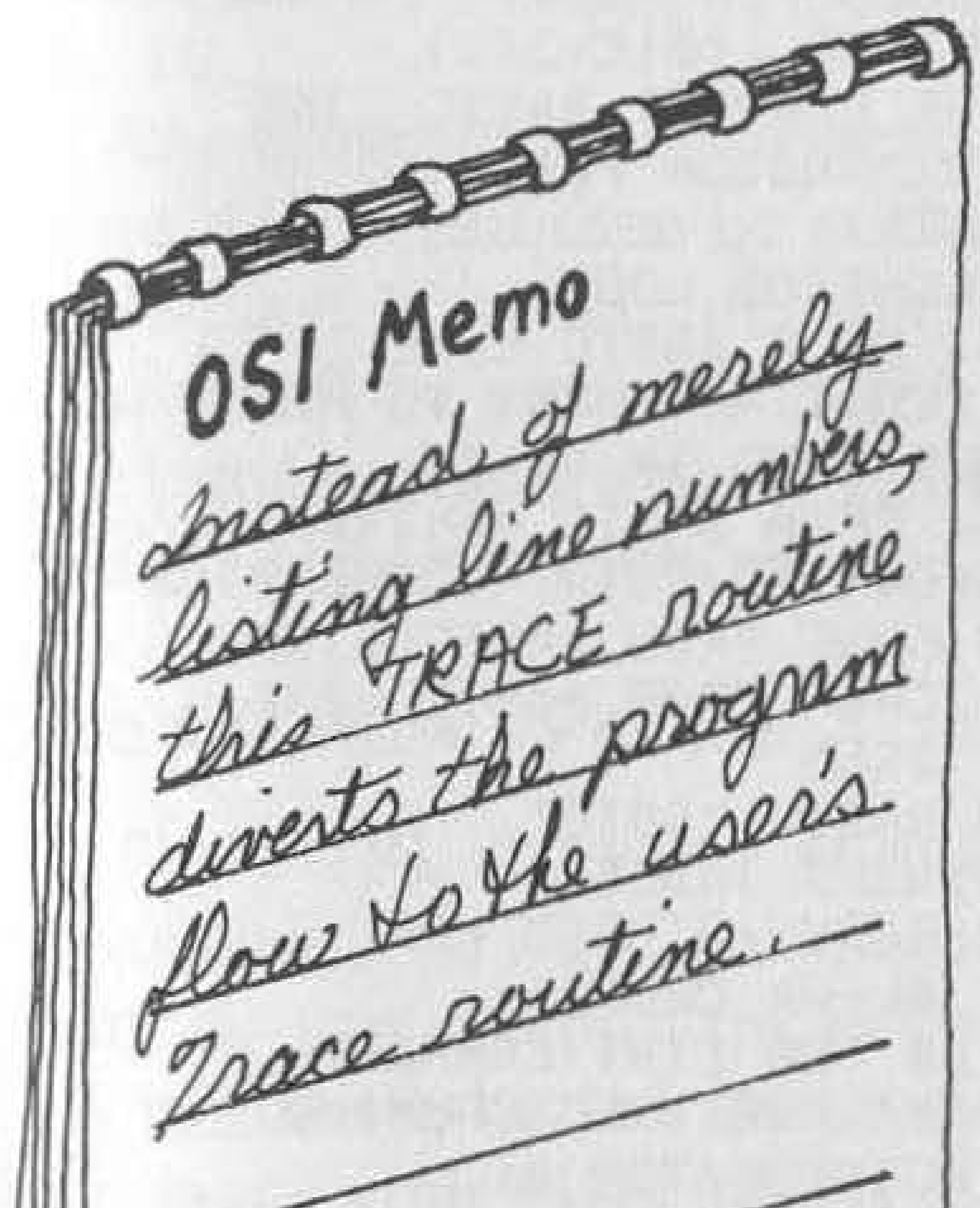
(continued)

Listing 2*(continued)*

```
B0AC 20732D          JSR STROUT      SHOW STRING VARIABLE
B0AF 24              .BYTE '$= ',0
B0B0 3D
B0B1 20
B0B2 00
B0B3 ACA8B0          LDY STRFLG      RECOVER INDEX
B0B6 C8              INY             BUMP IT ONE
B0B7 B1AC            LDA (VARPTR),Y  GET STRING LENGTH
B0B9 8DA8B0          STA STRFLG      SAVE IT
B0BC F0D4            BEQ NEXT        IF ZERO LENGTH, QUIT
B0BE CEA8B0          DEC STRFLG      DECREMENT LENGTH COUNTER
B0C1 C8              INY             BUMP INDEX
B0C2 B1AC            LDA (VARPTR),Y  GET ADDRESS OF STRING
B0C4 8594            STA VARPNT      SAVE IT
B0C6 C8              INY
B0C7 B1AC            LDA (VARPTR),Y
B0C9 8595            STA VARPNT+1
B0CB A000            LDY #0          INIZ INDEX
B0CD B194     STR1   LDA (VARPNT),Y  GET CHAR FROM MEMORY
B0CF 204323          JSR CHROUT      PRINT IT
B0D2 CCA8B0          CPY STRFLG      CHECK IF DONE
B0D5 F0BB            BEQ NEXT        YES, => NEXT
B0D7 C8              INY             NO, BUMP INDEX
B0D8 D0F3            BNE STR1        LOOP UNTIL DONE
```

Extended Trace

by Kerry Lourash



Extended trace is a vast improvement over trace programs that simply print line numbers. This assembly-language program is for OSI BASIC-in-ROM computers with a CTRL-C vector in RAM. X-Trace allows a BASIC subroutine to be called after execution of each and every statement in a subject program. You can design your own trace routine (in BASIC) to check variables, program flow, free memory space, etc. In addition, X-Trace provides options difficult to implement in BASIC.

My goal in designing X-Trace was to make it as self-contained and user friendly as possible. No zero-page locations are used by X-Trace. Vectors for the USR and CTRL-C routines are saved and then restored when X-Trace is done. Even a string variable used by X-Trace is stored within the program.

What X-Trace Does

X-Trace calls a BASIC subroutine, as opposed to the usual BASIC call to a machine-language subroutine. This technique allows you enormous flexibility and ease in designing a trouble-shooting routine. To further simplify the task, the starting line of the BASIC trace subroutine may be changed in mid-program, allowing multiple trace subroutines. Also, X-Trace stores the line number of the next statement to be executed in a string variable with a name you select. I call this string SUB$. In addition to the line number, SUB$ may contain subroutine nesting information. For example:

```
SUB$=50    *30*10
```

Listing 1

```
0000            ;********************
0000            ;*                  *
0000            ;* X-TRACE          *
0000            ;* BY KERRY LOURASH *
0000            ;*                  *
0000            ;********************
0000             ;
0000              ASCII=$B96E  CONVERT 2-BYTE HEX TO ASCII
0000              CFLAG=$212   CTRL C FLAG
0000               CONT=$A636  STORE VALUES FOR "CONT" COMMAND
0000              CTRLC=$21C   CTRL C VECTOR ($21C,21D)
0000             CURLIN=$87    HOLDS # OF CURRENT BASIC LINE
0000             DIMFLG=$5E    DEFAULT DIMENSION FLAG
0000                END=$80    "END" TOKEN
0000               EXEC=$A5C2  BASIC EXECUTION LOOP
0000               FIND=$A432  FIND LOC. OF A BASIC LINE
0000                FIX=$B408  CONVERT FLOATING POINT TO HEX
0000             FINSUB=$A1A4  FIND GOSUB INFO IN STACK
0000             FINVAR=$AD53  FIND LOC. OF A BASIC VARIABLE
0000              FLOAT=$B7E8  HEX TO FLOATING-POINT CONVERSION
0000             GETCHR=$00BC  GET NEXT CHAR FROM BASIC LINE
0000             GOTCHR=$00C2  GET SAME CHAR FROM BASIC LINE
0000              GOSUB=$8C    "GOSUB" TOKEN
0000               GOTO=$A6D0  ENTRY TO BASIC "GOTO"
0000             KEYTBL=$A084  BASIC KEYWORD TABLE
0000             KYPORT=$DF00  KEYBOARD INPUT STORED HERE
0000              LEGAL=$AD81  TEST FOR ALPHA CHAR.
0000             QUOFLG=$60    QUOTE FLAG FOR LIST COMMAND
0000             RETURN=$A6E8  ENTRY POINT "RETURN" COMMAND
0000               SIGN=$B0    SIGN OF ACCUMULATOR #1
0000             STONUM=$B774  STORE A VALUE IN BASIC VARIABLE
0000             TXTPNT=$C3    BASIC'S POINTER IN PROGRAM
0000                USR=$0B    USER VECTOR ($0B,0C)
0000             VARADD=$97    ADDRESS OF VARIABLE ($97,98)
0000             VARIBL=$93    NAME OF VARIABLE
0000             VARLOC=$95    LAST VARIABLE VALUE ADDRESS
0000             VARTYP=$5F    STRING OR NUMERIC FLAG
0000             YINDEX=$97    STORAGE FOR Y REG.
0000            ;
1000             *=$1000
1000            ;
1000            ; *****************
1000            ; SELECT A TRACE SUB
1000            ;
1000 A5B0        BRANCH LDA SIGN       GET SIGN OF F.P.A #1
1002 3012               BMI VECTOR     BRANCH IF NEGATIVE
1004 2008B4             JSR FIX        CONVERT TO HEX
1007 A511               LDA $11        IS NUMBER=0?
1009 0512               ORA $12
100B D072               BNE LISLIN     NO, LIST BASIC LINE
100D            ; *******************
100D            ; RESTORE CTRL C VECTOR
100D            ;
100D AD0D12      NORMAL LDA CSAVE
1010 AE0E12             LDX CSAVE+1
1013 4C5610             JMP V1
1016            ; ************************
1016            ; SAVE BASIC TRACE SUB'S
```

(continued)

SUB$ indicates that the next statement to be executed is in line 50. When tracing multistatement lines, the line number will be the same for every statement except the last, when the number of the next line will be in SUB$. Note that there are three spaces between 50 and the first asterisk. Spaces are used to pad the length of SUB$ to five characters. If LEN(SUB$) is greater than five, there is subroutine nesting information in SUB$. The numbers 30 and 10 indicate that the subject program is two levels deep in subroutines at this point. In other words, a RETURN command in the next statement would return to line 30, which was called by line 10.

Any program line can be stored in SUB$ with an X = USR (line #) command. SUB$ can then be printed or POKEd to a location in video memory for display. For example, the next line to be executed in the subject program could be stored in SUB$ with an X = USR(VAL(SUB$)) command. Any information formerly in SUB$ is erased, but it could be transferred to another string if necessary.

The CTRL-C Vector

The CTRL-C vector at $021C,$021D points to a ROM routine that checks for a CTRL-C command. A flag at $0212 can turn off the CTRL-C check so you can poll the keyboard. At the end of every BASIC statement this routine is called to see if you wish to stop the program. A CTRL-C halt saves your place in the BASIC program. If the program code is not altered, a CONT command causes the program to continue where it left off. The X-Trace program switches the CTRL-C vector to point to a machine-language program that calls a BASIC subroutine.

The GOSUB Command

When a line such as 100 GOSUB 300 is executed, the following happens:

1. The stack is checked to see if room is available for GOSUB information.
2. The parser pointer, the current line number (100), and a GOSUB token ($8C) are pushed onto the stack. An address ($A5FB) is already on the stack. The parser pointer is BASIC's "bookmark" that tells it where to resume execution when a RETURN is encountered.
3. The GOTO subroutine at $A6B9 reads the GOSUB's target line number (300), finds the line in the workspace, and prepares BASIC to resume execution at that line.
4. BASIC goes to the execution loop ($A5C2) and executes the subroutine.
5. When a RETURN is encountered, the parser pointer and current line number are pulled from the stack and restored. BASIC resumes execution at the statement after the GOSUB statement (after 100 GOSUB 300).

GOSUBs may be nested; that is, a GOSUB to a second subroutine can be done from the first subroutine. The second subroutine may contain a

Listing 1 *(continued)*

```
1016                ; STARTING LINE NUMBER.
1016                ; SAVE CTRLC VECTOR AND
1016                ; REPLACE WITH TRACE VECTOR
1016                ;
1016 297F    VECTOR AND #$7F        CHANGE SIGN OF NUMBER
1018 85B0           STA SIGN
101A 2008B4         JSR FIX         CONVERT TO HEX
101D 8C1012         STY TRASAV      SAVE START OF TRACE SUB
1020 8D1112         STA TRASAV+1
1023 20C200  VARBLE JSR GOTCHR      GET FIRST CHAR AFTER " )"
1026 2081AD         JSR LEGAL       IS IT A LETTER?
1029 9032           BCC ERR         NO, PRINT TM ERROR
102B 8DEC11         STA CHR1+1      STORE 1ST LETTER OF VAR.
102E A000           LDY #0
1030 20BC00         JSR GETCHR      GET 2ND CHAR AFTER " )"
1033 AA             TAX             SAVE IT IN X REG.
1034 F006           BEQ V2+1        BRANCH IF END OF STMT.
1036 20BC00         JSR GETCHR      SET PARSE POINTER AT END
1039 F003           BEQ V3+1        BRANCH ALWAYS
103B 2498    V2     BIT $98         CHAR=0 (TYA)
103D 248A    V3     BIT $8A         RESTORE CHAR (TXA)
103F 8DF011         STA CHR2+1      STORE 2ND VAR. LETTER
1042 AD1D02         LDA CTRLC+1     CTRL C ADDRESS <$F000?
1045 C9F0           CMP #$F0
1047 9013           BCC EXIT
1049 8D0E12         STA CSAVE+1
104C AD1C02         LDA CTRLC
104F 8D0D12         STA CSAVE
1052 A905    V0     LDA #XTRACE*256/256 REPLACE CTRL C
1054 A211           LDX #XTRACE/256     VECTOR WITH XTRACE
1056 8D1C02  V1     STA CTRLC
1059 8E1D02         STX CTRLC+1
105C 60      EXIT   RTS
105D 4CBCAA  ERR    JMP $AABC       PRINT TM ERR & EXIT
1060                ; ******************
1060                ; CHECK NEXT STATEMENT
1060                ; FOR "END" TOKEN
1060                ;
1060 A000    RTN    LDY #0           GET 1ST CHAR OF NEXT
1062 B1C3           LDA (TXTPNT),Y  STMT
1064 D002           BNE COLON        BRANCH IF NOT A NULL
1066 A004           LDY #4
1068 C8      COLON  INY
1069 B1C3           LDA (TXTPNT),Y
106B C980           CMP #END         IS 2ND CHAR AN "END" TOKEN?
106D D0ED           BNE EXIT         NO,BACK TO TRACE SUB
106F                ; *********************
106F                ; RESTORE USR VECTOR AND
106F                ; TRACE VECTOR.
106F                ; SIMULATE BASIC "RETURN"
106F                ; TO SUBJECT PROGRAM
106F                ;
106F AD1212         LDA USRSAV      RESTORE USER VECTOR
1072 850B           STA USR
1074 AD1312         LDA USRSAV+1
1077 850C           STA USR+1
1079 205210         JSR V0          RESTORE TRACE VECTOR
107C 4CE8A6         JMP RETURN
107F                ; *******************
107F                ; STORE A LINE IN SUB$
```

(continued)

call to a third subroutine, and so forth. X-Trace finds the subroutine calls on the stack and stores their return line numbers in SUB$.

How X-Trace Works

The user sets the USR vector to the BRANCH routine and calls X-Trace with:

X = USR(negative trace subroutine starting line number)variable

For example, X=USR(−260)SU specifies that the starting line of the BASIC trace subroutine is at line 260 and the trace variable is SU$ (or SUB$, as I call it). The BRANCH routine goes to VECTOR, which saves the line number of the trace subroutine (the stock CTRL-C vector) and points the CTRL-C vector at XTRACE. VECTOR returns to BASIC, which executes the first statement in the subject program.

At the end of the statement, the CTRL-C vector sends BASIC to the XTRACE routine. XTRACE does the following:

1. Checks the CTRL-C
2. Saves the current USR vector
3. Saves the subject program's line number in SUB$
4. Finds subroutine calls in the stack and stores them in SUB$
5. Simulates a GOSUB to the BASIC trace subroutine

While in the trace subroutine, you have the option of storing a BASIC line in SUB$. The format is: X=USR(line number). You can PRINT the string or POKE it somewhere in video memory.

The RTN routine looks for an END command in the next statement to be executed. When RTN detects an END, a simulated RETURN to the subject program is performed (don't worry; you can use END in the subject program without side effects). The NORMAL routine is called with an X=USR(0). It restores the normal CTRL-C vector and turns off X-Trace. The USR vector must be set to the BRANCH routine when the USR command is executed.

```
5 REM PROGRAM #1
10 POKE11,0:POKE12,16
20 X=USR( -100 )SU
30 GOSUB50
40 X=USR( 0 ):END
50 GOSUB60:RETURN
60 GOSUB70:RETURN
70 GOSUB80:RETURN
80 GOSUB90:RETURN
90 RETURN
100 PRINTSUB$:END
110 END
```

Here are three short programs to demonstrate X-Trace. Program 1 is a demonstration of the subroutine nesting display of X-Trace. Lines 10 and 20 set the USR vector to the start of the X-Trace program and specify the subroutine's starting line number (100) and the string used by XTRACE (SU$). Next, a series of GOSUBs fills SUB$ with subroutine information. Line 40 turns off X-Trace and ends the program.

Listing 1 *(continued)*

```
107F               ;
107F  A900    LISLIN LDA #0          SET LEN SUB$=0
1081  8D0F12         STA LENCNT
1084  2032A4         JSR FIND        FIND LINE IN BASIC WORKSPACE
1087  905C           BCC XIT         EXIT IF NOT FOUND
1089  A611           LDX $11
108B  A512           LDA $12
108D  20D711         JSR CONVRT      CHANGE LINE# TO ASCII
1090  A2FF           LDX #$FF
1092  E8      L6     INX
1093  BD0101         LDA $101,X
1096  9D1412         STA SUB$,X      STORE ASCII IN SUB$
1099  D0F7           BNE L6
109B  A920           LDA #$20
109D  9D1412         STA SUB$,X
10A0  8E0F12         STX LENCNT
10A3  A001           LDY #1          CLEAR QUOTE FLAG
10A5  8460           STY QUOFLG
10A7  A003           LDY #3
10A9  D011           BNE L1
10AB  A497    L5     LDY YINDEX      RESTORE BASIC LINE PNTR.
10AD  297F    L0     AND #$7F        ZERO HI BIT
10AF  20FA10         JSR STORE
10B2  C922           CMP #$22        IS CHAR A "?
10B4  D006           BNE L1
10B6  A560           LDA QUOFLG      TOGGLE QUOTE FLAG
10B8  49FF           EOR #$FF
10BA  8560           STA QUOFLG
10BC  C8      L1     INY             GET NEXT CHAR
10BD  B1AA           LDA ($AA),Y
10BF  F024           BEQ XIT         BRANCH IF IT'S A NULL
10C1  10EA           BPL L0          BRANCH IF NOT A TOKEN
10C3  2460           BIT QUOFLG      CHECK QUOTE FLAG
10C5  30E6           BMI L0
10C7  38             SEC             SUBTRACT 7F FROM TOKEN
10C8  E97F           SBC #$7F
10CA  AA             TAX             RESULT IN X REG
10CB  8497           STY YINDEX
10CD  A0FF           LDY #$FF        FIND KEYWORD
10CF  CA      L2     DEX
10D0  F008           BEQ L4          BRANCH IF FOUND
10D2  C8      L3     INY
10D3  B984A0         LDA KEYTBL,Y
10D6  10FA           BPL L3
10D8  30F5           BMI L2
10DA  C8      L4     INY             GET CHAR
10DB  B984A0         LDA KEYTBL,Y
10DE  30CB           BMI L5          BRANCH IF LAST CHAR
10E0  20FA10         JSR STORE       STORE CHAR IN SUB$
10E3  D0F5           BNE L4          BRANCH ALWAYS
10E5               ;
10E5  20E411  XIT    JSR STRING
10E8  BA             TSX             GET STACK POINTER
10E9  E8      X0     INX             FIND $A5FB CALL ON STACK
10EA  BD0101         LDA $101,X
10ED  C9FB           CMP #$FB
10EF  D0F8           BNE X0
10F1  BD0201         LDA $102,X
10F4  C9A5           CMP #$A5
10F6  D0F1           BNE X0
```

(continued)

```
5 REM PROGRAM #2
10 POKE11,0:POKE12,16
20 X=USR(-60)LI
30 FORA=1TO10
40 B=B+C:C=C-1
50 NEXTA:END
60 V=VAL(LI$)
70 IFA=B OR ABS(C)=A
   THENPRINT PRE;A;B;C
80 PRE=V:END
```

Program 2 shows how to monitor the value of variables and store the previous statement number (PRE). When a variable changes in the subject program, you may want to know the exact line number in which the change occurred. X-Trace stores only the *next* statement number to be executed. *Note:* I recommend the use of a single subscripted variable in the trace subroutine (such as XY_1, XY_2, XY_3, etc.) to avoid conflict with variables in the subject program.

Program 3 shows how to switch BASIC trace subroutines. In this example, the trace subroutines are switched within the trace subroutines themselves. You can switch subroutines in the subject program, but that's a less tidy method, since you might forget to delete those lines from the subject program after they have served their purpose.

```
5 REM PROGRAM #3
20 X=USR(-60)N
30 FORI=1TO10
40 NEXTI
50 END
60 PRINTVAL(N$)
70 IFI=4THENX=USR(-90)N
80 END
90 PRINTUSR(VAL(N$)):PRINTN$
100 IFI=5THENX=USR(-60)N
110 END
```

Converting X-Trace

Please note the two changes necessary to convert X-Trace to C2/4P operation. They are located right after the "START of XTRACE" heading. *Always remember to isolate the BASIC trace subroutine from normal program flow so it doesn't try to trace itself.* I have tried to make the stack-handling routines as general as possible, but X-Trace may not be compatible with some modified USR or CALL routines.

Formatting

The major difficulty when tracing a program is displaying the information generated without clobbering the subject program's output. I list only a few methods.

1. Call $FD00 and build a string from keyboard input without writing to the screen.
2. Turn the screen output flag ($64) off and on to control output.
3. Slow the video output rate with a POKE to location $206 or a SAVE command.
4. POKE SUB$ to the screen at a point not used by BASIC.

See other reference sources for more solutions.

Listing 1 *(continued)*

```
10F8 9A              TXS                   SET STACK: BYPASS USR
10F9 60              RTS
10FA                ;
10FA AE0F12   STORE  LDX LENCNT            STORE A CHAR IN SUB$
10FD E8              INX
10FE 9D1412          STA SUB$,X
1101 8E0F12          STX LENCNT
1104 60              RTS
1105                ;
1105                ;*************************
1105                ;* START OF XTRACE ROUTINE *
1105                ;*************************
1105                ; DO CTRL C CHECK AND
1105                ; RESTORE CTRL C VECTOR
1105                ; IF IN IMMEDIATE MODE
1105                ;
1105 AD1202  XTRACE LDA CFLAG             GET CTRL C FLAG
1108 D019            BNE IMMED             SKIP CHECK IF FLAG SET
110A A9FE            LDA #$FE              #1 IF C2/4P***************
110C 8D00DF          STA KYPORT
110F 2C00DF          BIT KYPORT
1112 700F            BVS IMMED
1114 A9FB            LDA #$FB              #4 IF C2/4P***************
1116 8D00DF          STA KYPORT
1119 2C00DF          BIT KYPORT
111C 7005            BVS IMMED
111E A903            LDA #3
1120 4C36A6          JMP CONT              EXIT IF CTRL C HIT
1123                ;
1123 A588    IMMED  LDA CURLIN+1          IN IMMEDIATE MODE?
1125 C9FF            CMP #$FF
1127 D003            BNE SAVUSR            NO, BRANCH
1129 4C0D10          JMP NORMAL            RESTORE C VECTOR & RTS
112C                ; *********************
112C                ; SAVE PROGRAM'S USR
112C                ; VECTOR & POINT CTRL C
112C                ; VECTOR AT RTN ROUTINE
112C                ;
112C A50B    SAVUSR LDA USR               SAVE USER VECTOR
112E 8D1212          STA USRSAV
1131 A50C            LDA USR+1
1133 8D1312          STA USRSAV+1
1136 A900            LDA #BRANCH*256/256   BRANCH VECTOR IN
1138 850B            STA USR                USER VECTOR
113A A910            LDA #BRANCH/256
113C 850C            STA USR+1
113E A960            LDA #RTN*256/256      RTN VECTOR IN
1140 8D1C02          STA CTRLC              CTRL C VECTOR
1143 A910            LDA #RTN/256
1145 8D1D02          STA CTRLC+1
1148                ; ***********************
1148                ; SAVE CURRENT LINE NUMBER
1148                ; IN "SUB$" VARIABLE
1148                ;
1148 A000    STORLI LDY #0                 IF WE ARE NOT AT END OF
114A B1C3            LDA (TXTPNT),Y LINE, LINE# IS IN CURLIN
114C D00B            BNE CURENT
114E A003            LDY #3                 GET NEXT LINE# FROM
1150 B1C3            LDA (TXTPNT),Y BASIC WORKSPACE
1152 AA              TAX
```

(continued)

Listing 1 *(continued)*

```
1153 C8              INY
1154 B1C3            LDA (TXTPNT),Y
1156 C8              INY
1157 D004            BNE NEXTLI
1159 A588     CURENT LDA CURLIN+1   GET LINE# FROM CURLIN
115B A687            LDX CURLIN
115D 20D711   NEXTLI JSR CONVRT     CHANGE LINE# TO ASCII
1160 A0FF            LDY #$FF
1162 C8       NO     INY
1163 B90101          LDA $101,Y
1166 991412          STA SUB$,Y     STORE ASCII IN SUB$
1169 D0F7            BNE NO
116B A920            LDA #$20       PAD TO 5 DIGITS
116D 991412   N1     STA SUB$,Y     WITH SPACES
1170 C8              INY
1171 C005            CPY #5
1173 D0F8            BNE N1
1175 88              DEY
1176 8C0F12          STY LENCNT     LENCNT=5
1179          ; ********************
1179          ; FIND SUBROUTINE CALLS
1179          ; IN THE STACK & STORE
1179          ; THEM IN SUB$ VARIABLE
1179          ;
1179 BA              TSX
117A 20A4A1   NEXSUB JSR FINSUB     LOOK FOR SUBS ON STACK
117D C98C            CMP #GOSUB
117F D034            BNE SUB        BRANCH IF NO MORE SUBS
1181 AD0F12          LDA LENCNT     GET LENGTH OF SUB$
1184 C943            CMP #67
1186 B02D            BCS SUB        BRANCH IF =>67
1188 E8              INX
1189 BD0101          LDA $101,X     GET LINE #'S FROM STACK
118C 85AE            STA $AE
118E E8              INX
118F BD0101          LDA $101,X
1192 85AD            STA $AD
1194 8A              TXA            CONVERT LINE #'S TO
1195 48              PHA            ASCII AT $100-10C
1196 20DB11          JSR CON
1199 A92A            LDA #'*        FIRST, STORE A "*"
119B 20FA10          JSR STORE
119E A000            LDY #0
11A0 C8       NEXCHR INY
11A1 E8              INX
11A2 B90001          LDA $100,Y     GET ASCII DIGIT
11A5 9D1412          STA SUB$,X     PUT IT IN SUB$
11A8 D0F6            BNE NEXCHR     LOOP IF NOT A NULL
11AA 8E0F12          STX LENCNT     SAVE LENGTH OF SUB$
11AD 68              PLA            RESTORE STACK INDEX
11AE AA              TAX            INCR PAST SUB INFO
11AF E8              INX
11B0 E8              INX
11B1 E8              INX
11B2 4C7A11          JMP NEXSUB     LOOK FOR ANOTHER SUB
11B5          ;
11B5          ; ****************
11B5          ; PUSH GOSUB INFOR-
11B5          ; MATION ONTO STACK
11B5          ;
11B5 20E411   SUB    JSR STRING
11B8 A5C4            LDA TXTPNT+1   PUSH PARSER POINTER
```

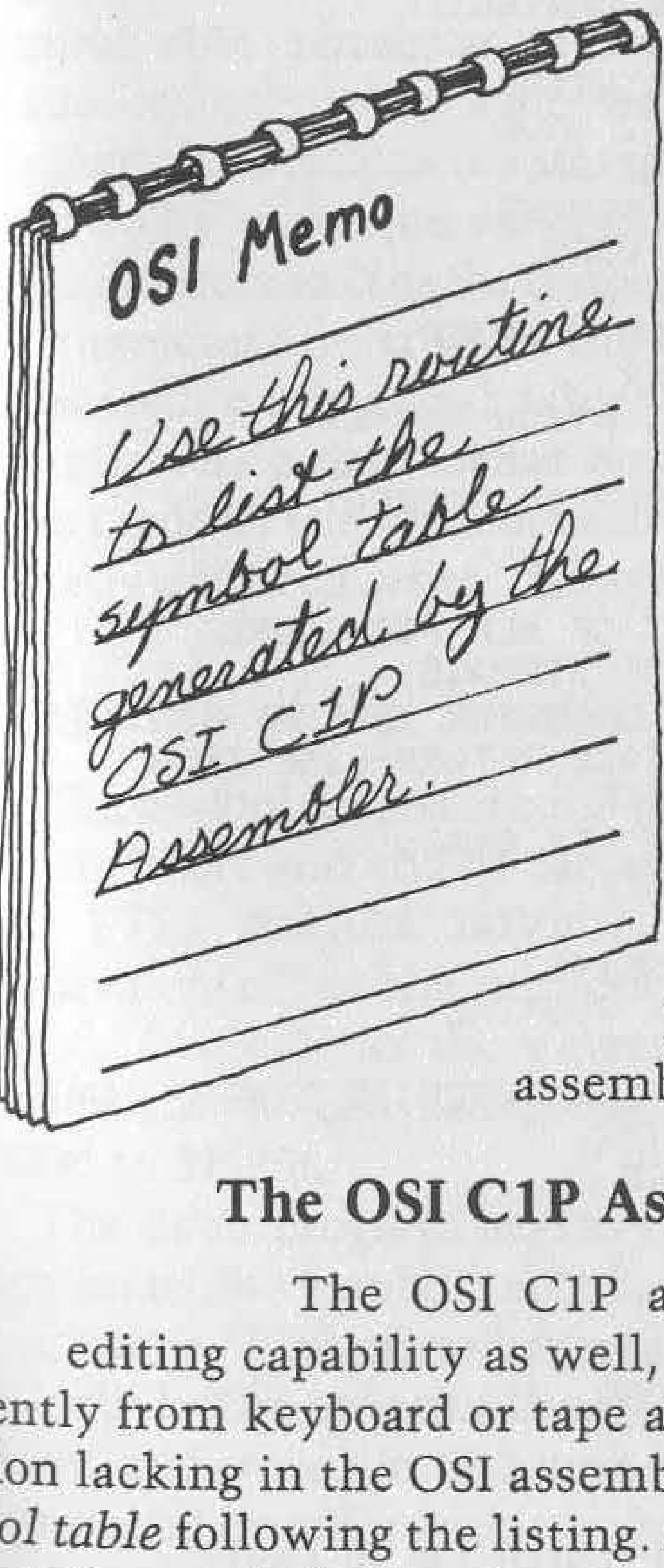
(continued)

Listing 1 *(continued)*

```
11BA 48                 PHA
11BB A5C3               LDA TXTPNT
11BD 48                 PHA
11BE A588               LDA CURLIN+1  PUSH CURRENT LINE#
11C0 48                 PHA
11C1 A587               LDA CURLIN
11C3 48                 PHA
11C4 A98C               LDA #GOSUB    PUSH "GOSUB" TOKEN
11C6 48                 PHA
11C7          ; ********************
11C7          ; DO A SIMULATED GOSUB
11C7          ; TO THE BASIC TRACE SUB
11C7          ;
11C7 AD1012             LDA TRASAV
11CA 8511               STA $11
11CC AD1112             LDA TRASAV+1
11CF 8512               STA $12
11D1 20D0A6             JSR GOTO      SET UP GOTO INFO
11D4 4CC2A5             JMP EXEC      JUMP TO BASIC EXEC LOOP
11D7          ; *******************
11D7          ; HEX TO ASCII AT $100
11D7          ;
11D7 85AD      CONVRT   STA $AD
11D9 86AE               STX $AE
11DB A290      CON      LDX #$90
11DD 38                 SEC
11DE 20E8B7             JSR FLOAT     HEX TO FLOATING POINT#
11E1 4C6EB9             JMP ASCII     F.P.TO ASCII AT $100-10C
11E4          ; ************************
11E4          ; FIND OR CREATE SUB$ VAR.
11E4          ;
11E4 A0FF      STRING   LDY #$FF      SPECIFY STRING VAR.
11E6 845F               STY VARTYP
11E8 C8                 INY           SET DIMFLG=0
11E9 845E               STY DIMFLG
11EB A953      CHR1     LDA #'S       VARIABLE NAME =SU$
11ED 8593               STA VARIBL
11EF A955      CHR2     LDA #'U
11F1 0980               ORA #$80      SET HI BIT OF "U"
11F3 8594               STA VARIBL+1
11F5 2053AD             JSR FINVAR    FIND OR CREATE SU$
11F8 A000               LDY #0
11FA EE0F12             INC LENCNT
11FD AD0F12             LDA LENCNT     SET LENGTH OF SU$
1200 9195               STA (VARLOC),Y
1202 C8                 INY
1203 A914               LDA #SUB$*256/256 STORE LOCATION
1205 9195               STA (VARLOC),Y    OF SU$
1207 C8                 INY
1208 A912               LDA #SUB$/256
120A 9195               STA (VARLOC),Y
120C 60                 RTS
120D          ; ***********
120D          ; STORAGE AREA
120D          ;
120D 0000      CSAVE  .WORD 0        CTRL C VECTOR STORAGE
120F 00        LENCNT .BYTE 0        LINE LENGTH COUNT
1210 0000      TRASAV .WORD 0        TRACE SUB LINE STORAGE
1212 0000      USRSAV .WORD 0        USER VECTOR STORAGE
1214           SUB$
125B           *=*+71                72-BYTE SUB TABLE
125B                                ;OR LINE STRING
```

Symbol Table Lister

by Rolf Johannesen



Programming in assembly rather than a high-level language (BASIC, Pascal) may be preferred for one of three reasons: speed, economy of memory usage, and the ability to do things not available in the higher-level languages. Small sections of code can be assembled by hand and entered using a simple monitor. However, this is a tedious process and prone to error. For any serious assembly-language coding an assembler program must be used. An assembler will read source code, check for errors, generate all necessary cross-references, and produce the desired assembled code. A listing may be produced optionally by the assembler.

The OSI C1P Assembler/Editor

The OSI C1P assembler does all of the above and has editing capability as well, so the user can enter source code conveniently from keyboard or tape and edit it before assembly. One useful option lacking in the OSI assembler is the ability to list or print out a *symbol table* following the listing. A symbol table lists all symbols and labels, together with their assigned values, and is a valuable adjunct in reading a program listing. When modifying a program, the symbol table helps you avoid inadvertent duplication of symbols or labels. A complete cross-reference program would be even more useful and would not be difficult to write. For my own use, the extra effort and extra memory required did not seem to be worthwhile. This article presents a symbol table lister for the OSI C1P. The listing included here is for the OS65D disk system; comments indicate changes needed to run the program with the cassette version of the assembler.

Listing 1

```
 10                  ;     SYMBOL TABLE LISTING PROGRAM
 20                  ;     FOR OS65D V3.3
 30                  ;     COMMENTS GIVE CHANGES FOR CASSETTE
 40                  ;     BASED OSI ASSEMBLER
 50                  ;     BY ROLF B. JOHANNESEN
 60                  ;     13917 CONGRESS DRIVE
 70                  ;     ROCKVILLE, MD  20853
 80                  ;     LAST REVISION 28 NOV 82
 90                  ;     PAGE ZERO LOCATIONS
100 0010=            CC   = $10          CHARACTER COUNTER
110 0011=            CSV  = CC+1         SAVED CHARACTER
120 0012=            MCTR= CSV+1         MULT. CHAR. COUNTER
130 0013=            XP   = MCTR+1        X POINTER
140 0014=            XSV  = XP+1         X REG. SAVE
150 0015=            YSV  = XSV+1        Y REG. SAVE
160 0016=            LN   = YSV+1        LINE NUMBER
170 0018=            LW   = LN+2         LAST WORD
180 001A=            PTR  = LW+2         POINTER
190 001C=            PTR2= PTR+2         SECOND POINTER
200 001E=            BFR  = PTR2+2       BUFFER
210 0026=            DEST= BFR+8         DESTINATION BUFFER
220 002E=            M    = DEST+8       MINIMUM SYMBOL VALUE
230 0032=            MP   = M+4          MINIMUM IN CURRENT LOOP
240 0036=            BCB  = MP+4
250                  ;     ADDRESS EQUATES
260 12C9=            STMEM=$12C9         START OF MEM FOR SOURCE
270 12CB=            STS  = $12CB        TOP OF STORAGE
280 12FE=            NL   = $12FE        NEXT LOCN FOR SOURCE
290 1A56=            CRL  = $1A56        CARRIAGE RETURN-LINE FEED
300                  ; CRL = $A86C FOR CASSETTE
310 19E9=            PHEX= $19E9         PRINT HEX CHAR.
320                  ; PHEX INTERNAL FOR CASSETTE
330 1DD6=            DVD  = $1DD6        16-BIT DIVIDE ROUTINE
340                  ; DVD  INTERNAL FOR CASSETTE
350 2343=            PRINT=$2343         PRINT ROUTINE
360                  ; PRINT=$FFEE FOR CASSETTE
370 2F83=            LL   = $2F83        LAST LINE USED IN SYMBOL TABLE
380                  ; LL   = $000A FOR CASSETTE
390                  ;     PROGRAM STARTS HERE
400 1F3E                 *=$1F3E
410                  ;    *=$1391 FOR CASSETE
420 1F3E A900        STRT LDA #0
430 1F40 852E             STA M          INITIALIZE MINIMUM
440 1F42 852F             STA M+1        TO ZERO
450 1F44 8530             STA M+2
460 1F46 8531             STA M+3
470 1F48 38               SEC
480 1F49 AD832F           LDA LL         SET POINTER LW TO LAST
490 1F4C E904             SBC #4
500 1F4E 8518             STA LW
510 1F50 AD842F           LDA LL+1       LOCN IN SYMBOL TABLE
520 1F53 E900             SBC #0
530 1F55 8519             STA LW+1
540 1F57 20561A           JSR CRL
550 1F5A A9FF        LOOP1 LDA #$FF      MAKE MP > ANY POSSIBLE
560 1F5C 8533             STA MP+1       SYMBOL
570 1F5E ACCB12      LOOP2 LDY STS       SET PTR+Y TO TOP
580 1F61 ADCC12           LDA STS+1      OF SYMBOL TABLE
```

(continued)

Operation of the Assembler/Editor

In the OSI assembler, source code is stored in memory as it is read in, beginning at the location following the end of the assembler. Numbered lines are inserted at their correct position. Each line begins with two bytes containing the line number in hex in the order low, high. The line ends with a return ($0D). Line feeds are not stored in the source text but are added after each return during printing. There is no special signal to indicate end-of-text as in BASIC; rather the editor keeps the next location available for text in a table (see below.) When an assembly is requested, a symbol table is built, which begins at the last location in RAM and moves to successively lower addresses as more symbols are added. Each symbol requires six locations for storage: four bytes for the symbol itself (encoded) and two bytes for the value of the symbol. A symbol may be from one to six characters in length. It must begin with an alphabetic and the remaining characters must be in the set A-Z, 0-9, :, ., or $. The symbol table is not sorted, nor is a hash table used; the symbols are simply entered in the order in which they are encountered. A forward reference causes an entry to be made in the symbol table with a value that appears to be random. A value is adjusted when the symbol is defined.

Operation of the Symbol Table Lister

The assembler maintains pointers to the start and end of source code and the start and end of the symbol table. These are shown as STMEM, NL, STS, and LL in the accompanying listing. Let me define "equivalence" as the numerical representation in which the symbol is stored, "value" as the value assigned to the symbol; e.g., "LABEL" always has the equivalence $4B2A2120; its value may be anything from $0000 to $FFFF.

The lister program begins by zeroing a 4-byte memory location, M. It then scans the symbol table to find the smallest equivalence greater than or equal to M (the smallest symbol numerically is also the earliest alphabetically). The value of the found minimum equivalence is incremented by one and stored in M before the table is searched again. Thus, the table is searched once for each symbol to be printed. This method is not as efficient as a true sort, but it requires less memory. For a table of 100 symbols, the output is only slightly slower than the rate at which characters are written to the screen. After the minimum equivalence has been found in a particular pass (lines 550-1310), the symbol is decoded into its ASCII value (lines 1320-1900). The ASCII representation of the symbol is searched for multiple characters and converted to the form used by the assembler for source code (e.g., L666 = $4C363636 → $4C36FE) (lines 1910-2360).

Next, the source file is searched for the line defining the symbol (lines 2370-2780). If the symbol is not defined (and this will have caused an

Listing 1 *(continued)*

```
1F64 851B         STA PTR+1       DECREMENT Y AS TABLE
1F66 A900         LDA #0          IS READ
1F68 851A         STA PTR
1F6A C010   LOOP3 CPY #$10        WHEN Y GETS BELOW $10
1F6C B00E         BCS TRN         ADD $80 AND DECREMENT
1F6E 98           TYA             PTR BY $80 TO AVOID
1F6F 0980         ORA #$80        ADDRESSING ERRORS IF
1F71 A8           TAY             Y DECREMENTS FROM
1F72 A51A         LDA PTR         00 TO FF
1F74 4980         EOR #$80
1F76 851A         STA PTR
1F78 1002         BPL TRN
1F7A C61B         DEC PTR+1
1F7C 98     TRN   TYA             COMPARE PTR+Y TO LW
1F7D 38           SEC             TO SEE IF SEARCH ENDED
1F7E E903         SBC #3
1F80 A8           TAY
1F81 18           CLC
1F82 651A         ADC PTR
1F84 08           PHP
1F85 C518         CMP LW
1F87 D011         BNE CONT
1F89 28           PLP
1F8A A51B         LDA PTR+1
1F8C 6900         ADC #0
1F8E C519         CMP LW+1
1F90 D007         BNE CM1         IF MP+1=$FF THEN
1F92 A533         LDA MP+1        SYMBOL TABLE EXHAUSTED
1F94 C9FF         CMP #$FF         SO QUIT   BUT IF
1F96 D041         BNE PRNT        MP+1<$FF THEN A SYMBOL
            ;                      HAS BEEN FOUND  PRINT IT
1F98 60           RTS
            ; CHANGE RTS TO JMP $1300 FOR CASSETTE
1F99 08     CM1   PHP
1F9A 28     CONT  PLP             DOUBLE LOOP FOR 32-BIT
1F9B A200         LDX #0          SUBTRACT
1F9D 38     CLOOP SEC             WHEN X=0, COMPARE
1F9E B11A         LDA (PTR),Y     CURRENT VALUE IN SYMBOL
1FA0 F530         SBC M+2,X       TABLE WITH  M  IF VALUE
1FA2 C8           INY             IS <M THEN OMIT 2d LOOP
1FA3 B11A         LDA (PTR),Y     IF VALUE=>M THEN
1FA5 F531         SBC M+3,X       COMPARE CURRENT VALUE
1FA7 88           DEY             WITH MINIMUM (THIS LOOP)
1FA8 88           DEY             IN MP  IF VALUE=>MP THEN
1FA9 88           DEY             CONTINUE SEARCH     BUT
1FAA B11A         LDA (PTR),Y     IF VALUE<MP THEN
1FAC F52E         SBC M,X         REPLACE MP BY
1FAE C8           INY             NEW MINIMUM
1FAF B11A         LDA (PTR),Y
1FB1 F52F         SBC M+1,X
1FB3 08           PHP
1FB4 E000         CPX #0
1FB6 D008         BNE TMP
1FB8 28           PLP
1FB9 9019         BCC NXWORD
1FBB C8           INY
1FBC A204         LDX #4
1FBE D0DD         BNE CLOOP
1FC0 28     TMP   PLP
1FC1 B011         BCS NXWORD
```

(continued)

assembler error) the lister program prints a ? instead of a line number. Additionally, if the symbol is more than two characters long, the fourth character will be an embedded ?. Finally, the symbol, its value, and the line number where defined are all printed out (lines 2790-3130). This process is repeated until all symbols have been found and printed.

Inasmuch as the extended monitor (EM) is always loaded together with the assembler in OS65D, the program uses EM routines where possible (DIVIDE and PHEX). These routines are listed as comments to be assembled and used with the cassette-based assembler. Print and carriage-return line-feed routines are available in both OS65D and BASIC-in-ROM; addresses are given for both.

The program as given for OS65D uses memory from $1F3E to $218F. It starts one location above the end of the EM and can be stored on disk with the EM to be called in each time the assembler is loaded. For 5-inch disks this is Track 10; for 8-inch disks it is Track 7. The symbol table lister should be called immediately only after an assembly (A0-A3) has been run. Then type !GO 1F3E in response to the prompt character.

The program listed here begins at $1391 and runs to $1619. The value in STMEM has been changed accordingly to $161A. Note that this change must be made as soon as the assembler is loaded, before any source code is entered. This reduces the space available for an assembler source file by $289 (649 decimal) locations. If this reduction in space turns out to be crucial, the lister could be relocated to overlay part of the assembler. If this is done, the part of the assembler to be overlaid should be stored on tape. The assembler can then be reused by loading only the short overlay file rather than the entire program. The lister uses some page-zero locations for storage but does not change any values required by the assembler, so the assembler can be re-run after running the lister. Output goes to the print vector at $FFEE, which is a JMP (indirect) to $021A, 021B. These locations are initialized by the monitor to send output to the screen or tape, depending on the value in $0205. They can, of course, be changed to point to a print routine if a printer is available.

Listing 1 *(continued)*

```
1FC3 A200          LDX #0
1FC5 88            DEY
1FC6 B11A     MVMP LDA (PTR),Y        COPY SYMBOL (CODED) AND
1FC8 9532          STA MP,X           ITS VALUE FROM PTR+Y
1FCA C8            INY                INTO MP
1FCB E8            INX
1FCC E006          CPX #6
1FCE D0F6          BNE MVMP
1FD0 98            TYA
1FD1 E905          SBC #5
1FD3 A8            TAY
1FD4 88       NXWORD DEY
1FD5 88            DEY
1FD6 4C6A1F        JMP LOOP3
1FD9 A208     PRNT LDX #8             FILL PRINT BUFFER
1FDB A920          LDA #$20           WITH SPACES
1FDD 951D     STB STA BFR-1,X
1FDF CA            DEX
1FE0 D0FB          BNE STB
1FE2 B532     CPM LDA MP,X            COPY CURRENT MINIMUM TO
1FE4 952E          STA M,X            GLOBAL MINIMUM
1FE6 E8            INX
1FE7 E004          CPX #4
1FE9 D0F7          BNE CPM
1FEB E630          INC M+2            INCREMENT GLOBAL MIN.
1FED D002          BNE LOOP3.         FOR NEXT PASS
1FEF E631          INC M+3
1FF1 A000     LOOP3. LDY #0           NOTE LOOP3. NOT= LOOP3
1FF3 8413          STY XP
1FF5 A200     LOOP4 LDX #0            DECODE MAX OF 6 BYTES
1FF7 B93200   LOOP4P LDA MP,Y          DIVIDE BY 1600, THEN 40
1FFA 85CC          STA $CC            REMAINDERS ARE BYTES
1FFC C8            INY                OF SYMBOL
1FFD B93200        LDA MP,Y
2000 85CD          STA $CD
2002 C8            INY
2003 206221   LOOP5 JSR DVR
2006 F046          BEQ GADR           IF QUOTIENT=01 TO $1A THEN
2008 C91B     NXCHR CMP #$1B          ALPHABETIC  ADD $40
200A 900A          BCC ALPH           IF QUOTIENT=$1B TO $24 THEN
200C C925          CMP #$25           NUMERIC     ADD $15
200E 9008          BCC NUM            IF QUOTIENT>$24 THEN : . OR $
2010 AA            TAX                TABLE LOOK-UP
2011 BD6721        LDA CHR-$25,X
2014 D004          BNE PP
2016 692B     ALPH ADC #$2B
2018 6915     NUM ADC #$15
201A A613     PP   LDX XP
201C 951E          STA BFR,X          PUT ASCII CHAR INTO BFR
201E E613          INC XP
2020 E005          CPX #5
2022 F02A          BEQ GADR
2024 E002          CPX #2
2026 D004          BNE TSR
2028 A415          LDY YSV
202A D0C9          BNE LOOP4
202C A5C8     TSR LDA $C8
202E D004          BNE TSTX
2030 A5C9          LDA $C9
2032 F01A          BEQ GADR
```

(continued)

Listing 1 *(continued)*

```
1790 2034  A614      TSTX LDX XSV
1800 2036  E004           CPX #4
1810 2038  D008           BNE LPREP
1820 203A  A5C8           LDA $C8
1830 203C  A000           LDY #0
1840 203E  84C8           STY $C8
1850 2040  F0C6           BEQ NXCHR
1860 2042  A5C8      LPREP LDA $C8
1870 2044  85CC           STA $CC
1880 2046  A5C9           LDA $C9
1890 2048  85CD           STA $CD
1900 204A  A415           LDY YSV
1910 204C  D0B5           BNE LOOP5
1920 204E  A200      GADR LDX #0          PRINT 8 CHARS FROM BFR
1930 2050  B51E      GB$ LDA BFR,X
1940 2052  204323         JSR PRINT
1950 2055  E8             INX
1960 2056  E008           CPX #8
1970 2058  D0F6           BNE GB$
1980 205A  A205           LDX #5
1990 205C  B532           LDA MP,X        PRINT SAVED VALUE OF
2000 205E  20E919         JSR PHEX        SYMBOL (CURRENT LOOP)
2010 2061  CA             DEX             IN HEX
2020 2062  B532           LDA MP,X
2030 2064  20E919         JSR PHEX
2040 2067  A000           LDY #0          SET UP SEARCH OF ASCII
2050 2069  A200           LDX #0          SYMBOL FOR DUPLICATE
2060 206B  8612           STX MCTR        CHARACTERS
2070 206D  8611           STX CSV
2080 206F  B91E00    LOOP6 LDA BFR,Y
2090 2072  C8             INY
2100 2073  C920           CMP #$20
2110 2075  F01C           BEQ CXIT
2120 2077  C511           CMP CSV
2130 2079  F014           BEQ DUPL
2140 207B  48             PHA
2150 207C  A512           LDA MCTR
2160 207E  F007           BEQ STOR
2170 2080  9526           STA DEST,X
2180 2082  E8             INX
2190 2083  A900           LDA #0
2200 2085  8512           STA MCTR
2210 2087  68        STOR PLA
2220 2088  9526           STA DEST,X
2230 208A  E8             INX
2240 208B  8511           STA CSV
2250 208D  D0E0           BNE LOOP6
2260 208F  C612      DUPL DEC MCTR        DECREMENT MCTR FOR EACH
2270 2091  D0DC           BNE LOOP6       MULTIPLE CHARACTER
2280 2093  A512      CXIT LDA MCTR        IF NO DUPLICATE THEN
2290 2095  F003           BEQ CRTN        EXIT
2300 2097  9526           STA DEST,X      STORE NEGATIVE VALUE IN
2310 2099  E8             INX             DEST IF DUPLICATE CHAR
2320 209A  8610      CRTN STX CC          NOW DEST IS IN ASM
2330 209C  A920           LDA #$20        SOURCE FORMAT
2340 209E  9526           STA DEST,X
2350 20A0  E8             INX
2360 20A1  E008           CPX #8
2370 20A3  D0F7           BNE CRTN+2
2380 20A5  ACC912         LDY STMEM       SET UP SEARCH OF SOURCE
```

(continued)

Listing 1 *(continued)*

```
20A8 ADCA12         LDA STMEM+1    CODE FOR SYMBOL
20AB 851D           STA PTR2+1
20AD A900           LDA #0
20AF 851C           STA PTR2
20B1 A200     GORD  LDX #0
20B3 CCFE12         CPY NL         IF SOURCE EXHAUSTED
20B6 D00A           BNE GORD.      AND NO MATCH FOUND
20B8 A51D           LDA PTR2+1     THEN PRINT ?
20BA CDFF12         CMP NL+1
20BD D003           BNE GORD.
20BF 4C4C21         JMP QUEST
20C2 205821   GORD. JSR INCY
20C5 8516           STA LN
20C7 205821         JSR INCY
20CA 8517           STA LN+1
20CC 205821   LS    JSR INCY
20CF 30FB           BMI LS         SKIP LEADING BLANKS
20D1 C920           CMP #$20       BOTH SINGLE AND MULT.
20D3 F0F7           BEQ LS
20D5 D003           BNE TNC
20D7 205821   NC    JSR INCY
20DA D526     TNC   CMP DEST,X     COMPARE SOURCE CODE
20DC D00A           BNE NXLN$      TO SAVED SYMBOL
20DE E8             INX
20DF E410           CPX CC
20E1 F00E           BEQ FOUND      MATCH OF CORRECT #
20E3 D0F2           BNE NC         OF CHARACTERS
20E5 205821   NXLN  JSR INCY
20E8 C90D     NXLN$ CMP #$0D
20EA F0C5           BEQ GORD
20EC 205821         JSR INCY
20EF D0F7           BNE NXLN$
20F1 205821   FOUND JSR INCY       IF FOLLOWED BY TERMINATOR
20F4 C920           CMP #$20       THEN TRUE FIND
20F6 F00C           BEQ TRFIND     ELSE BURIED IN LONGER
20F8 C90D           CMP #$0D       SYMBOL  CONTINUE SEARCH
20FA F008           BEQ TRFIND
20FC C92A           CMP #'*
20FE F004           BEQ TRFIND
2100 C93D           CMP #'=
2102 D0E4           BNE NXLN$
2104 A920     TRFIND LDA #$20      FILL BCB WITH BLANKS
2106 A205           LDX #5
2108 9535     STBL  STA BCB-1,X
210A CA             DEX
210B D0FB           BNE STBL
210D A204           LDX #4         CONVERT TO ASCII BY
210F A516           LDA LN         SUCCESSIVE DIVISIONS
2111 85CC           STA $CC        BY 10  REMAINDERS
2113 A517           LDA LN+1       ARE OR'D WITH $30
2115 85CD           STA $CD        TO GIVE ASCII CHARACTERS
2117 A90A     DVLOOP LDA #$0A      BETWEEN 0 AND 9
2119 85CE           STA $CE        END WHEN QUOTIENT = 0
211B A900           LDA #0
211D 85CF           STA $CF
211F 206E21         JSR DV10
2122 A5C8           LDA $C8
2124 0930           ORA #$30
2126 9536           STA BCB,X
2128 CA             DEX
```

(continued)

Listing 1 *(continued)*

```
2129 A5CC          LDA $CC
212B 05CD          ORA $CD
212D D0E8          BNE DVLOOP
212F A003     HRTN LDY #3
2131 A920     SB: LDA #$20
2133 204323   SB   JSR PRINT
2136 88            DEY
2137 D0FA          BNE SB
2139 A000          LDY #0
213B B93600   SN   LDA BCB,Y       PRINT LINE NUMBER
213E 204323        JSR PRINT
2141 C8            INY
2142 C005          CPY #5
2144 D0F5          BNE SN
2146 20561A   PXIT JSR CRL
2149 4C5A1F        JMP LOOP1       CONTINUE
214C A93F     QUEST LDA #'?        SYMBOL NOT FOUND IN
214E 8536          STA BCB         SOURCE   PRINT ?
2150 A900          LDA #0
2152 8537          STA BCB+1
2154 A006          LDY #6
2156 D0D9          BNE SB:
2158 B11C     INCY LDA (PTR2),Y
215A C8            INY
215B D002          BNE IXT
215D E61D          INC PTR2+1
215F 48       IXT PHA
2160 68            PLA
2161 60            RTS
2162 BD8821   DVR LDA DVS,X
2165 85CE          STA $CE
2167 E8            INX
2168 BD8821        LDA DVS,X
216B 85CF          STA $CF
216D E8            INX
216E 8614     DV10 STX XSV
2170 8415          STY YSV
2172 A204          LDX #4
2174 A900          LDA #0
2176 95C7     STRZER STA $C7,X
2178 CA            DEX
2179 D0FB          BNE STRZER
217B A210          LDX #$10
217D 20D61D        JSR DVD
2180 8A            TXA
2181 08            PHP
2182 A614          LDX XSV
2184 A415          LDY YSV
2186 28            PLP
2187 60            RTS
              ; DIVISORS FOR CODED LABELS
2188 4006     DVS .WORD 1600,40
218A 2800
              ; NON-ALPANUMERICS ALLOWED IN LABELS
218C 3A       CHR .BYTE ':.$?'
218D 2E
218E 24
218F 3F
              ;
              ; THE SUBROUTINES BELOW ARE ALREADY
```

(continued)

Listing 1 *(continued)*

```
3550          ; AVAILABLE IN THE OS65D EXTENDED MONITOR
3560          ; WHICH IS ALWAYS IN CORE WITH THE
3570          ; ASSEMBLER
3580          ; THEY ARE LISTED FOR THE CONVENIENCE
3590          ; OF USERS OF THE CASSETTE ASSEMBLER
3600          ; PHEX PHA
3610          ;      LSR A
3620          ;      LSR A
3630          ;      LSR A
3640          ;      LSR A
3650          ;      JSR PH1
3660          ;      PLA
3670          ; PH1  AND #$0F
3680          ;      ORA #$30
3690          ;      CMP #$3A
3700          ;      BCC PH2
3710          ;      ADC #6
3720          ; PH2  JMP PRINT
3730          ;
3740          ; DIVIDE ROUTINE
3750          ; DIVIDE ROL $CC
3760          ;      ROL $CD
3770          ;      DEX
3780          ;      BMI DVI
3790          ;      ROL $C8
3800          ;      ROL $C9
3810          ; DVD  SEC                  ENTRY POINT FOR DIVIDE
3820          ;      LDA $C8
3830          ;      SBC $CE
3840          ;      TAY
3850          ;      LDA $C9
3860          ;      SBC $CF
3870          ;      BCC DIVIDE
3880          ;      STA $C9
3890          ;      TYA
3900          ;      STA $C8
3910          ;      BCS DIVIDE
3920          ; DVI  LDY $CD
3930          ;      LDX $CC
3940          ;      RTS
```

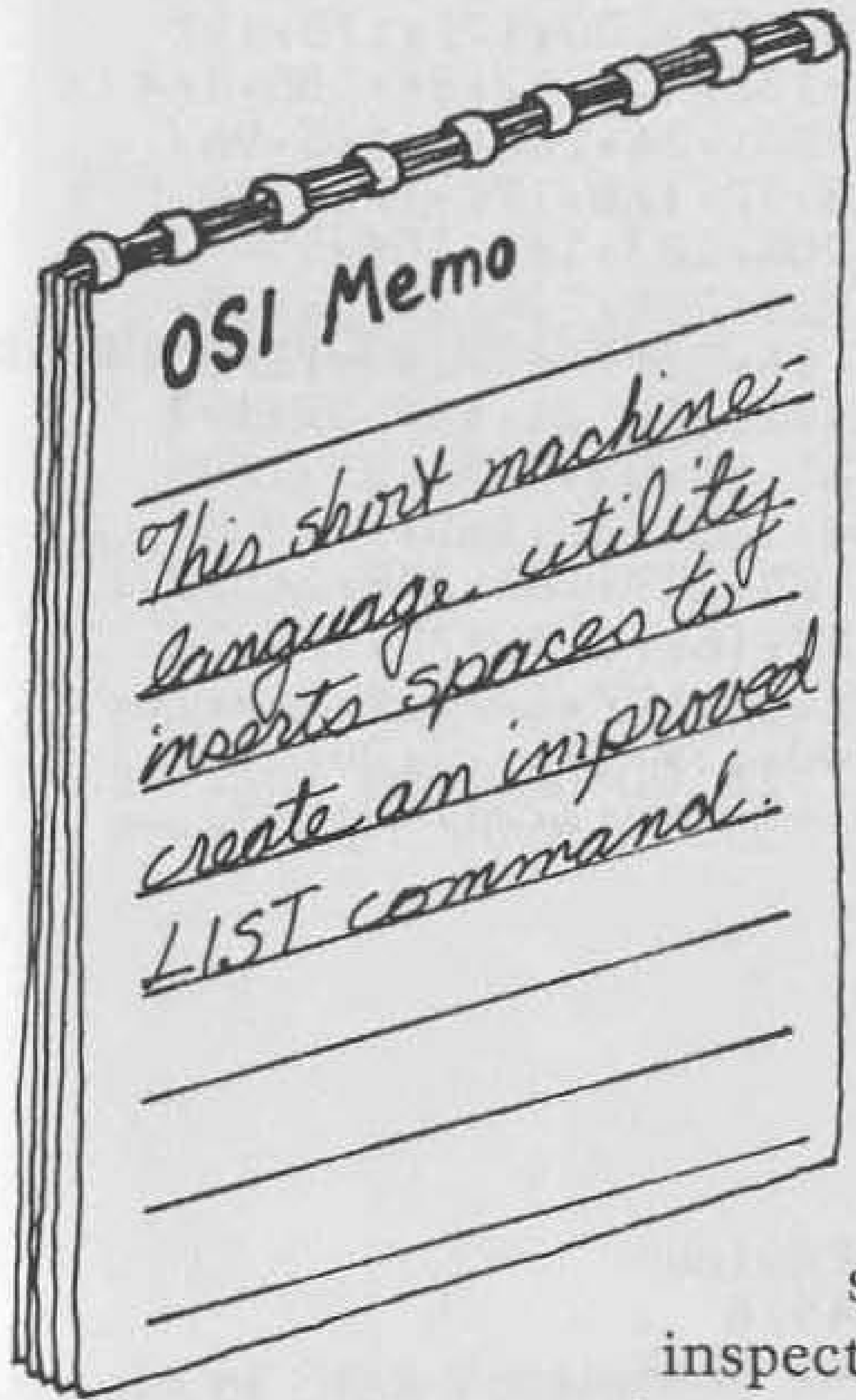
Sample Symbol Table Listing

ALPH	2016	1640
BCB	0036	240
BFR	001E	200
CC	0010	100
CHR	218C	3520
CLOOP	1F9D	950
CM1	1F99	920
CONT	1F9A	930
CPM	1FE2	1380
CRL	1A56	290
CRTN	209A	2320
CSV	0011	110
CXIT	2093	2280
DEST	0026	210
DUPL	208F	2260
DVD	1DD6	330
DVLOOP	2117	2900
DVR	2162	3280
DVS	2188	3500
DV10	216E	3340
FOUND	20F1	2710
GADR	204E	1920
GB$	2050	1930
GORD	20B1	2430
GORD.	20C2	2500
HRTN	212F	3020
INCY	2158	3210
IXT	215F	3250
LL	2F83	370
LN	0016	160
LOOP1	1F5A	550
LOOP2	1F5E	570
LOOP3	1F6A	620
LOOP3.	1FF1	1460
LOOP4	1FF5	1480
LOOP4P	1FF7	1490
LOOP5	2003	1550
LOOP6	206F	2080
LPREP	2042	1860

LS	20CC	2540
LW	0018	170
M	002E	220
MCTR	0012	120
MP	0032	230
MVMP	1FC6	1210
NC	20D7	2590
NL	12FE	280
NUM	2018	1650
NXCHR	2008	1570
NXLN	20E5	2660
NXLN$	20E8	2670
NXWORD	1FD4	1300
PHEX	19E9	310
PP	201A	1660
PRINT	2343	350
PRNT	1FD9	1330
PTR	001A	180
PTR2	001C	190
PXIT	2146	3130
QUEST	214C	3150
SB	2133	3040
SB:	2131	3030
SN	213B	3080
STB	1FDD	1350
STBL	2108	2820
STMEM	12C9	260
STOR	2087	2210
STRT	1F3E	420
STRZER	2176	3380
STS	12CB	270
TMP	1FC0	1170
TNC	20DA	2600
TRFIND	2104	2800
TRN	1F7C	720
TSR	202C	1750
TSTX	2034	1790
XP	0013	130
XSV	0014	140
YSV	0015	150

Smart Lister

by Kerry Lourash



Since OSI ROM BASIC allows only 72 characters in a line, it is often necessary to write code with no spaces between characters. This practice produces programs that are extremely difficult to read when listed. Smart Lister is a short machine-language utility that acts as an improved LIST command, inserting spaces at strategic places in the BASIC lines it lists to make the lines more legible.

I was envious when I first saw the Apple's method of program storage. Apple removes non-significant spaces from BASIC lines when they are tokenized, then adds spaces when the lines are listed. On closer inspection, however, the Apple system is not completely satisfactory. An Apple listing is too spread out for my taste; I think arithmetic operators (−, +, /, *) should *not* be segregated by spaces. Also, when two keywords are adjacent, a double space separates them. Apple doesn't check to see if the previous character was a space before printing a space. Since OSI doesn't screen out spaces on input, I wanted to include a redundant space check in my list program.

Here are the rules for Smart Lister:

1. Don't add redundant spaces.
2. Insert a space after every statement (colon).
3. Insert a space after every keyword with a token value equal to or less than the STEP token.
4. Insert a space before the TO, THEN, OR, AND, and STEP keywords.

To use the routine, simply call Lister as a USR routine and reply to the lower-case "list" prompt as you would type a LIST command. With X=USR(X) installed as line zero of a program, Lister can be called with a RUN command. Lister can be loaded in any part of memory without modification, and it occupies less than 300 bytes.

Listing 1: ROM Version of Smart Lister

```
20 PRINT"SMART LISTER": PRINT"ROM VERSION"
40 PRINT"START IS NOW AT $6000": X=24576
60 FOR I=X TO X+288: READ A: POKE I,A: NEXT
100 DATA169,108,32,229,168,169,105,32,229,168,169,115,32
110 DATA229,168,169,116,32,229,168,32,87,163,169,19,133,195
120 DATA169,0,133,196,32,194,0,144,6,240,4,201,45,208,108
130 DATA32,127,167,32,50,164,32,194,0,240,12,201,45,208,93
140 DATA32,188,0,32,127,167,208,85,165,17,5,18,208,6,169
150 DATA255,133,17,133,18,160,1,132,96,177,170,240,65,32
160 DATA41,166,32,108,168,200,177,170,170,200,177,170,197
170 DATA18,208,4,228,17,240,2,176,42,132,151,32,94,185,164
180 DATA151,169,32,32,229,168,133,19,201,34,208,6,165,96
190 DATA73,255,133,96,200,177,170,208,27,168,177,170,170
200 DATA200,177,170,134,170,133,171,208,183,162,254,154,76
210 DATA116,162,240,230,240,213,208,211,208,224,16,69,36
220 DATA96,48,203,133,20,201,157,240,16,201,160,240,12,201
230 DATA168,240,8,201,169,240,4,201,162,208,11,169,32,197
240 DATA19,240,5,133,19,32,229,168,165,20,56,233,127,170
250 DATA132,151,160,255,202,240,8,200,185,132,160,16,250
260 DATA48,245,200,185,132,160,48,28,32,229,168,208,245,201
270 DATA58,208,134,36,96,48,130,32,229,168,200,177,170,201
280 DATA32,240,161,136,169,32,208,158,41,127,32,229,168,164
290 DATA151,200,177,170,136,201,32,240,139,165,20,201,163
300 DATA144,231,201,168,240,227,201,169,240,223,208,129
310 PRINT"**LOADED**"
```

Listing 2: Disk Version of Smart Lister

```
10 PRINT"SMART LISTER": PRINT"DISK VERSION"
30 PRINT"START IS NOW AT $6000": X=24576
50 FOR I=X TO X+291: READ A: POKE I,A: NEXT
90 DATA32,247,44
100 DATA169,108,32,238,10,169,105,32,238,10,169,115,32,238
110 DATA10,169,116,32,238,10,32,88,5,169,27,133,199,169,0
120 DATA133,200,32,198,0,144,6,240,4,201,45,208,108,32,108
130 DATA9,32,51,6,32,198,0,240,12,201,45,208,93,32,192,0
140 DATA32,108,9,208,85,165,25,5,26,208,6,169,255,133,25
150 DATA133,26,160,1,132,29,177,172,240,65,32,25,8,32,115
160 DATA10,200,177,172,170,200,177,172,197,26,208,4,228,25
170 DATA240,2,176,42,132,150,32,220,28,164,150,169,32,32
180 DATA238,10,133,27,201,34,208,6,165,29,73,255,133,29,200
190 DATA177,172,208,27,168,177,172,170,200,177,172,134,172
200 DATA133,173,208,183,162,254,154,76,116,4,240,230,240
210 DATA213,208,211,208,224,16,69,36,29,48,203,133,28,201
220 DATA157,240,16,201,160,240,12,201,168,240,8,201,169,240
230 DATA4,201,162,208,11,169,32,197,27,240,5,133,27,32,238
240 DATA10,165,28,56,233,127,170,132,150,160,255,202,240
250 DATA8,200,185,132,2,16,250,48,245,200,185,132,2,48,28
260 DATA32,238,10,208,245,201,58,208,134,36,29,48,130,32
270 DATA238,10,200,177,172,201,32,240,161,136,169,32,208
280 DATA158,41,127,32,238,10,164,150,200,177,172,136,201
290 DATA32,240,139,165,28,201,163,144,231,201,168,240,227
300 DATA201,169,240,223,208,129
310 PRINT"**LOADED**"
```

Sample of Normal Listing

```
LIST

 10 FORX=1TO10:A(X)=1:NEXTX
 20 IFA>2THENGOSUB99
 30 POKEA,B:POKEA+1,C
```

Sample of ROM Version

```
Z=USR(8)
list10-30

 10 FOR X=1 TO 10: A(X)=1: NEXT X
 20 IF A>2 THEN GOSUB 99
 30 POKE A,B: POKE A+1,C
```

Sample of Disk Version

```
DISK!"GO 6000"
list1-

 10 FOR X=1 TO 10: A(X)=1: NEXT X
 20 IF A>2 THEN GOSUB 99
 30 POKE A,B: POKE A+1,C
```

Listing of [illegible] Version of Binary Listing

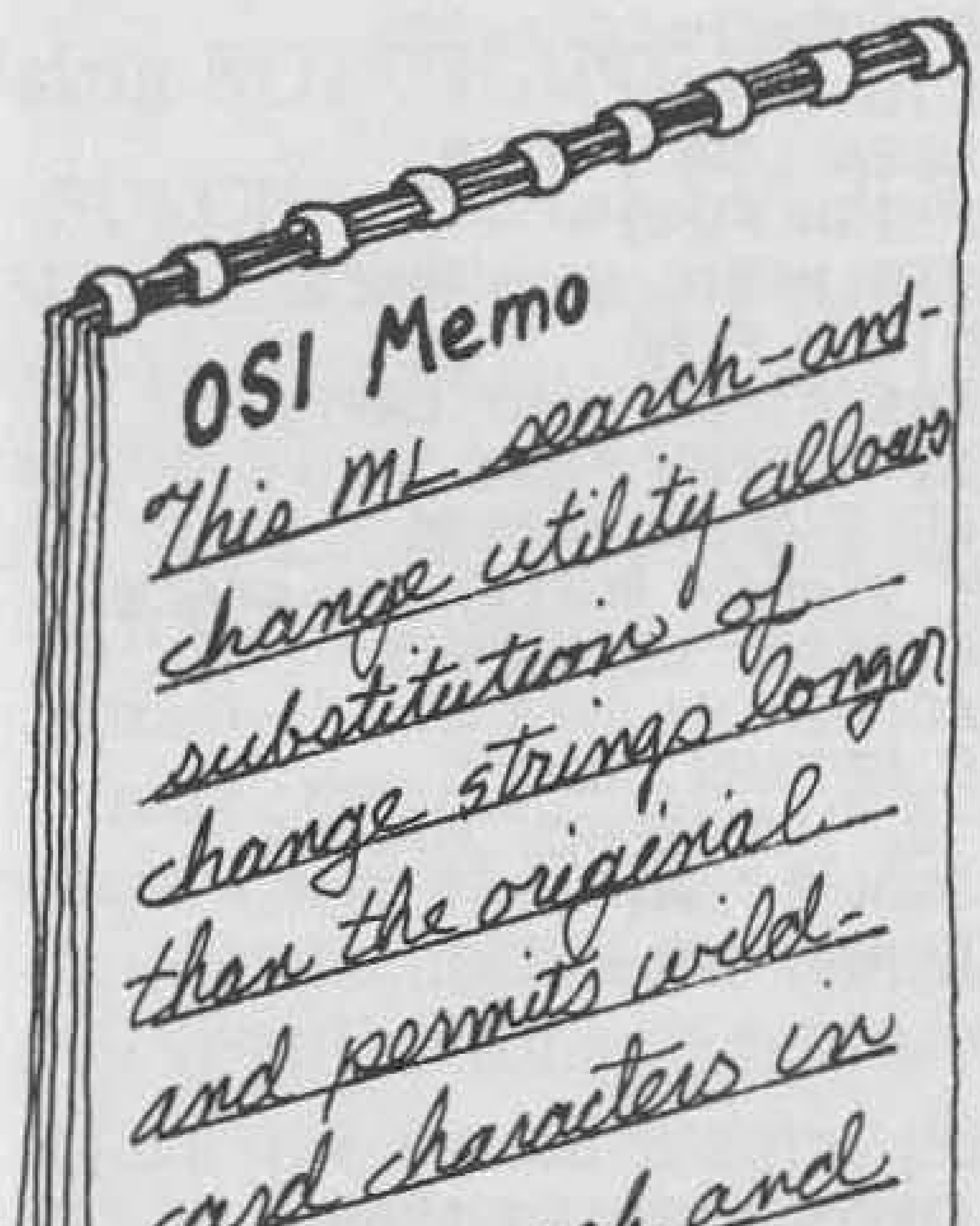
```
[illegible]
```

[illegible] of Disk Version of Binary Listing

```
[illegible]
```

SURCHANGE

by Kerry Lourash



SURCHANGE searches for, displays, and changes code in BASIC programs. As many as seventy-one characters may be searched for and changed. Don't-care characters are allowed in both search and change strings. The user may specify change strings shorter, equal to, or longer than the search string. To avoid confusion, here are the definitions of some terms used in this article: *search string* refers to the characters for which SURCHANGE is told to look; *workspace string* is a set of characters in the BASIC program that matches the search string; and *change string* is the set of characters that SURCHANGE POKEs into the BASIC program when it finds a match to the search string.

There are eight options, used singly or in pairs:

Default	Print line numbers of lines that contain workspace strings.
1. Print	Print line numbers plus workspace strings.
2. Stmt	Print line numbers plus the statements in which workspace strings are found.
3. Line	Print lines in which workspace strings are found.
4. Quote	Search only within quotes and REM statements (text).
Default	If option 4 is not chosen, search only outside of quotes and REMs (program).
5. Var	Search for occurrences of a BASIC variable (specified by the search string).
6. Change	Replace all workspace strings with the change string.

Don't-care characters are allowed in both search and change strings. To illustrate what a don't-care character is, consider the following example:

```
SEARCH? YXXX
```

Listing 1

```
0000               ; SURCHANGE
0000               ;BY KERRY LOURASH
0000               ;
0000               ; ZERO PAGE
0000               ;
0000                  BUF=$97      TEMP STORAGE FOR SEARCH CHAR.
0000                 BUFF=$13      START OF INPUT BUFFER
0000                CFLAG=$98      CHANGE OPTION FLAG
0000               CHRCNT=$6B      # OF CHARS. IN CURRENT LINE
0000                 CLEN=$6C      LENGTH OF CHANGE STRING
0000                  DIF=$5D      CLEN MINUS SLEN
0000               ORIGIN=$9A      START OF WORKSPACE INDEX
0000                LFLAG=$9B      LINE OPTION FLAG
0000               LINCNT=$5E      # OF SCREEN LINES USED
0000                PFLAG=$6D      PRINT OPTION FLAG
0000                POINT=$6E      POINTER TO BASIC WORKSPACE
0000                QFLAG=$9C      QUOTE OPTION FLAG
0000               SCNCNT=$E       # OF CHARS SINCE LAST CR/LF
0000                SFLAG=$9E      STATEMENT OPTION FLAG
0000                 SLEN=$99      LENGTH OF SEARCH STRING
0000                 STAK=$9D      START OF SEARCH BUFFER IN STACK
0000                START=$79      START OF BASIC WORKSPACE
0000                 TEXT=$60      TEXT FLAG
0000                VFLAG=$70      VARIABLE OPTION FLAG
0000               WPOINT=$AA      ADDRESS OF WORKSPACE STRING
0000               YINDEX=$9F      TEMP STORAGE FOR POINT INDEX
0000                YSAVE=$97      TEMP STORAGE FOR PRINT INDEX
0000               ;
0000               ; BROM ROUTINES
0000               ;
0000               DELETE=$014F    DELETE CHARS FROM PROGRAM
0000                 BROM=$A2B4    BROM ROUTINES COPIED FROM STACK
0000                CHAIN=DELETE+$5A  RECHAIN BASIC LINES
0000               FILBUF=$A946    FILL BUFFER ROUTINE
0000               INCHAR=$FFEB    INPUT ONE CHAR FROM KYBD.
0000               LETTER=$AD81    CHECK FOR LETTERS A-Z
0000               LIFEED=$A86C    PRINT CR/LF
0000               NUMBER=$00C5    CHECK FOR NUMBER 0-9
0000               NUMPRT=$B95E    PRINT NUMBER IN A,X
0000               OUTPUT=$A8E5    PRINT ONE CHARACTER
0000               PUSHUP=DELETE+$35  MAKE ROOM FOR LINE
0000               QUESTN=$A8E3    PRINT A QUESTION MARK
0000                RESET=$A477    RESET BASIC POINTERS
0000                SPACE=$A8E0    PRINT A SPACE
0000               TOGOUT=$A39D    TOGGLE VIDEO OUTPUT FLAG
0000               TOKBUF=$A3A8    TOKENIZE LINE BUFFER
0000               TOKTBL=$A084    START OF TOKEN TABLE
0000               WARMST=$A274    ENTRY TO BASIC WARMSTART
0000               ;
7D00               *=$7D00
7D00               ;
7D00 A200          OPTION LDX #0        SET PROMPT INDEX
7D02 20947F               JSR PROMPT    PRINT OPTION PROMPT
7D05 8598                 STA CFLAG     ZERO FLAGS
7D07 859B                 STA LFLAG
7D09 856D                 STA PFLAG
7D0B 859C                 STA QFLAG
7D0D 859E                 STA SFLAG
```

(continued)

I'm using "X" for the don't-care symbol; in the actual program it is CTRL-G, the ASCII BEL character. This search string finds all strings of four characters starting with a "Y". For an example of don't-care characters in a change string:

CHANGE? YXXX

This change string changes only the first letter of the workspace string. The last three letters remain the same.

Using SURCHANGE

SURCHANGE can be called by POKEing its starting address into the USR vector and typing X=USR(X). To avoid typing the USR command every time, you could insert the USR command as line zero in the program on which you are working. Typing RUN then calls SURCHANGE. First, SURCHANGE prints a list of options and a prompt to select options (OPTIONS?). Options are selected by typing a combination of digits (no commas). If you make a mistake, use the usual OSI backspace (shift O). You may terminate the line and start over with a shift P, although the prompt will not be repeated. RETURN signals the end of option selection. If this procedure seems familiar, it should; you're using the Fill-the-Buffer (FTB) routine of OSI BASIC.

Next, the search prompt (SEARCH?) is printed. The FTB routine is used here, too. Don't-care characters are input by typing CTRL-G. If you hit RETURN without an input when typing the search or change string, SURCHANGE prints the exit prompt. If you type a "Y", SURCHANGE exits to the immediate mode. Hitting any other key causes a jump to the start of SURCHANGE.

The change prompt (CHANGE?) appears if you've chosen the change option. Only the line numbers of changes will be printed when the change option is selected. If a line is made too long (longer than 71 characters), the graphics symbol $E9 is printed after the line number.

I have attempted to provide a paged display of SURCHANGE's output. It would be nice to be able to count the number of CR/LFs generated by the video routine to determine when the screen is full. So far, I haven't figured out how to accomplish this, short of writing a separate video routine. After a certain number of lines have been printed, SURCHANGE pauses. If the space bar is hit, the display continues. Any other key causes an exit to the immediate mode without an "OK" to scroll the screen. If you use the line-print option (3), you can display lines and edit them (assuming you have an editor program).

Options

Default options are automatically selected if options 1-3 or option 4 is not selected. When the change option is chosen, SURCHANGE

Listing 1 *(continued)*

```
7D0F 8570            STA VFLAG
7D11 855E            STA LINCNT
7D13 2046A9          JSR FILBUF    GET CHOICE OF OPTIONS
7D16 E8       OP     INX           AFTER FILBUF, X=$12
7D17 B500            LDA $0,X      EXAMINE BUFFER CONTENTS
7D19 F023            BEQ LOGIC     BRANCH IF DONE
7D1B 38              SEC           CONVERT ASCII TO NUMBER
7D1C E931            SBC #$31
7D1E A8              TAY           NUMBER TO Y REG.
7D1F D002            BNE OP1       SET CORRECT FLAG
7D21 C66D            DEC PFLAG
7D23 88       OP1    DEY
7D24 D002            BNE OP2
7D26 C69E            DEC SFLAG
7D28 88       OP2    DEY
7D29 D002            BNE OP3
7D2B C69B            DEC LFLAG
7D2D 88       OP3    DEY
7D2E D002            BNE OP4
7D30 C69C            DEC QFLAG
7D32 88       OP4    DEY
7D33 D002            BNE OP5
7D35 C670            DEC VFLAG
7D37 88       OP5    DEY
7D38 D0DC            BNE OP
7D3A C698            DEC CFLAG
7D3C D0D8            BNE OP        BRANCH ALWAYS
7D3E          ;
7D3E A698     LOGIC  LDX CFLAG     IS CHANGE FLAG SET?
7D40 F006            BEQ L1
7D42 859B            STA LFLAG     FORCE DEFAULT OPTION
7D44 856D            STA PFLAG
7D46 859E            STA SFLAG
7D48 A570     L1     LDA VFLAG     BOTH V & Q FLAGS SET?
7D4A 259C            AND QFLAG
7D4C F003            BEQ GETSUR
7D4E 20E3A8          JSR QUESTN    PRINT A QUESTION MARK
7D51          ;
7D51 A24C     GETSUR LDX #$4C
7D53 20947F          JSR PROMPT    PRINT SEARCH PROMPT
7D56 202E7F          JSR INPUT     GET SEARCH STRING
7D59 A24E     STACK  LDX #$4E      SET STACK PTR TO $014E
7D5B 9A              TXS
7D5C AA              TAX           SLEN TO X REG.
7D5D 8699            STX SLEN
7D5F E8              INX
7D60 B513     ST     LDA BUFF,X    PUSH SEARCH STRING
7D62 48              PHA           ONTO STACK
7D63 CA              DEX
7D64 10FA            BPL ST
7D66 BA              TSX           START OF SEARCH STRING
7D67 869D            STX STAK      TO STAK
7D69 A2FE            LDX #$FE      RESET STACK
7D6B 9A              TXS
7D6C          ;
7D6C A598            LDA CFLAG
7D6E F029            BEQ SEARCH
7D70 A253     GETCNG LDX #$53
7D72 20947F          JSR PROMPT    PRINT CHANGE PROMPT
7D75 202E7F          JSR INPUT     PRINT & STORE CHANGE STRING
```

(continued)

automatically selects the default display option. If options 4 and 5 are both selected, SURCHANGE prints a question mark in front of the search prompt, since it is unlikely the user would look for a variable in the text area of a program. The default display option displays the line numbers of lines that contain workspace strings. The numbers are displayed with a single space separating them. If a number is printed more than once, more than one workspace string is present in the line. This option allows a very dense display and calls attention to multiple occurrences of a workspace string in a line.

Option 1 displays line numbers plus the workspace string. Due to the presence of don't-care characters in a search string, the workspace string may not be identical to the search string. This option is handy when don't-care characters are used. Also, option 1 emphasizes multiple occurrences of workspace strings in a line, although its display format is not as compact as the default option's.

The statement option (2) prints the line number and the statement in which the workspace string is found (a line may contain multiple statements). Colons found at the beginning and end of the statement are also printed. The presence or absence of colons indicates the statement's position in the line.

 X = 3:—statement at start of line
:X = 3:—statement in middle of line
:X = 3 —statement at end of line
 X = 3 —statement is the entire line

Option 2 allows the user to follow the use of a variable throughout a program or to examine all occurrences of any token (and its arguments) in a program. A statement is printed only once, even if it contains more than one workspace string. For example, in the statement A = A − 3 the variable A occurs twice. If "A" were the search string, the statement would be printed only once.

The line option (3) lets the user see the entire line that contains the workspace string. This option displays a maximum amount of information but also fills the screen rapidly. Like the statement option, the line option prints a line only once, even if it contains more than one workspace string. The line option can be used as an aid to edit individual lines. With SURCHANGE, find the lines to be edited, exit the SURCHANGE program, and either use an editor to change the lines or retype them.

The quote option (4) searches the text portion of a BASIC program. Text includes PRINT statements, INPUT prompts, string variables, string DATA elements, and REM statements. Due to the structure of SURCHANGE, the initial quotation mark of a string is not considered to be part of the text. If the quote option is not chosen, SURCHANGE searches the program area outside of quotes and REMs. The reason for defining two areas of search is that BASIC tokenizes its keywords (USR,

Listing 1 *(continued)*

```
7D78 856C                STA CLEN
7D7A                ;
7D7A A296                LDX #$96        MOVE BROM ROUTINES
7D7C A089                LDY #$89        TO STACK
7D7E BDB4A2    COPY      LDA BROM,X
7D81 994E01              STA DELETE-1,Y
7D84 CA                  DEX
7D85 E067                CPX #$67
7D87 D002                BNE CP
7D89 A25A                LDX #$5A
7D8B 88        CP        DEY
7D8C D0F0                BNE COPY
7D8E A960                LDA #$60        INSERT RTS INSTRUCTIONS
7D90 8D8001              STA DELETE+$31
7D93 8DA801              STA DELETE+$59
7D96 8DB801              STA DELETE+$69
7D99                ;
7D99 A579      SEARCH    LDA START       BASIC WORKSPACE POINTER
7D9B 856E                STA POINT       STOREDIN POINT, POINT+1
7D9D A57A                LDA START+1
7D9F 856F                STA POINT+1
7DA1 A003      NEXLIN    LDY #3          SKIP LINE POINTERS
7DA3 849A                STY ORIGIN
7DA5 A900                LDA #0          INITIALIZE TEXT FLAG
7DA7 8560                STA TEXT
7DA9 E69A      SETBUF    INC ORIGIN
7DAB A49A                LDY ORIGIN
7DAD A69D                LDX STAK        SET STACK POINTER TO
7DAF 9A                  TXS             START OF SEARCH BUFFER
7DB0 68        NEXBUF    PLA             GET SEARCH CHAR.
7DB1 F04D                BEQ MATCH       FOUND A MATCH?
7DB3 C907                CMP #7          DON'T CARE CHAR?
7DB5 D002                BNE STOBUF
7DB7 B16E                LDA (POINT),Y
7DB9 8597      STOBUF    STA BUF         SAVE CHAR. IN BUF
7DBB B16E      NEXBYT    LDA (POINT),Y
7DBD AA                  TAX
7DBE F01E                BEQ FIXLIN      END OF BASIC LINE?
7DC0 E08E      REM       CPX #$8E        REM TOKEN?
7DC2 F011                BEQ TOGGLE      YES, TOGGLE TEXT FLAG
7DC4 E022      QUOTE     CPX #'"
7DC6 F00D                BEQ TOGGLE
7DC8 A59C      CKTEXT    LDA QFLAG       CHECK TEXT FLAG
7DCA C560                CMP TEXT
7DCC D0DB                BNE SETBUF
7DCE E497      COMPAR    CPX BUF         DO CHARS MATCH?
7DD0 D0D7                BNE SETBUF
7DD2 C8                  INY             INCREMENT WORKSPACE INDEX
7DD3 D0DB                BNE NEXBUF      BRANCH ALWAYS
7DD5 A560      TOGGLE    LDA TEXT        TOGGLE TEXT FLAG
7DD7 49FF                EOR #$FF
7DD9 8560                STA TEXT
7DDB 4CCE7D              JMP COMPAR
7DDE                ;
7DDE A8        FIXLIN    TAY             SET POINT TO NEXT LINE
7DDF B16E                LDA (POINT),Y
7DE1 AA                  TAX
7DE2 C8                  INY
7DE3 B16E                LDA (POINT),Y
7DE5 866E                STX POINT
```

(continued)

POKE, NULL, etc.), unless the words are in REMs or quotes. A token is a one-byte code for a keyword. BASIC saves memory space and increases execution speed because it stores and reads only one byte instead of a whole keyword. Thus, if you're searching for "ON", SURCHANGE needs to know whether you mean the word "ON" or the one-byte token for the keyword ON.

The variable option (5) helps search for a BASIC variable. In a normal search, looking for the variable "A" might find other variables such as A$, AB, A(X), etc. When the variable option is chosen, every variable found is tested to be sure it's not a subset of another variable.

The change option (6) enables modification of a BASIC program. Change strings may be shorter, equal in length, or longer than the search string. This is a powerful option and should always be used with caution. Unless changing text, SURCHANGE will tokenize the change string before it is inserted in the program. Therefore, the change string may look deceptively longer or shorter than the search string when it is printed on the screen. For example, "RETURN" is one byte long when tokenized, while "A=6" is three bytes long. If "A=6" is substituted for RETURN, all lines changed will be two bytes longer. If a line is longer than 71 bytes, it can still be LISTed, SAVEd, and even RUN. When you try to LOAD a long line, however, you'll find that the line is too long to fit into the input buffer. SURCHANGE prints a graphic character $E9 after a line number when the line becomes too long. Be sure to remember which lines are too long; they are identified only when the line is being changed, not during search operations.

Finding Your Way Around

SURCHANGE takes getting used to. I suggest you type in a ten- to twenty-line program and practice finding and changing things before you do any serious work. Here are a few tricks I use. To delete all non-text spaces in a program, select option 6. Type a space and a don't-care character for the search string. Now, type a single don't-care character for the change string. This gets rid of almost all single spaces and partially erases multiple spaces. Repeat as needed to erase all spaces. This strategy may work with other items you wish to delete.

When typing in a program, use a "%" or other seldom-used character to stand in for a phrase, which is inserted by SURCHANGE after the program is completed. Of course, you must be careful not to make a line too long by the insertion. Lines of up to 255 characters can be created with the change option. They use less memory space and run faster than normal lines. The big disadvantage of long lines is that they have to be saved and loaded in a machine-language format.

Listing 1 *(continued)*

```
7DE7 856F              STA POINT+1
7DE9 D0B6              BNE NEXLIN    END OF PROGRAM?
7DEB          ;
7DEB A25A     END      LDX #$5A
7DED 20947F            JSR PROMPT    PRINT EXIT PROMPT
7DF0 20EBFF            JSR INCHAR    GET CHAR. FROM KYBD.
7DF3 C959              CMP #'Y
7DF5 F003              BEQ DONE
7DF7 4C007D            JMP OPTION    LOOP TO START OF SURCHANGE
7DFA 4C74A2   DONE     JMP WARMST    GOTO IMMEDIATE MODE
7DFD 4C1C7F   RET      JMP RETURN
7E00          ;
7E00 88       MATCH    DEY           SAVE WORKSPACE INDEX
7E01 849F              STY YINDEX
7E03 A2FE              LDX #$FE      RESET STACK
7E05 9A                TXS
7E06          ;
7E06 A570     VARIBL   LDA VFLAG     TEST VARIABLE FOUND
7E08 F01C              BEQ LINE
7E0A A49A              LDY ORIGIN    INDEX TO START OF STRING
7E0C C004              CPY #4        FIRST CHAR. IN LINE?
7E0E F006              BEQ V0
7E10 88                DEY           GET PREVIOUS CHARACTER
7E11 B16E              LDA (POINT),Y
7E13 20237F            JSR LEGAL     IS IT A ALPHANUMERIC CHAR?
7E16 A49F     V0       LDY YINDEX    GET CHAR. IN FRONT OF STRING
7E18 C8       V1       INY
7E19 B16E              LDA (POINT),Y
7E1B C924              CMP #'$
7E1D F0DE              BEQ RET
7E1F C928              CMP #'(
7E21 F0DA              BEQ RET
7E23 20237F            JSR LEGAL
7E26          ;
7E26 A002     LINE     LDY #2        GET 2-BYTE LINE #
7E28 B16E              LDA (POINT),Y
7E2A AA                TAX
7E2B C8                INY
7E2C B16E              LDA (POINT),Y
7E2E 205EB9            JSR NUMPRT    CONVERT TO ASCII, PRINT
7E31 E8       LIN      INX           PUT # OF DIGITS IN CHRCNT
7E32 BD0001            LDA $0100,X
7E35 D0FA              BNE LIN
7E37 866B              STX CHRCNT
7E39          ;
7E39 A56D     PCHECK   LDA PFLAG
7E3B D044              BNE FINI
7E3D A59E     SCHECK   LDA SFLAG
7E3F F02E              BEQ LCHECK
7E41 A49F              LDY YINDEX    FIND END OF LINE
7E43 B16E     S0       LDA (POINT),Y  OR TERMINATING COLON
7E45 F013              BEQ S2
7E47 C8                INY
7E48 C922              CMP #'"
7E4A D006              BNE S1
7E4C A560              LDA TEXT      TOGGLE TEXT FLAG
7E4E 49FF              EOR #$FF      IF QUOTE IS FOUND
7E50 8560              STA TEXT
7E52 2460     S1       BIT TEXT      LOOP IF IN TEXT
7E54 30ED              BMI S0
```

(continued)

Changing SURCHANGE

C2/4P owners should change the COUNTR routine, as noted in the listing. They may also want to eliminate the CR/LF between the two lines of options in the option prompt. The easiest method is to substitute two spaces ($20) for the $D, $A after "3-LINE" in TABL at the end of the program. If you wish to examine the BASIC-in-ROM routines copied to the stack, or if you must move them to another location, simply change the DELETE label to the start of the new location. SURCHANGE is relocatable from object code with the exception of references to the prompt table (TABL). All references to TABL should be adjusted to conform to its new location.

Two more changes may be made: the graphic character (#$E9) in line 337 (TOOLING) may be changed to an asterisk (#$2A) for compatibility with printers; and the output pager may be disabled by deleting lines 279-280 or replacing the code with NOPs (#$EA).

How SURCHANGE Works

SURCHANGE occupies three pages of RAM and uses part of the stack for BASIC-in-ROM routines and the search buffer. It wipes out the NMI and IRQ vectors. To conserve zero-page space for other accessory programs, SURCHANGE uses only zero-page addresses normally used by BASIC. The change buffer is located in the line buffer ($13-5A).

To start, OPTION prints a list of options and the option prompt. The option flags are zeroed and FILBUF is called to find out what options are wanted. When the options have been specified, their respective flags are set. LOGIC selects the default-print option if the change flag is set, and prints a question mark in front of the search prompt if both the variable and quote flags are set. GETSUR prints the search prompt and calls INPUT. INPUT zeros the video character counter ($E) so a full 71-character line can be typed without a premature CR/LF. FILBUF is called again to store and print the search string. After the search string is typed in, the number of characters in the string is counted. If no string has been input, the routine goes to END to see if the user wishes to start over. If the search is to be conducted within quotes, the tokenize-the-buffer routine (TOKBUF) is skipped. The number of characters in the string is returned in the A register. INPUT returns to STACK, where the stack pointer is set to $014E and the length of the search string is stored in SLEN. The search string is pushed onto the stack and the stack pointer position saved in STAK. The stack pointer is then reset to the top of the stack.

If the change option has been selected, GETCNG prints the change prompt and INPUT is called to get the change string. When INPUT returns, the length of the change string is stored in CLEN. COPY transfers BASIC-in-ROM routines for inserting, deleting, and rechaining BASIC lines to the stack, and inserts RTS instructions to make them subroutines.

Listing 1 *(continued)*

```
7E56 C93A            CMP #':
7E58 D0E9            BNE S0
7E5A 88      S2      DEY
7E5B 849F            STY YINDEX   SAVE NEW END OF STRING
7E5D A49A            LDY ORIGIN   LOOK BACK THRU LINE
7E5F B16E    BACKWD  LDA (POINT),Y
7E61 88              DEY
7E62 C93A            CMP #':
7E64 F004            BEQ BA
7E66 C003            CPY #3       AT START OF LINE?
7E68 D0F5            BNE BACKWD
7E6A C8      BA      INY
7E6B 849A            STY ORIGIN   SAVE NEW START OF LINE
7E6D D012            BNE FINI
7E6F A59B    LCHECK  LDA LFLAG
7E71 F03E            BEQ CHANGE
7E73 A49F            LDY YINDEX   FIND END OF LINE
7E75 C8      LC      INY
7E76 B16E            LDA (POINT),Y
7E78 D0FB            BNE LC
7E7A 88              DEY
7E7B 849F            STY YINDEX   SAVE END OF LINE
7E7D A004            LDY #4
7E7F 849A            STY ORIGIN   START OF LINE IS ALWAYS 4
7E81 20E0A8  FINI    JSR SPACE    PRINT SPACE
7E84 204E7F          JSR PLINE    PRINT LINE
7E87         ;
7E87 E65E    COUNTR  INC LINCNT   CHECK # OF CHARS. IN LINE
7E89 A56B            LDA CHRCNT   AND INCREMENT COUNT AS NEEDED
7E8B C917            CMP #$17   ** C2P: CHANGE TO #$3F **
7E8D 9008            BCC CHEC
7E8F C92F            CMP #$2F   ** C2P: CHANGE TO #$7F **
7E91 9002            BCC ADD1
7E93 E65E            INC LINCNT
7E95 E65E    ADD1    INC LINCNT
7E97 A55E    CHEC    LDA LINCNT
7E99 C90F            CMP #$F      COUNT <= 15 LINES?
7E9B 900E            BCC CONT
7E9D A900            LDA #0
7E9F 855E            STA LINCNT
7EA1 20EBFF          JSR INCHAR   GET KYBD. INPUT
7EA4 C920            CMP #$20     IS INPUT A SPACE CHAR?
7EA6 F003            BEQ CONT
7EA8 4C7DA2          JMP $A27D    GOTO IMM. MODE; NO OK MESS.
7EAB 206CA8  CONT    JSR LIFEED   PRINT CR/LF
7EAE 4C1C7F          JMP RETURN   RESUME SEARCH
7EB1         ;
7EB1 A598    CHANGE  LDA CFLAG
7EB3 F067            BEQ RETURN
7EB5 18              CLC          CALCULATE ABSOLUTE ADDRESS
7EB6 A59A            LDA ORIGIN   OF START OF WORKSPACE STRING
7EB8 656E            ADC POINT
7EBA 85AA            STA WPOINT
7EBC A46F            LDY POINT+1
7EBE 9001            BCC CH
7EC0 C8              INY
7EC1 84AB    CH      STY WPOINT+1
7EC3 38              SEC
7EC4 A56C            LDA CLEN     FIND CLEN MINUS SLEN
7EC6 E599            SBC SLEN
```

(continued)

The start-of-BASIC workspace pointer is transferred to SURCHANGE's workspace pointer (POINT). NEXLIN sets the Y register to index the start of the BASIC line, and TEXT, the quote status flag, is cleared. ORIGIN is initialized to the start of the line, the stack pointer is set to the start of the search buffer, and a character is pulled from the stack. Naturally, the contents of the stack are not altered by this operation, and SURCHANGE can re-examine the search buffer any number of times. If the character is a null, SURCHANGE has found a match to the search string and goes to the MATCH routine. If it is a don't-care character, the next character in the BASIC workspace is stored in BUF. Later, when the workspace character is compared to BUF, the two will match. If the search character is not a null or don't-care byte, it's stored in BUF.

NEXBYT tests the next character in the workspace. If the workspace character is a null, the end of the BASIC line has been reached. The routine branches to FIXLIN to reset POINT to the next line or to exit, if at the end of the program. If the workspace character is a REM token or a quotation mark, the TEXT flag is toggled. This means if TEXT is zero, it is changed to #$FF, and *vice versa*. If TEXT is not equal to the quote option flag, SURCHANGE loops back to SETBUF. Finally, at COMPAR, the search character is compared to the workspace character. If the two are identical, the next search character is pulled from the stack and the NEXBUF loop is done again. If the characters don't match, the stack pointer is reset to the start of the search buffer, the workspace counter (ORIGIN) is incremented, and SURCHANGE starts looking for a workspace string again.

FIXLIN, as mentioned before, transfers the BASIC next-line pointer to POINT. If the high byte of the pointer is zero, the end of the BASIC program has been reached. The stack pointer is set to the top of the stack, "EXIT?" is printed, and SURCHANGE waits for an input. At this point, the user can hit Y and exit to the BASIC immediate mode or hit any other key to rerun SURCHANGE.

If a match to the search string is found, the workspace index (Y) to POINT is stored in YINDEX. The stack pointer is set to the top of the stack. If VFLAG is set, VARIBL tests the characters adjacent to the workspace string to see if the string is a subset of another variable. If the correct variable has not been found, LEGAL jumps back into the search loop. LINE finds the current line number in the workspace and prints it. It also counts the number of digits in the line number for later use in the COUNTR or LONG routines. PCHECK prints a space and the workspace string if the print flag is set.

SCHECK finds the terminating colon of the statement or the end of the line. BACKWD finds the start of the statement or the start of the line. (I was strapped for space here so I didn't include a check in BACKWD to be sure a colon is really a statement separator and not part of a string.) LCHECK finds the start and end of the line. The start is easy — always

Listing 1 *(continued)*

```
7EC8 F02E          BEQ CEQUAL    IF CLEN = SLEN
7ECA 900C          BCC MOVDWN
7ECC 855D   MOVEUP STA DIF       MAKE ROOM FOR LONGER STRING
7ECE C65D          DEC DIF
7ED0 208401        JSR PUSHUP
7ED3 20857F        JSR REPLAC    INSERT CHANGE STRING
7ED6 301A          BMI MV3
7ED8 A47B   MOVDWN LDY $7B  SET UP VARIABLES FOR DELETESUB
7EDA 8471          STY $71
7EDC A4AB          LDY WPOINT+1
7EDE 8474          STY $74
7EE0 48            PHA
7EE1 38            SEC
7EE2 A599          LDA SLEN
7EE4 E56C          SBC CLEN
7EE6 18            CLC
7EE7 65AA          ADC WPOINT
7EE9 9001          BCC MV2
7EEB C8            INY
7EEC 8472   MV2    STY $72
7EEE 68            PLA
7EEF 204F01        JSR DELETE    ERASE XTRA CHARS FROM PROGRAM
7EF2 2077A4 MV3    JSR RESET     RESET BASIC POINTERS
7EF5 20A901        JSR CHAIN     RECHAIN LINE POINTERS
7EF8 20857F CEQUAL JSR REPLAC
7EFB 209DA3 LONG   JSR TOGOUT    CHECK FOR LONG LINE
7EFE A0FF          LDY #$FF
7F00 849F          STY YINDEX
7F02 A004          LDY #4
7F04 20507F        JSR PLINE+2
7F07 209DA3        JSR TOGOUT
7F0A 18            CLC
7F0B A56C          LDA CLEN      FIND NEW END OF STRING
7F0D 659A          ADC ORIGIN
7F0F 859F          STA YINDEX
7F11 A56B          LDA CHRCNT    IS LINE TOO LONG?
7F13 C947          CMP #$47
7F15 9005          BCC RETURN
7F17 A9E9   TOOLNG LDA #$E9      PRINT GRAPHIC CHAR.
7F19 20E5A8        JSR OUTPUT
7F1C A49F   RETURN LDY YINDEX
7F1E 849A          STY ORIGIN
7F20 4CA97D        JMP SETBUF    RESUME SEARCH
7F23        ;
7F23 20C500 LEGAL  JSR NUMBER    IS CHAR =0-9?
7F26 90F4          BCC RETURN
7F28 2081AD        JSR LETTER    IS CHAR =A-Z?
7F2B B0EF          BCS RETURN
7F2D 60            RTS
7F2E        ;
7F2E 850E   INPUT  STA SCNCNT    ZERO VIDEO CHAR COUNTER
7F30 2046A9        JSR FILBUF    PRINT AND STORE INPUT
7F33 88            DEY           Y=#$FF
7F34 C8     LILOOK INY           COUNT # OF CHARS. IN INPUT
7F35 B91300        LDA BUFF,Y
7F38 D0FA          BNE LILOOK
7F3A 88     TOKIZE DEY
7F3B 1003          BPL TKO
7F3D 4CEB7D        JMP END       IF NULL INPUT
7F40 98     TKO    TYA           SHOULD STRING BE TOKENIZED?
```

(continued)

the fourth byte from the beginning of the line. FINI prints a space to separate line number and line, and then PLINE prints all or part of the line and counts the characters in the line. COUNTR looks at the number of characters in the line just printed and decides whether LINENT, the line counter, shall be incremented by one, two, or three. CHEC decides if enough lines have been printed. If so, it calls INCHAR, which waits for a keystroke. Any other key causes an exit to the immediate mode, without the "OK" message.

CHANGE tests CFLAG and, if it is set, subtracts the length of the search string (SLEN) from the length of the change string (CLEN). If the two are equal, CHANGE goes directly to CEQUAL, where the change string replaces the workspace string. If CLEN is longer than SLEN, MOVEUP calls PUSHUP, a routine copied from ROM. PUSHUP makes room in the BASIC workspace for the longer change string. REPLAC is called to insert the change string into the BASIC program. LONG tests the new line length to see if it's longer than 71 characters. A graphics character $E9 is printed after the line number if the line is too long. If CLEN is less than SLEN, CHANGE branches to MOVDWN. Part of the BASIC-in-ROM line delete routine is paraphrased in MOVDWN, then DELETE is called to move the BASIC lines down and delete the extra bytes in the program. REPLAC is called to insert the change string. CHAIN rechains the BASIC line pointers. RETURN resets the BASIC workspace index (ORIGIN) and jumps back into the search loop.

Developing SURCHANGE was a real challenge. Many thanks to Earl Morris for advice and for finding the bugs in the program.

Listing 1 *(continued)*

```
7F41 A49C            LDY QFLAG
7F43 D008            BNE RTN
7F45 E8              INX
7F46 20A8A3          JSR TOKBUF    TOKENIZE STRING
7F49 98              TYA           FIND LENGTH OF STRING
7F4A 38              SEC
7F4B E906            SBC #6
7F4D 60       RTN    RTS
7F4E                 ;
7F4E A49A     PLINE  LDY ORIGIN    PRINT WORKSPACE STRING
7F50 8497     PO     STY YSAVE
7F52 B16E            LDA (POINT),Y
7F54 F0F7            BEQ RTN       END OF LINE?
7F56 101E            BPL PRINT     BRANCH IF NOT A TOKEN
7F58 38       TOKEN  SEC           FIND KEYWORD IN TABLE
7F59 E97F            SBC #$7F
7F5B AA              TAX
7F5C A0FF            LDY #$FF
7F5E CA       TO     DEX
7F5F F008            BEQ T2
7F61 C8       T1     INY           PRINT KEYWORD
7F62 B984A0          LDA TOKTBL,Y
7F65 10FA            BPL T1
7F67 30F5            BMI TO
7F69 C8       T2     INY
7F6A B984A0          LDA TOKTBL,Y
7F6D 3007            BMI PRINT     PRINT LAST CHAR. IN KYWORD
7F6F E66B            INC CHRCNT
7F71 20E5A8          JSR OUTPUT
7F74 D0F3            BNE T2
7F76 297F     PRINT  AND #$7F      ZERO HI BIT
7F78 20E5A8          JSR OUTPUT    PRINT CHARACTER
7F7B E66B            INC CHRCNT
7F7D A497            LDY YSAVE     DONE PRINTING LINE?
7F7F C49F            CPY YINDEX
7F81 C8              INY
7F82 90CC            BCC PO
7F84 60              RTS
7F85                 ;
7F85 A46C     REPLAC LDY CLEN      INSERT CHANGE STRING
7F87 B91300   REO    LDA BUFF,Y
7F8A C907            CMP #7        DON'T CARE CHAR?
7F8C F002            BEQ RE1
7F8E 91AA            STA (WPOINT),Y
7F90 88       RE1    DEY
7F91 10F4            BPL REO       BRANCH ALWAYS
7F93 60              RTS
7F94                 ;
7F94 BDA07F   PROMPT LDA TABL,X    PRINT A MESSAGE
7F97 E8              INX
7F98 C60E            DEC SCNCNT    AVOID AUTO CR/LF
7F9A 20E5A8          JSR OUTPUT    PRINT ONE CHARACTER
7F9D D0F5            BNE PROMPT    LOOP IF CHAR NOT A NULL
7F9F 60              RTS
7FA0                 ;
7FA0          TABL
7FA0 0D              .BYTE $D,$A,'SEARCH '
7FA1 0A
7FA2 53
7FA3 45
```

(continued)

Listing 1 *(continued)*

```
415 7FA4 41
415 7FA5 52
415 7FA6 43
415 7FA7 48
415 7FA8 20
416 7FA9 4F         .BYTE 'OPTIONS:',$D,$A
416 7FAA 50
416 7FAB 54
416 7FAC 49
416 7FAD 4F
416 7FAE 4E
416 7FAF 53
416 7FB0 3A
416 7FB1 0D
416 7FB2 0A
417 7FB3 20         .BYTE ' 1-PRINT 2-STMT'
417 7FB4 31
417 7FB5 2D
417 7FB6 50
417 7FB7 52
417 7FB8 49
417 7FB9 4E
417 7FBA 54
417 7FBB 20
417 7FBC 32
417 7FBD 2D
417 7FBE 53
417 7FBF 54
417 7FC0 4D
417 7FC1 54
418 7FC2 20         .BYTE ' 3-LINE',$D,$A
418 7FC3 33
418 7FC4 2D
418 7FC5 4C
418 7FC6 49
418 7FC7 4E
418 7FC8 45
418 7FC9 0D
418 7FCA 0A
419 7FCB 20         .BYTE ' 4-QUOTE 5-VAR 6-'
419 7FCC 34
419 7FCD 2D
419 7FCE 51
419 7FCF 55
419 7FD0 4F
419 7FD1 54
419 7FD2 45
419 7FD3 20
419 7FD4 35
419 7FD5 2D
419 7FD6 56
419 7FD7 41
419 7FD8 52
419 7FD9 20
419 7FDA 36
419 7FDB 2D
420 7FDC 43         .BYTE 'CHANGE',$D,$A
420 7FDD 48
420 7FDE 41
420 7FDF 4E
```

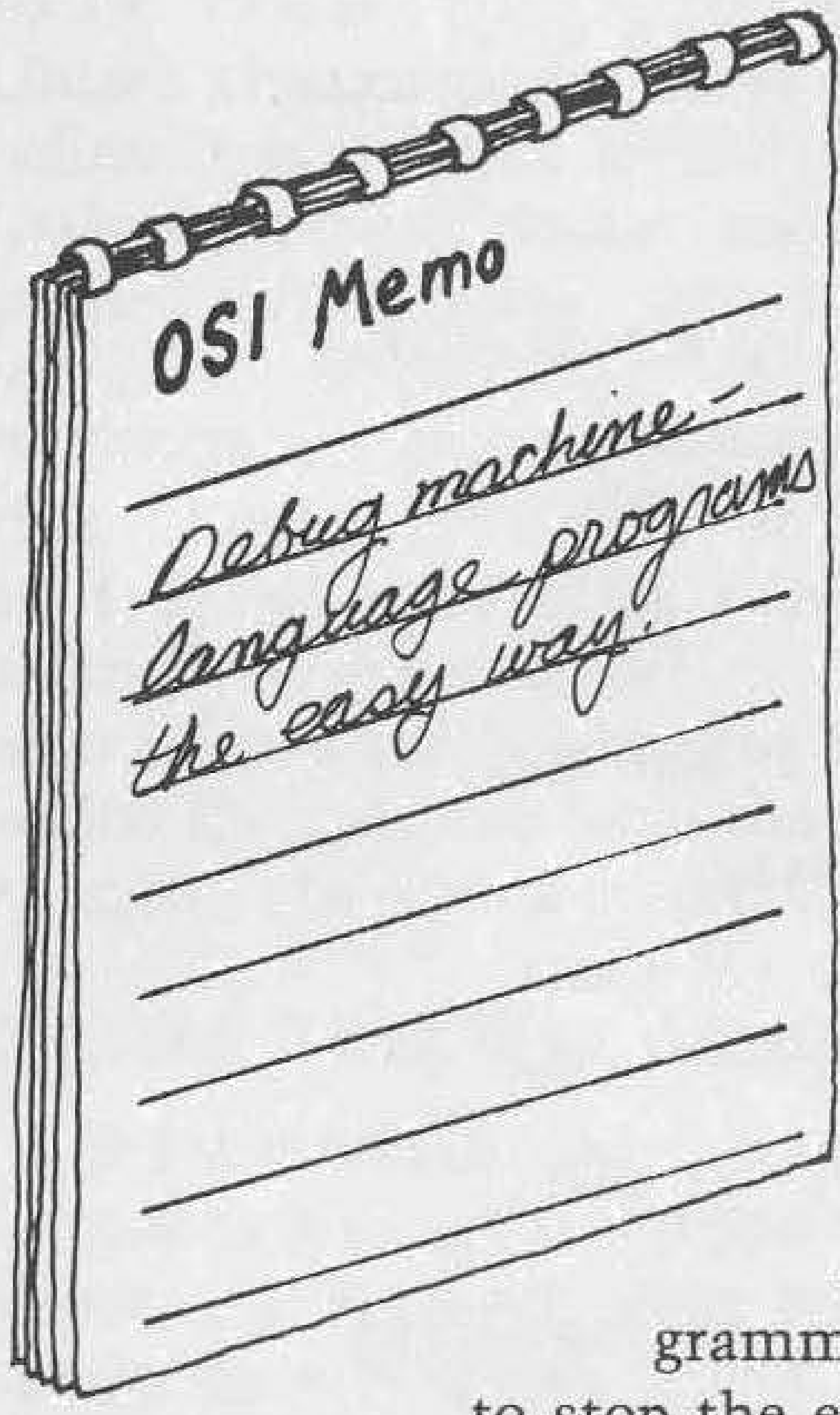
(continued)

Listing 1 *(continued)*

```
420 7FE0 47
420 7FE1 45
420 7FE2 0D
420 7FE3 0A
421 7FE4 4F          .BYTE 'OPTIONS',0
421 7FE5 50
421 7FE6 54
421 7FE7 49
421 7FE8 4F
421 7FE9 4E
421 7FEA 53
421 7FEB 00
422 7FEC 53          .BYTE 'SEARCH',0
422 7FED 45
422 7FEE 41
422 7FEF 52
422 7FF0 43
422 7FF1 48
422 7FF2 00
423 7FF3 43          .BYTE 'CHANGE',0
423 7FF4 48
423 7FF5 41
423 7FF6 4E
423 7FF7 47
423 7FF8 45
423 7FF9 00
424 7FFA 45          .BYTE 'EXIT?',0
424 7FFB 58
424 7FFC 49
424 7FFD 54
424 7FFE 3F
424 7FFF 00
```

An Improved Breakpoint Utility

by John S. Seybold



A while back I wrote a very basic breakpoint utility for the C1P, which was published in MICRO (49:84). Since then I have written an enhanced version of that utility. The new routine has several improvements over the original, including a hexadecimal display. I urge anyone who is interested in learning more about machine-language programming to read on, as you do not have to be an expert to use this utility. For those who may have missed the first article, I will start with a review of the use and operation of a breakpoint routine.

A breakpoint utility is used as an aid in machine- or assembly-language programming. The idea is to allow the programmer to stop the execution of a machine-language program, check various processor parameters, and then resume program execution. This is done by setting breakpoints at certain locations in the program. This particular utility displays the contents of the A, X, and Y registers, and the status flag register.

To set a breakpoint in the program, I use the 6502's BRK (break) instruction. When the 6502 encounters a BRK instruction, it treats the instruction as a software interrupt. In other words, it stops whatever it is doing and jumps to an interrupt routine — in this case, the breakpoint utility. When the processor is finished with the interrupt routine, it returns to the original program and resumes execution where it left off.

When the 6502 receives an interrupt or executes a BRK instruction it stores the contents of the status register on the stack and the address of the next instruction that it was going to execute. This is the only apparent difference between an interrupt request and BRK instruction. If a

Listing 1

```
0000                ;**********************
0000                ;* BREAKPOINT UTILITY *
0000                ;*                    *
0000                ;* BY JOHN S. SEYBOLD *
0000                ;**********************
0000                ;
0000                   SCR=$D310            STATUS REG DISPLAY
0000                 SCR.A=SCR-$86          A-REG. DISPLAY
0000                 SCR.X=SCR-$46          X-REG. DISPLAY
0000                 SCR.Y=SCR-$6           Y-REG. DISPLAY
0000                ;
1F50                 *=$1F50
1F50 D8                    CLD
1F51 8DE71F                STA A.SAVE  SAVE A-REGISTER
1F54 68                    PLA         PULL STATUS REGG.
1F55 48                    PHA         PUSH IT ON STACK AGAIN
1F56 8DE81F                STA STATUS  SAVE STATUS
1F59 8EE91F                STX X.SAVE  SAVE X
1F5C 8CEA1F                STY Y.SAVE  SAVE Y
1F5F A207                  LDX #7
1F61 2901           LOOP   AND #1      MASK LO BIT OF STATUS
1F63 18                    CLC
1F64 6930                  ADC #$30      CONVERT TO ASCII
1F66 9D50D3                STA SCR+$40,X PRINT STATUS BIT
1F69 ADE81F                LDA STATUS
1F6C 4A                    LSR A       GET NEXT BIT
1F6D 8DE81F                STA STATUS
1F70 CA                    DEX
1F71 10EE                  BPL LOOP    LOOP IF NOT DONE
1F73                ;
1F73                ;*******PRINT LABELS*******
1F73                ;
1F73 A941                  LDA #'A     PRINT 'A' LABEL
1F75 8D8AD2                STA SCR.A
1F78 A958                  LDA #'X     PRINT 'X' LABEL
1F7A 8DCAD2                STA SCR.X
1F7D A959                  LDA #'Y     PRINT 'Y' LABEL
1F7F 8D0AD3                STA SCR.Y
1F82 A207                  LDX #7
1F84 BDDF1F         L1     LDA TABLE,X  PRINT STATUS LABELS
1F87 9D10D3                STA SCR,X
1F8A CA                    DEX
1F8B 10F7                  BPL L1
1F8D                ;
1F8D                ;*******PRINT REGISTERS*******
1F8D                ;
1F8D ADE71F                LDA A.SAVE  GET A-REG. CONTENTS
1F90 20C51F                JSR CONVRT  CONVERT TO HEX #
1F93 8E8CD2                STX SCR.A+2 PRINT HEX ON SCREEN
1F96 8C8DD2                STY SCR.A+3
1F99 ADE91F                LDA X.SAVE  GET X
1F9C 20C51F                JSR CONVRT  CONVERT TO HEX
1F9F 8ECCD2                STX SCR.X+2 AND PRINT
1FA2 8CCDD2                STY SCR.X+3
1FA5 ADEA1F                LDA Y.SAVE  GET Y
1FA8 20C51F                JSR CONVRT  CONVERT AND PRINT
1FAB 8E0CD3                STX SCR.Y+2
1FAE 8C0DD3                STY SCR.Y+3
```

(continued)

BRK instruction is executed, the processor skips one byte when it returns from the routine. Hence, the first byte following a BRK instruction is never executed by the processor. When a BRK instruction is executed, the processor sets the B bit in the status register so that it can differentiate between a BRK instruction and a hardware interrupt.

Once the processor has executed the BRK instruction, you may use it to display information on the screen. The processor then jumps to the C1P's keyboard routine and waits for a key to be depressed. (This is how you make it wait for a command before returning to the original program.) You must be careful not to change anything that might affect the main program. Therefore, save all the registers before you change them. After you release the processor from the keyboard, the utility restores all the registers to their previous values and returns to the main program *via* the RTI (return from interrupt) instruction.

In addition to displaying all three of the user registers, the breakpoint utility prints the contents of the status register on the screen. Since the last thing the processor does before entering the breakpoint routine is save the processor status register, that register is the top element on the stack. To retrieve it, simply put the contents of the A register in a safe place and execute a PLA (pull accumulator) instruction. Now you have the processor status register in the accumulator and can display it on the screen. The loop in lines 200 to 280 of the breakpoint utility listing displays the ASCII equivalent of each bit of the register on the screen; i.e., "0" or "1".

Lines 320 to 420 of the breakpoint routine print the labels for the registers A, X, and Y, and for each of the status bits. Lines 460 to 570 print the contents of the user registers in hexadecimal on the screen. A sample printout is shown in figure 1. Once everything has been printed, the routine restores the X and Y registers and then jumps to the keyboard routine, which uses only the A register. If an "S" is entered from the keyboard, the processor will jump to the C1P monitor rather than back to the main program. If any key other than an "S" is depressed, the processor restores the A register and returns to the main program and continues execution.

Figure 1: Sample Status Output

```
A  AB
X  47
Y  FF   NV BDIZC
        10110100
```

Listing 1 *(continued)*

```
1FB1                 ;
1FB1                 ;*******EXIT*******
1FB1                 ;
1FB1 AEE91F                 LDX X.SAVE   RESTORE X AND Y
1FB4 ACEA1F                 LDY Y.SAVE
1FB7 2000FD                 JSR $FD00    POLL KEYBOARD
1FBA C953                   CMP #'S      IS IT AN 'S'?
1FBC D003                   BNE DONE
1FBE 4C00FE                 JMP $FE00    TO MONITOR
1FC1 ADE71F        DONE     LDA A.SAVE   RESTORE A-REG.
1FC4 40                     RTI          AND REENTER PROGRAM
1FC5                 ;
1FC5                 ;*******SUBROUTINES*******
1FC5                 ;
1FC5 48            CONVRT PHA               TEMP. SAVE A-REG.
1FC6 290F                 AND #%00001111
1FC8 20D61F               JSR CHECK         LO NYBBLE TO ASCII
1FCB A8                   TAY               SAVE IT IN Y
1FCC 68                   PLA               RESTORE A-REG.
1FCD 4A                   LSR A        MOVE HI NYBBLE TO LO
1FCE 4A                   LSR A
1FCF 4A                   LSR A
1FD0 4A                   LSR A
1FD1 20D61F               JSR CHECK    CONVERT TO ASCII
1FD4 AA                   TAX          SAVE IT IN X
1FD5 60                   RTS
1FD6                 ;
1FD6 0930          CHECK  ORA #$30     ADD #$30 TO GET ASCII
1FD8 C93A                 CMP #$3A     GREATER THAN 10?
1FDA 9002                 BCC FIXED
1FDC 6906                 ADC #6       ADD #7 FOR A-F
1FDE 60            FIXED  RTS
1FDF                 ;
1FDF 4E            TABLE .BYTE 'NV BDIZC'
1FE0 56
1FE1 20
1FE2 42
1FE3 44
1FE4 49
1FE5 5A
1FE6 43
1FE7 00            A.SAVE .BYTE 0
1FE8 00            STATUS .BYTE 0
1FE9 00            X.SAVE .BYTE 0
1FEA 00            Y.SAVE .BYTE 0
```

Figure 2: Status Bit Definitions

N	Negative Flag
V	Overflow Flag
	Unused
B	Break Flag
D	Decimal Mode Flag
I	Interrupt Mask Flag
Z	Zero Flag
C	Carry Flag

Using the Utility

An experience common to most machine-code programmers is having a program consistently return with odd results or, worse yet, not return at all. When you use the Breakpoint Utility, you can go through the program in small steps and isolate the problem. In most cases, breakpoints can be added to the program without reassembly.

Give the utility a try. You will have to enter the machine code into your computer through the monitor or, if you have an assembler, enter the source code and assemble it. Once the program is in memory, I recommend you make a copy of it on tape before proceeding.

The first thing you must do to set up the utility is to point the IRQ vector to the utility. When the 6502 receives an IRQ (interrupt request) or a BRK instruction, it will jump to whatever address is held in the last two bytes of memory. This is where OSI systems have their ROMs and the address in those two locations is $01C0. The first step, then, is to use the monitor to enter $4C,50,1F starting at $01C0, which tells the processor to jump to $1F50 (the address of the utility) when it executes a BRK instruction. Once this is done, you can try using the utility.

Enter a short test program at $0500 (see figure 3). The NOP (no operation) instruction is only a place-keeper to remind you that the 6502 will skip a byte when it returns from the utility. The NOP is never actually executed. To remind yourself that one byte is skipped upon return, you should use a NOP instruction in this spot each time you use the routine. If the BRK instruction is put in over a three-byte instruction, be sure to fill in the rest of the instruction (two bytes) with NOPs so the processor does not resume execution in the middle of an instruction.

Figure 3: Test Program

```
500  18                   CLC
501  A900                 LDA #0
503  AA                   TAX
504  A8                   TAY
505  00        L O O P BRK
506  EA                   NOP
507  6940                 ADC #$40
509  4C0505               JMP LOOP
```

Now go to $500 and run the test program. Immediately you should see a display like that in figure 4. If you do not, check the test program and then the Breakpoint Utility for errors. Once you have the display on screen, the processor waits for you in the keyboard routine. Examine the display before resuming.

Figure 4: Test Program Status

```
A   00
X   00
Y   00   NV BDIZC
         00110110
```

First notice that the C bit is indeed zero, as it should be since the first instruction in the test program cleared it. Also notice that all the registers contain $CC and the Z bit is set, since the last instruction before the breakpoint transferred $00 into Y. You can also see that the N and V bits are zero, as they should be. The B bit is set indicating that the processor has executed a BRK instruction, as expected. The blank spot is an unused bit in the status register. The status bits are defined in figure 2. For further information on the status bits, consult the reference at the end of this article.

Now, if you press any key except "S", the program will go through its loop once and return to the utility. Observe that the contents of A is $40 and that the Z bit has been cleared, indicating that the result of the last operation was not zero. If you depress a key again, the Breakpoint Utility comes back with $80 in A and with the N and V bits set. N was set because the most significant bit of the result of the last operation was set, meaning that it is a negative number in two's complement arithmetic. The V bit is set because there was a carry from bit 6 to bit 7 in the result, which implies a sign change in two's complement arithmetic.

If you send the program through the loop again, the V bit is cleared and the contents of A change again. The next time through the loop the contents of A is $00, the Z bit is set, the N bit is cleared, and the C bit set. If you go through the loop once more, you see that since the C bit was not cleared, it was added in with the result so $41 is in A.

The Breakpoint Utility can give you a lot of information with very little effort about what is happening in your program. I thought it would be nice to have the contents of the program counter also printed out so you could keep your place when using multiple breakpoints, but I felt it would make the program too long. If you have more than 8K of memory, you may wish to relocate the utility. This should not be too difficult, but be sure you change all the subroutine calls and table references and do not forget to put the new starting address into locations $01C1 and $01C2. You might be able to modify the utility for use on bigger OSI machines, but I am not sure what changes would be necessary.

Review of Operating Instructions

1. Load Breakpoint Utility into memory.
2. Enter $4C,50,1F into memory starting at $01C0.
3. Add breakpoints to program under test by keying in BRK instructions ($00) at the desired locations. Remember that the byte following the BRK instruction is ignored.
4. Press the "S" key to stop the utility and jump to the monitor. Press any other key to return to the program under test.

Reference

1. De Jong, M., *Programming and Interfacing the 6502 with Experiments*, Sams, 1980.

Now, if you alter any key change [illegible] the prog may well be changing the [illegible] [illegible]

[illegible]

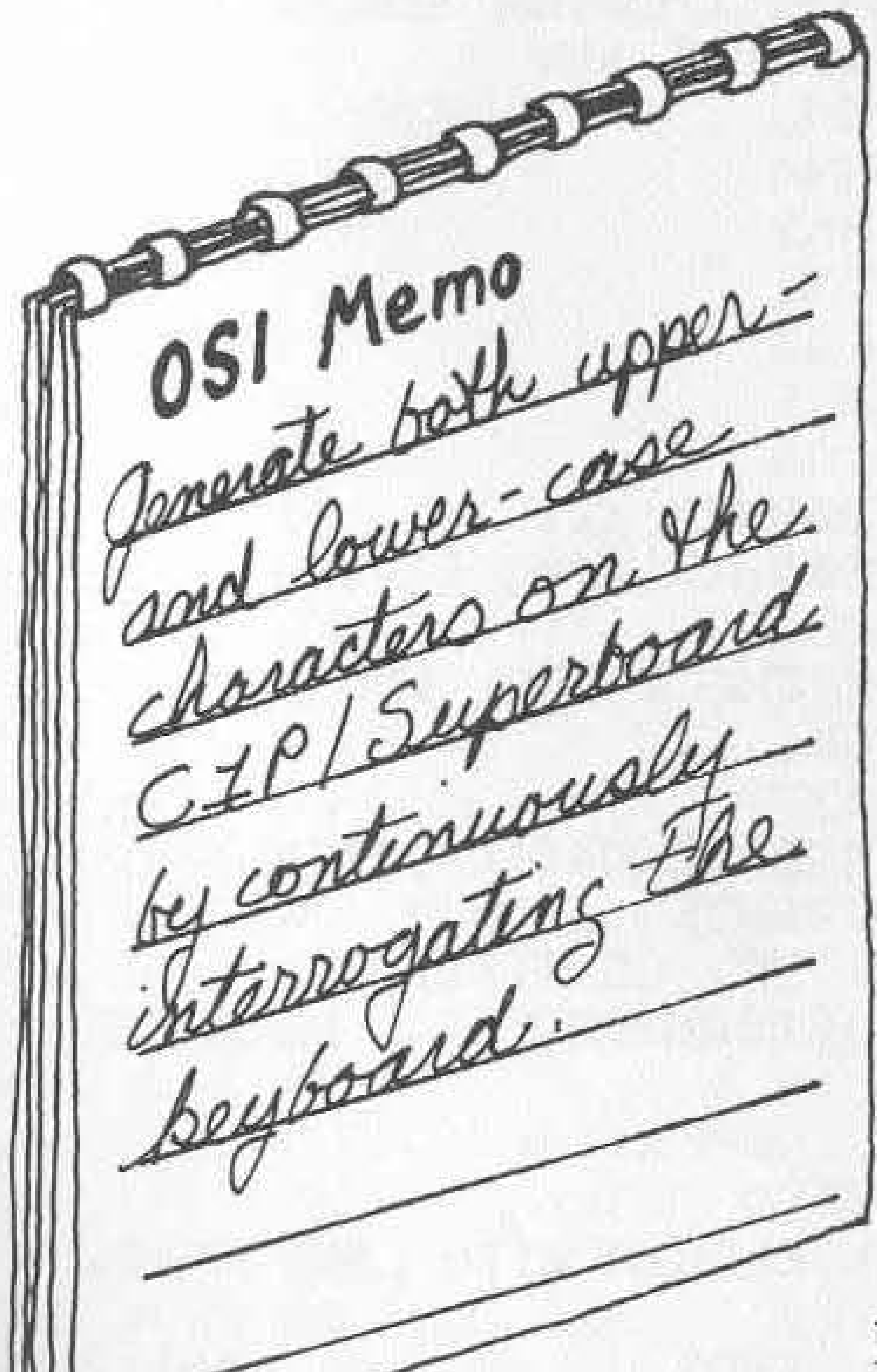
[illegible] that changes would be necessary.

Review of Operating Instructions

[illegible]

Polled Keyboard for C1P/Superboard

by Michael J. Alport



I had been thinking of writing a program that would enable the OSI keyboard to operate as an ordinary typewriter in conjunction with a word processor when an article appeared in MICRO (22:17) describing just such a program. I was pleased at the thought of having a debugged program that had to be keyed in only. My joy was short-lived, however, when I realized that Edward Carlson's program was written for the 542 board and would not work with the 600 board found in the C1P/Superboard microcomputer. The difference between the two boards is quite simple. Instead of polling the rows/columns with a byte consisting of a combination of seven 0's and a 1, the 600 board uses a combination of seven 1's and a 0. I suspected that a simple fix would be to replace all Mr. Carlson's STA $DF00 and LDA $DF00 instructions with JSR $FCBE's and JSR $FCCF's, respectively. These are monitor routines that use an EOR #$FF to invert the bit pattern, replacing 1's with 0's and *vice versa*. However, it is sometimes easier to rewrite a complete program than to attempt to modify someone else's. So while I was rewriting the program, I took the opportunity to add a number of features that were not included in the original program.

The program itself should be self-explanatory, especially when read in conjunction with Mr. Carlson's article. I will, however, make a few comments about the additional features included in my program.

The shift-lock key is continually polled to determine whether it is in the up or down position. If it is in the down position, control is transferred to the normal monitor keyboard routine beginning at $FEED. If the

Listing 1

```
0000                ;***********************
0000                ;* CIP POLLED KEYBOARD  *
0000                ;*                      *
0000                ;* BY MICHAEL J. ALPORT *
0000                ;***********************
0000                ;
0000                 CRTEMU=$BF2D        PRINT CHAR TO SCREEN
0000                 KBPOLL=$FD00        KEYBOARD POLLING ROUTINE
0000                 KYPORT=$DF00        KEYBOARD PORT
0000                ;
1F00                 *=$1F00
1F00                ;
1F00 20211F   ENTER   JSR KEYBRD   MAIN ROUTINE
1F03 8D8B1F           STA LOC      SAVE CHAR FOR REPEAT
1F06 202DBF           JSR CRTEMU   PRINT CHAR ON SCREEN
1F09 20F51F           JSR DELAY    DEBOUNCE KEY
1F0C A900     KYDONE  LDA #0
1F0E 8D00DF           STA KYPORT
1F11 AD00DF           LDA KYPORT
1F14 C9FF             CMP #$FF
1F16 F004             BEQ N1
1F18 C9FE     NEXT    CMP #$FE
1F1A D0F0             BNE KYDONE
1F1C 20F51F   N1      JSR DELAY    DEBOUNCE KEY
1F1F F0DF     LOOP    BEQ ENTER    BRANCH ALWAYS
1F21                ;
1F21 A2FE     KEYBRD  LDX #%11111110  CHECK CTRL ROW
1F23 8E00DF           STX KYPORT
1F26 AE00DF           LDX KYPORT
1F29 8E8A1F           STX CTRL        SAVE UNTIL LATER
1F2C E0FE             CPX #%11111110  SHIFT LOCK ?
1F2E D003             BNE CONT        UP, CONTINUE
1F30 4C00FD           JMP KBPOLL      DOWN, TO REG. ROUTINE
1F33                ;
1F33 E07F     CONT    CPX #%01111111  REPEAT?
1F35 D004             BNE NREP        NO
1F37 AD8B1F           LDA LOC         RETURN WITH LAST CHAR.
1F3A 60               RTS
1F3B E0DF     NREP    CPX #%11011111  ESC?
1F3D D003             BNE CHAR        NO
1F3F A91B             LDA #$1B        RETURN WITH $1B
1F41 60               RTS
1F42 A007     CHAR    LDY #7       SET UP ROW COUNT
1F44 88       ROW     DEY          BEGIN ROW SEARCH
1F45 30DA             BMI KEYBRD   NO CHARACTER, TRY AGAIN
1F47 A207             LDX #7       SET UP COLUMN COUNT
1F49 CA       COL     DEX          BEGIN COLUMN SEARCH
1F4A 30F8             BMI ROW
1F4C B9EE1F           LDA MASK,Y   LOAD MASK BYTE
1F4F 8D00DF           STA KYPORT
1F52 AD00DF           LDA KYPORT
1F55 DDEE1F           CMP MASK,X   COMPARE WITH MASK BYTE
1F58 D0EF             BNE COL      NOT A MATCH
1F5A 8E891F   CALC    STX XREG     SAVE COL. COUNT
1F5D A900             LDA #0       CALC. CHAR. POSITION
1F5F 18               CLC
1F60 88       AGAIN   DEY
1F61 3004             BMI ADDX
```

(continued)

shift-lock is up, the new keyboard routine is executed. Therefore you can use the new keyboard routine in conjunction with BASIC by placing the address of this keyboard routine in BASIC's input vector location.

I found it necessary to add a delay routine (in addition to the original KYDONE routine) to eliminate excessive contact bounce found on my keyboard. This delay routine may not be needed on other keyboards.

Listing 1 *(continued)*

```
590 1F63 6907              ADC #7
600 1F65 90F9              BCC AGAIN
610 1F67 6D891F   ADDX     ADC XREG
620 1F6A A8                TAY
630 1F6B AD8A1F            LDA CTRL         CHECK FOR SHIFT KEY
640 1F6E 2906              AND #%00000110
650 1F70 C906              CMP #%00000110
660 1F72 F005              BEQ NSHIFT       NOT SHIFT
670 1F74 18                CLC          SHIFT- ADD 49 TO CHAR POINTER
680 1F75 98                TYA
690 1F76 6931              ADC #49
700 1F78 A8                TAY
710 1F79 BE8C1F   NSHIFT   LDX CHRTBL,Y   GET CHAR FROM TABLE
720 1F7C AD8A1F            LDA CTRL        CHECK FOR CTRL KEY
730 1F7F 2940              AND #%01000000
740 1F81 D004              BNE NCTRL       NOT CTRL
750 1F83 8A                TXA
760 1F84 0980              ORA #%10000000  SET HI BIT
770 1F86 60                RTS
780 1F87          ;
790 1F87 8A       NCTRL    TXA
800 1F88 60                RTS
810 1F89          ;
820 1F89 00       XREG     .BYTE 0      X-REG. STORAGE
830 1F8A 00       CTRL     .BYTE 0      CTRL KEY STORAGE
840 1F8B 00       LOC      .BYTE 0      KEY STORAGE
850 1F8C          ;
860 1F8C 31       CHRTBL   .BYTE '1234567890:-'
860 1F8D 32
860 1F8E 33
860 1F8F 34
860 1F90 35
860 1F91 36
860 1F92 37
860 1F93 38
860 1F94 39
860 1F95 30
860 1F96 3A
860 1F97 2D
870 1F98 7F                .BYTE $7F,$20,'.lo',$A,$D,$20,$20
870 1F99 20
870 1F9A 2E
870 1F9B 6C
870 1F9C 6F
870 1F9D 0A
870 1F9E 0D
870 1F9F 20
870 1FA0 20
```

(continued)

Listing 1 *(continued)*

```
880 1FA1 77         .BYTE 'wertyuisdfghjkxcvbnm,'
880 1FA2 65
880 1FA3 72
880 1FA4 74
880 1FA5 79
880 1FA6 75
880 1FA7 69
880 1FA8 73
880 1FA9 64
880 1FAA 66
880 1FAB 67
880 1FAC 68
880 1FAD 6A
880 1FAE 6B
880 1FAF 78
880 1FB0 63
880 1FB1 76
880 1FB2 62
880 1FB3 6E
880 1FB4 6D
880 1FB5 2C
890 1FB6 71         .BYTE 'qaz',$20,'/;p'
890 1FB7 61
890 1FB8 7A
890 1FB9 20
890 1FBA 2F
890 1FBB 3B
890 1FBC 70
900 1FBD 21         .BYTE '!"#$%&',$27,'()0*='
900 1FBE 22
900 1FBF 23
900 1FC0 24
900 1FC1 25
900 1FC2 26
900 1FC3 27
900 1FC4 28
900 1FC5 29
900 1FC6 30
900 1FC7 2A
900 1FC8 3D
910 1FC9 7F         .BYTE $7F,$20,'>LO',$A,$D
910 1FCA 20
910 1FCB 3E
910 1FCC 4C
910 1FCD 4F
910 1FCE 0A
910 1FCF 0D
920 1FD0 20         .BYTE $20,$20,'WERTYUISDFGHJKXCVBNM'
920 1FD1 20
920 1FD2 57
920 1FD3 45
920 1FD4 52
920 1FD5 54
920 1FD6 59
920 1FD7 55
```

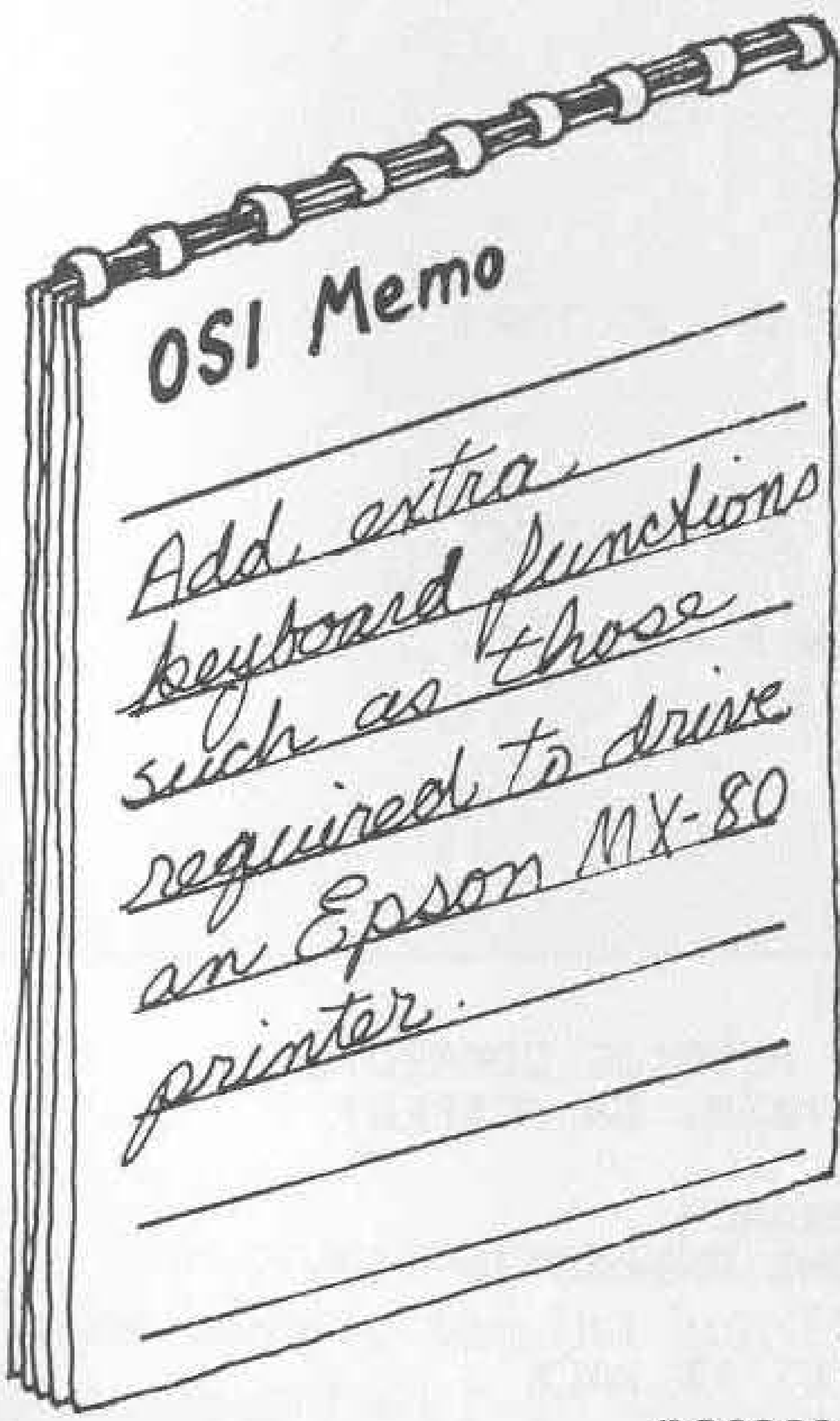
(continued)

Listing 1 *(continued)*

```
 920 1FD8 49
 920 1FD9 53
 920 1FDA 44
 920 1FDB 46
 920 1FDC 47
 920 1FDD 48
 920 1FDE 4A
 920 1FDF 4B
 920 1FE0 58
 920 1FE1 43
 920 1FE2 56
 920 1FE3 42
 920 1FE4 4E
 920 1FE5 4D
 930 1FE6 3C                .BYTE '<QAZ',$20,'?+P'
 930 1FE7 51
 930 1FE8 41
 930 1FE9 5A
 930 1FEA 20
 930 1FEB 3F
 930 1FEC 2B
 930 1FED 50
 940 1FEE          ;
 950 1FEE 7F        MASK    .BYTE %01111111
 960 1FEF BF                .BYTE %10111111
 970 1FF0 DF                .BYTE %11011111
 980 1FF1 EF                .BYTE %11101111
 990 1FF2 F7                .BYTE %11110111
1000 1FF3 FB                .BYTE %11111011
1010 1FF4 FD                .BYTE %11111101
1020 1FF5          ;
1030 1FF5 A2FF      DELAY   LDX #$FF      DEBOUNCE ROUTINE
1040 1FF7 A020      LP1     LDY #$20
1050 1FF9 88        LP2     DEY
1060 1FFA D0FD              BNE LP2
1070 1FFC CA                DEX
1080 1FFD D0F8              BNE LP1
1090 1FFF 60                RTS
```

Something for Nothing

by Leo Jankowski



A frustration encountered when using the C1P is the presence of dead keys, particularly ESC. I became fed up with having to type 'PRINTCHR$(27)' whenever I wanted to hit the ESC key. A disassembly of the ROM BASIC code reveals that all the necessary routines are there in ROM, so it's just a matter of using them. If you're in a hurry, use the following:

```
.0222/20 BA FF C91B F0 07 C9 7F F0
       03 4C 99 A3 4C 69 FF
```

Then warm start and POKE 538,34:POKE 539,2. Hit RUBOUT for a rapid screen clear!

Since I use an Epson MX-80 printer that possesses a plethora of codes, the next step was to program a few keys to access all those codes, thereby controlling the printer. Unfortunately, placing printer commands in a program still demands a command like this:

```
10 PRINTCHR$(27);:PRINT"E"
```

The C1P will print the ESC symbol in a line of BASIC but will not remember it. On the other hand, PRINT " □ E" will work.

Another annoyance is the C1P's habit of mixing graphics with the error codes and then proceeding to tell you that everything is OK. Actually you lose a line on the screen and the cursor!

The following program gets rid of the lot. The new cursor is CHR$(187). If you enter the program in machine code, then the entry point is $0222. Everything after that is automatic; <BREAK> W (or cold start!) does not affect the program. The table lists the keys that access all the Epson codes.

Listing 1

```
0000               ;************************
0000               ;* SOMETHING FOR NOTHING *
0000               ;*                      *
0000               ;*   BY L.J. JANKOWSKI   *
0000               ;************************
0222                *=$0222
0222 A2FF                   LDX #$FF     RESET STACK
0224 9A                     TXS
0225 A949                   LDA #$49     MESSAGE PRINTER VECTOR
0227 8504                   STA $04
0229 A902                   LDA #2
022B 8505                   STA $05
022D A962                   LDA #$62     INPUT VECTOR
022F 8D1802                 STA $0218
0232 A902                   LDA #2
0234 8D1902                 STA $0219
0237 A922                   LDA #$22     WARMSTART VECTOR
0239 8501                   STA $01
023B A902                   LDA #2
023D 8502                   STA $02
023F 4C74A2                 JMP $A274    JUMP TO WARMSTART
0242               ;
0242 0D                     .BYTE $D,$A,0  NEW MESSAGE
0243 0A
0244 00
0245 0D                     .BYTE $D,$A,0
0246 0A
0247 00
0248 60                     RTS
0249               ;
0249 48                     PHA          ERROR MESSAGE CORRECTOR
024A AD65D3                 LDA $D365    ERR. MESS. ON SCREEN?
024D C93F                   CMP #'?
024F D008                   BNE $0259    NO, BRANCH
0251 AD67D3                 LDA $D367    GET 2ND CHARACTER
0254 297F                   AND #$7F     FIX IT
0256 8D67D3                 STA $D367    AND PUT IT BACK
0259 68                     PLA
025A A002                   LDY #$02     PRINT LF,CR
025C A942                   LDA #$42
025E 4CC3A8                 JMP $A8C3
0261               ;
0261 00            XSAVE    .BYTE 0
0262               ;
0262 8E6102                 STX XSAVE    PRINT NEW CURSOR
0265 AE0002                 LDX $0200
0268 A9BB                   LDA #$BB
026A 9D00D3                 STA $D300,X
026D AE6102                 LDX XSAVE
0270 20BAFF                 JSR $FFBA
0273 207C02                 JSR $027C    LOOK FOR CONTROL CODES
0276 4C99A3                 JMP $A399
0279               ;
0279 4C69FF        OUT      JMP $FF69    REGULAR OUTPUT ROUTINE
027C               ;
027C C91B          CHECK    CMP #$1B     ESC?
027E F0F9                   BEQ OUT
0280 C90A                   CMP #$A      LINE FEED?
```

(continued)

Table 1: Control Codes for Epson Printer

		CTRL
FF	form feed	L
HT	horizontal tab	I
VT	vertical tab	K
SO	shift out	N
DC4	cancel SO	T
SI	shift in	O
DC2	cancel SI	R
NUL	null	@
DC1	select	Q
DC3	deselect	S
CAN	cancel	X
DEL	delete	RUBOUT

The program has been designed for easy editing. If you want to add more keys to the list just tack the code onto the end of the program. Always end with an RTS.

Listing 1 *(continued)*

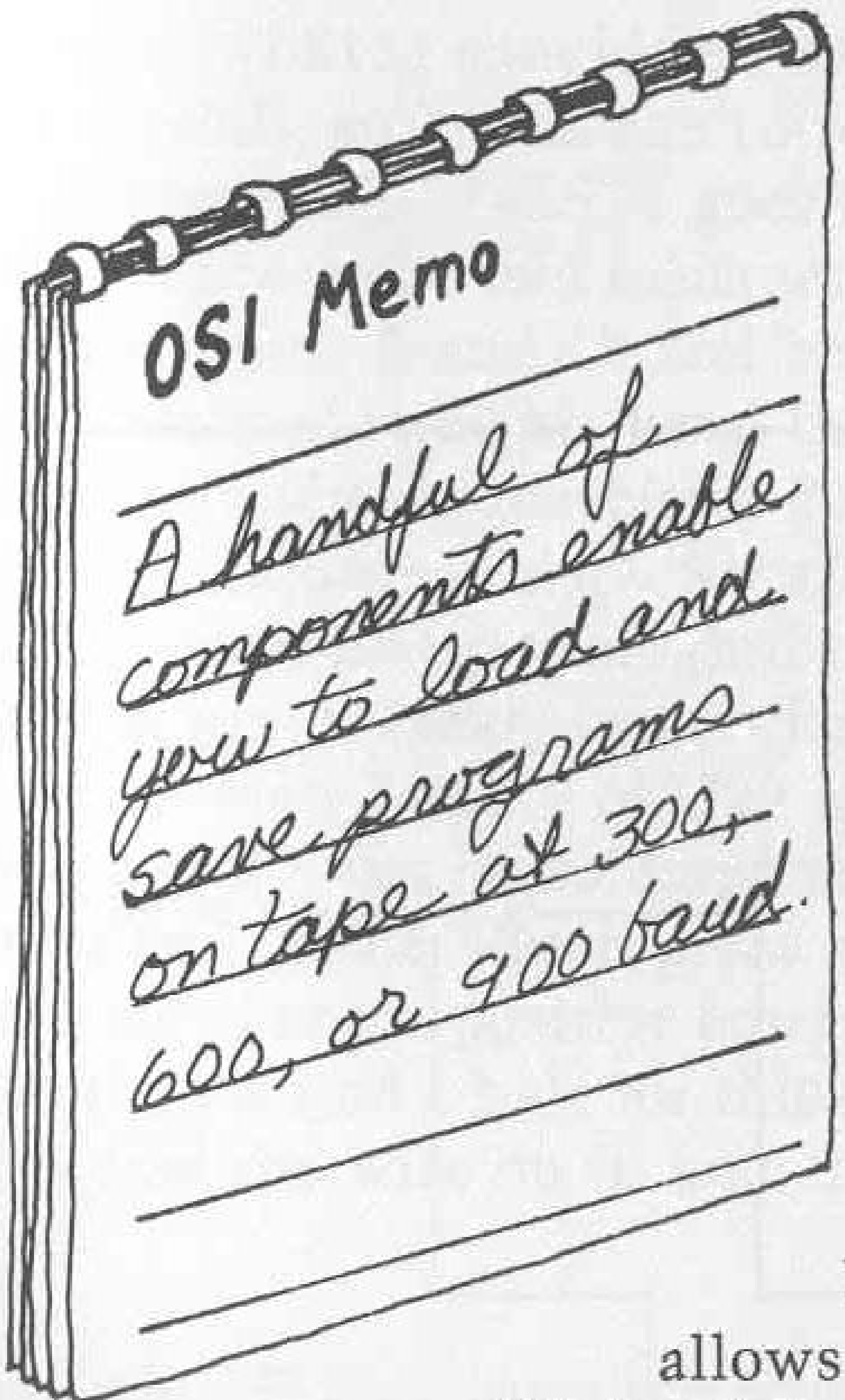
```
0282 F0F5          BEQ OUT
0284 C90C          CMP #$C      FORM FEED?
0286 F0F1          BEQ OUT
0288 C909          CMP #9       CTRL I?
028A F0ED          BEQ OUT
028C C90B          CMP #$B      VERTICAL TAB?
028E F0E9          BEQ OUT
0290 C90E          CMP #$E      SHIFT OUT?
0292 F0E5          BEQ OUT
0294 C914          CMP #$14     DC4?
0296 F0E1          BEQ OUT
0298 C90F          CMP #$F      CTRL O?
029A F0DD          BEQ OUT
029C C912          CMP #$12     DC2?
029E F0D9          BEQ OUT
02A0 C900          CMP #0       NULL?
02A2 F0D5          BEQ OUT
02A4 C911          CMP #$11     DC1?
02A6 F0D1          BEQ OUT
02A8 C913          CMP #$13     DC3?
02AA F0CD          BEQ OUT
02AC C918          CMP #$18     CANCEL?
02AE F0C9          BEQ OUT
02B0 C97F          CMP #$7F     RUBOUT?
02B2 F0C5          BEQ OUT
02B4 60            RTS
```

Listing 2

```
10 FOR I=546 TO 692: READ A: POKE X,A: NEXT
100 DATA162,255,154,169,73,133,4,169,2,133,5,169,98,141,24
110 DATA2,169,2,141,25,2,169,34,133,1,169,2,133,2,76,116
120 DATA162,13,10,0,13,10,0,96,72,173,101,211,201,63,208
130 DATA8,173,103,211,41,127,141,103,211,104,160,2,169,66
140 DATA76,195,168,0,142,97,2,174,0,2,169,187,157,0,211,174
150 DATA97,2,32,186,255,32,124,2,76,153,163,76,105,255,201
160 DATA27,240,249,201,10,240,245,201,12,240,241,201,9,240
170 DATA237,201,11,240,233,201,14,240,229,201,20,240,225
180 DATA201,15,240,221,201,18,240,217,201,0,240,213,201,17
190 DATA240,209,201,19,240,205,201,24,240,201,201,127,240
200 DATA197,96
300 POKE 1,34: POKE 2,2: PRINT "FINISHED"
500 NEW
```

Saving Time with Your C1P

by John S. Seybold



There have been several articles on how to modify the baud rate on the cassette storage circuit of the C1P. However, I decided to submit this approach since I think it is better than any I have seen to date. There are three speeds — 300, 600, and 900 baud — and they all work. The only baud modifications I have seen that work above 600 baud have a very high error rate. At 900 baud, I can load a 6K program in BASIC without any errors. Another advantage of my approach is its simplicity; it uses only two 7400 series ICs, two resistors, and a switch.

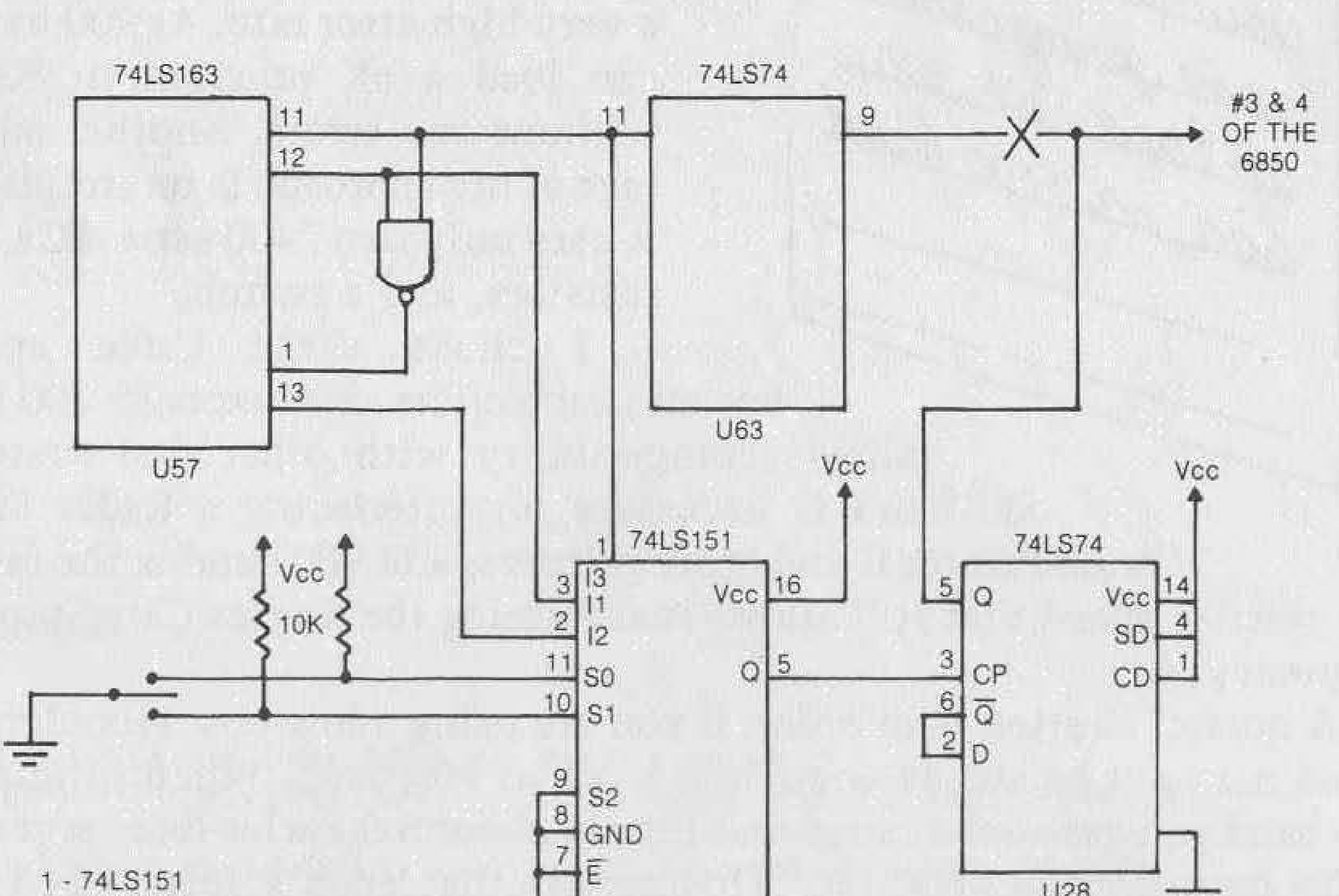
I chose these three speeds because each offers an advantage: 300 baud allows compatibility with other OSI systems; 600 baud is necessary for interfacing a Radio Shack Quick Printer II and other printers; and 900 baud is the fastest reliable speed that still allows maintaining the Kansas City Standard frequency.

A note of caution is in order: if you are using a low-cost recorder, the speed may not be steady enough to work at 900 baud. When running at 900 baud you get a tone burst that is only about 5.3 cycles long since the audio frequency is 4800 Hz. This means that even a relatively small variation in tape speed may cause the KC receiver circuit to miss a bit that it should recognize. Originally I was using a small hand-held recorder that would not work reliably at 900, but replacing it with a slightly better unit solved the problem. I think any recorder in the $50 to $100 range would be acceptable.

Basically what is done is the frequency supplied to the transmit clock of the ACIA is made selectable. Three frequencies — 9.5, 18.9, and 28.4

KHz — are tapped from the 74LS163 divide-by-13 counter (U57). These three signals are fed into the 74LS151 multiplexer, which is installed at location U44. The binary combination at pins 11 and 10 is chosen by the switch position and determines which of the three signals are present at the output of the multiplexer (pin 5). The output is then divided by two with the 74LS74 D flip-flop to correct the asymmetry of the signals from the divide-by-13 counter. The output of the flip-flop is then selectable at 4.7, 9.5, and 14.2 KHz. This is the frequency for the TX clock of the ACIA (U14). By increasing the frequency of the TX clock, you are decreasing the length of the tone burst used for each data bit, which of course speeds up the saving and loading processes.

Figure 1



Construction

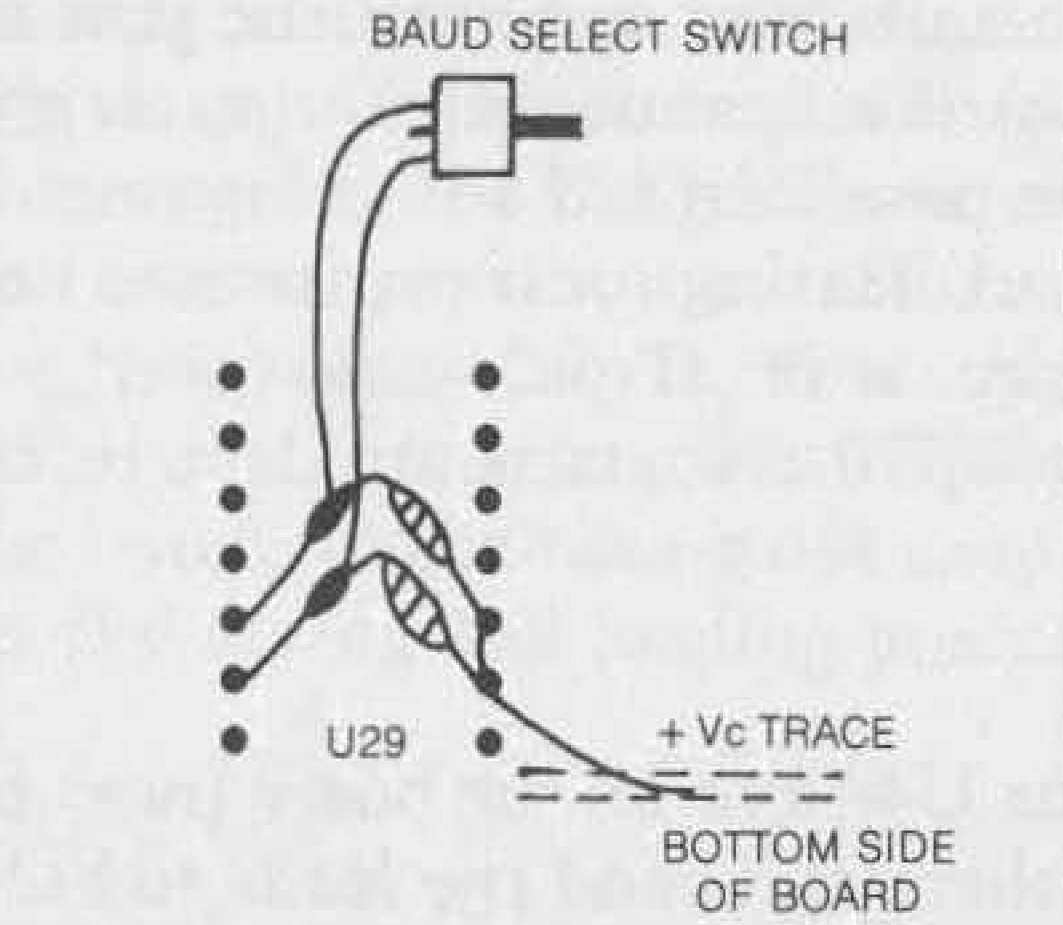
I recommend that you use the sockets for the ICs to prevent damage while soldering. I also suggest that you install them and bend the pins to hold them in place and not solder them until all connections to each pin have been made. This way you avoid the possibility of soldering one of the holes shut, as well as the necessity of heating each pad twice. For hook-up wire I used 28-guage wire-wrap wire. This works well — especially for making the connections to U57 (as there is no place to do this conveniently).

The 74LS151 should be mounted at the U44 slot on the board (next to the crystal), so install the 16-pin socket there and bend the leads to hold it in place. The 74LS74 goes in slot U28, so mount the 14-pin socket there. Locate the two resistors in U29. The resistors should be mounted between pins 2 and 13 and between pins 3 and 12. Leaving one lead of each resistor straight, bend the other lead so that it makes a 45° angle with the body of the resistor. Now strip ½ inch of insulation from the end of a 3-inch piece of wire. Starting at the bottom of the board, run the bare end of the wire up through the hole at pin 2 and then down through the hole at pin 3. Next, insert the straight end of the resistors through the holes at pins 2 and 3 of U29 and the bent ends through pins 13 and 12. Pins 2 and 3 can be soldered now and the leads underneath clipped. The other end of the 3-inch piece of wire should be shortened, stripped, and connected to the positive power bus along the edge where it is the widest. I could not find a hole for this connection, so I just cleaned a spot on the bus, laid the wire on it, and soldered it into place.

Strip 3/8 inch from the end of a 2-inch piece of wire and, starting from the top of the board, connect pins 6 and 7 of U29 and connect the other end to pin 7 of U28 (the 74LS74). Pin 7 of U28 may be soldered, but wait to solder the two connections on U29. Now, using a 12-inch to 18-inch piece of 22-guage stranded wire, insert one end into the hole at pin 7 of U29 and solder. The other end should be soldered to the center lead of the baud-select switch.

Using two more 12-inch to 18-inch pieces of 22-guage stranded wire, connect the bent lead of each resistor to one of the outer two leads on the baud-select switch. The easiest way to do this is to strip ½ inch of insulation from one end of each wire and wrap the end around the bent resistor lead, soldering as shown in figure 2.

Figure 2



Next, connect the foil pads from pins 10 and 11 of U44 to pins 12 and 13 of U29 (the bent ends of the two resistors), respectively, and solder all four connections. Note that it does not matter which resistor lead is connected to which pin of U44 as the switch can be turned around and have the same effect. In all cases, the center position is the 300 baud setting. All the connections to the two resistors are shown in figure 2.

Pins 7, 8, and 9 of U44 all should be connected to ground. The hole next to pin 9 is ideal for this. Use wire-wrap wire and work from the top of the board. You should be able to get all three wires into the ground hole and then solder them into place. Pin 16 of U44 should be connected to the positive bus using a short piece of wire run to the hole right in front of it.

Pins 1, 2, and 3 of U44 now can be connected to pins 11, 12, and 13, respectively, of U57. Starting from the top of the board, hook one end of a piece of wire to pin 1 of U44 and solder. Then run the other end back and down through any one of the holes between U58 and U43. Flip the board over and cut the wire to length. Strip ¼ inch of insulation from the end, carefully heat the foil pad of pin 11 of U57, insert the end of the wire into the hole along with the IC pin, and add just a bit of solder. Repeat this procedure for the other two wires, using the other hole between U58 and U43 for routing the wires. The last wire to be connected to U44 is run from pin 5 to pin 3 of U28.

There are two adjacent foil cuts that must be made near the ACIA. The two cuts are made at W5, just behind pin 13 of the 6850. This cut should disconnect pin 9 of U63 from pins 3 and 4 of the 6850. You may wish to verify this with an ohmmeter. Now run a wire from pin 5 of U28 to the leads from pins 3 and 4 of the 6850. This connection is easiest to make right at W5, as the leads from both 3 and 4 are there and have holes (see figure 3).

To finish connecting the 74LS74, hook pins 1, 4, and 14 to the positive bus. The best way to do this is to connect all three pins together and run a wire to pin 5 of location U29. Now connect pins 2 and 6 of U28 and solder them, as well as any other unsoldered foil pads with connections. This completes the construction. Make a careful visual check of the board and if an ohmmeter is available, use it to verify all connections.

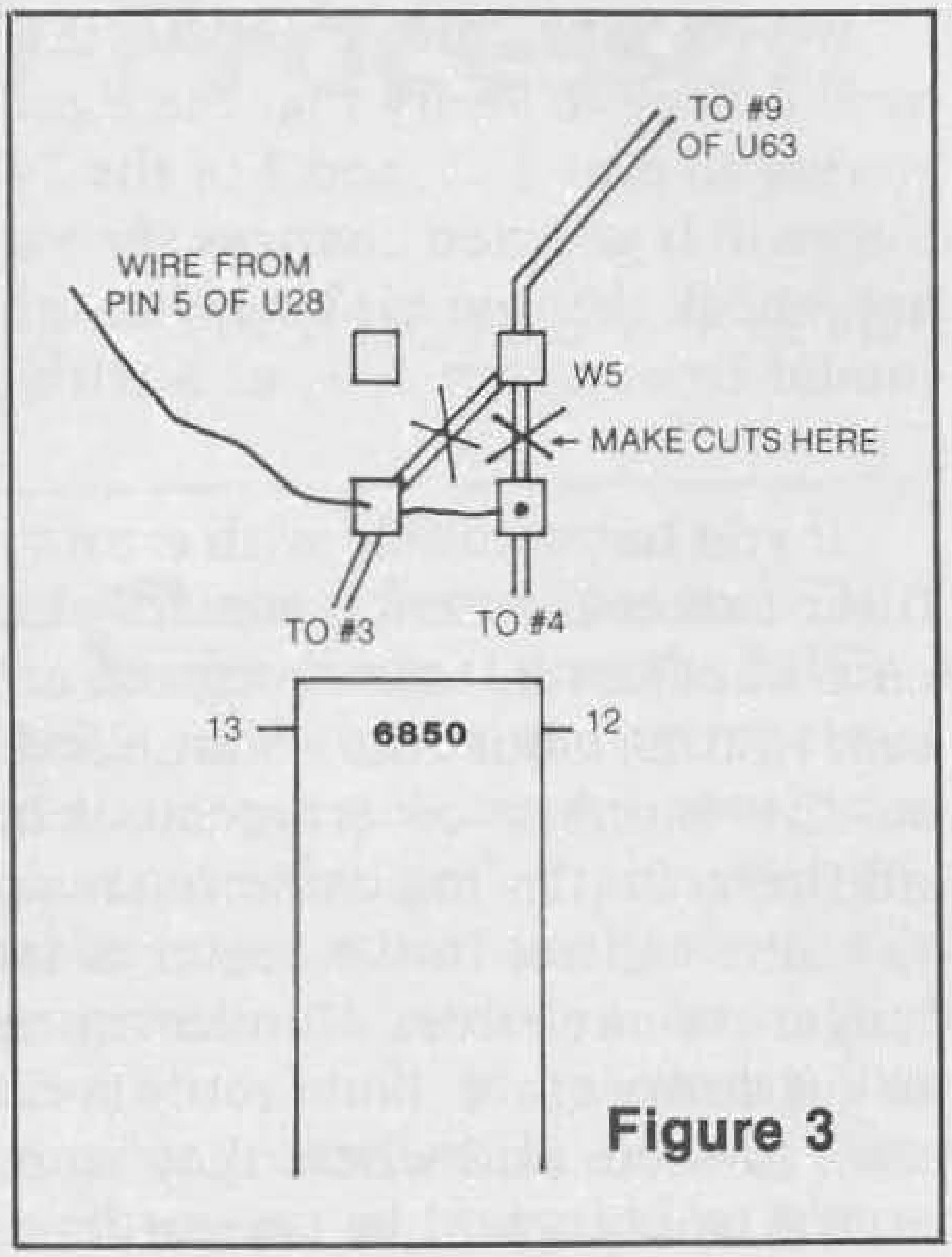


Figure 3

Checkout

Install the two ICs in their respective sockets with pin 1 towards the keyboard and connect the 5-volt supply to the board. With the baud select switch in the center position, load a short BASIC program. If you are unable to load a program, refer to the following section on troubleshooting. Once a program has been loaded, put the machine in the save mode and list the program.

When you change the position of the baud select switch, you should notice the speed of the listing increasing. It is relatively easy to determine which position corresponds to which baud rate. Now try saving and reloading the program at a higher speed. To avoid confusion, you will find it a good idea to label all your tapes with the baud rate at which they were recorded.

Troubleshooting

If you have trouble, the first thing to do is turn off the power and verify all connections with an ohmmeter against the schematic shown in figure 1. Next, with the power on, check voltages on all the pins that should be grounded, or at 5 volts to see that they are. Also, with the baud select switch in the center position, check that pins 10 and 11 of the 74LS151 are at 5 volts.

If you still cannot locate the problem, you will need to use an oscilloscope to verify that the signals from pins 11, 12, and 13 of U57 are getting to pins 1, 3, and 2 of the 74LS151. Then check to see if changing the switch position changes the signal at pin 5 of the 74LS151. If it does not, check the two-bit binary number at pins 10 and 11 of the 74LS151; it should be at either 1, 2, or 3 with pin 10 as the most significant bit.

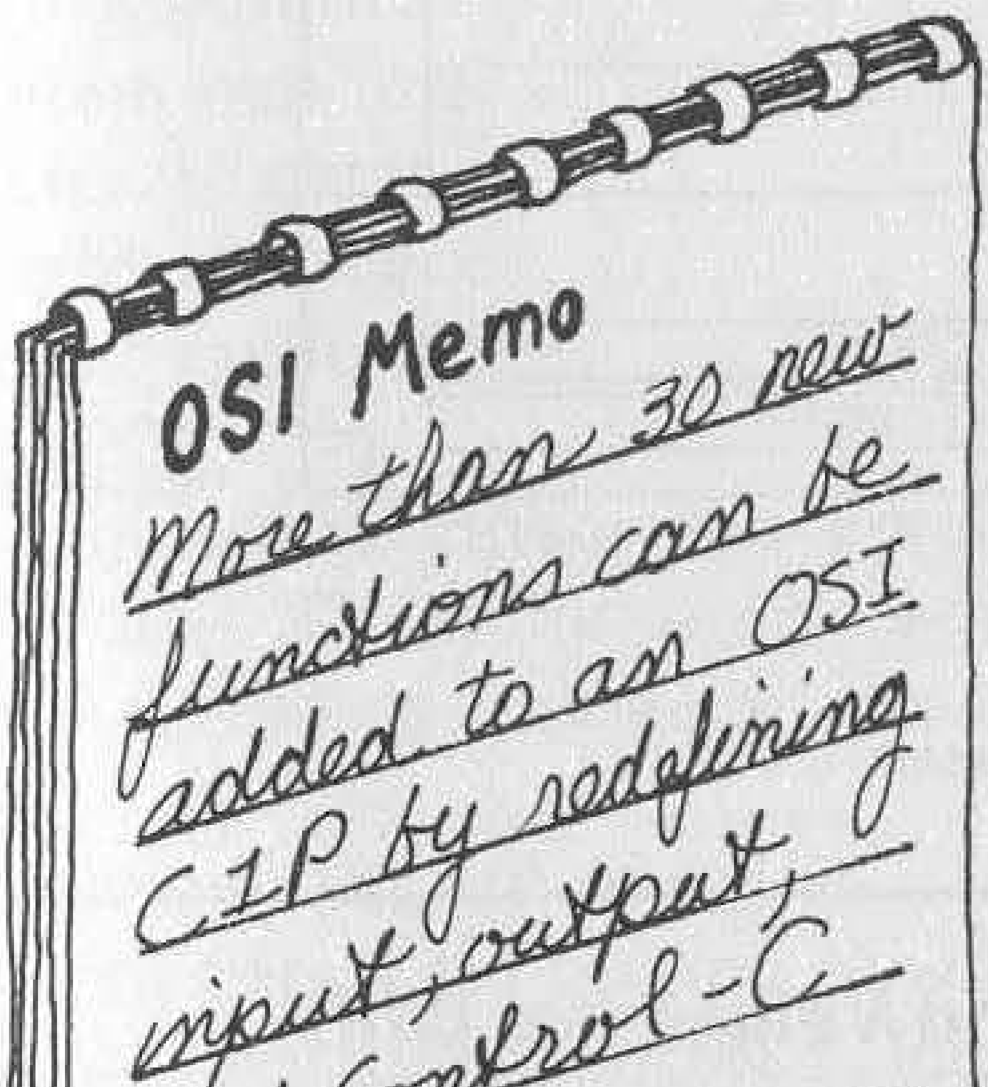
If you have trouble with errors, you will have to adjust R57, the input filter center-frequency adjustment. This pot affects only the input circuit; to adjust it, tape a program at 900 baud and then start loading it. For best results, adjust the volume and tone controls of the cassette deck first so the number of erroneous characters appearing in the listing is minimized. (On my cassette recorder, I set the volume at one-third and the tone control in the center of its range.) Next adjust R57 until you no longer see any errors. Continue turning the potentiometer until you start to get errors again. Now set it between the two settings where the errors start to occur and where they stop. Once the adjustment has been made at 900 baud, it will be correct for the two slower settings. Changing the setting of R57 does not affect the Kansas City receiver circuit significantly at 300 baud. Any old tapes that were made at 300 baud should still work, as will any tapes that you purchase.

Conclusion

Once R57 has been adjusted, the circuit is ready to use. On my system, I found the reliability at the two higher speeds virtually the same as that at 300 baud. Besides saving time and tape, I have made use of the higher rates for doing quick line searches while programming. If you want, you can change speeds while listing so you can find a certain part of your program quickly. This must, of course, be done while in the save mode.

Extended I/O Processor

by Michael J. Keryan

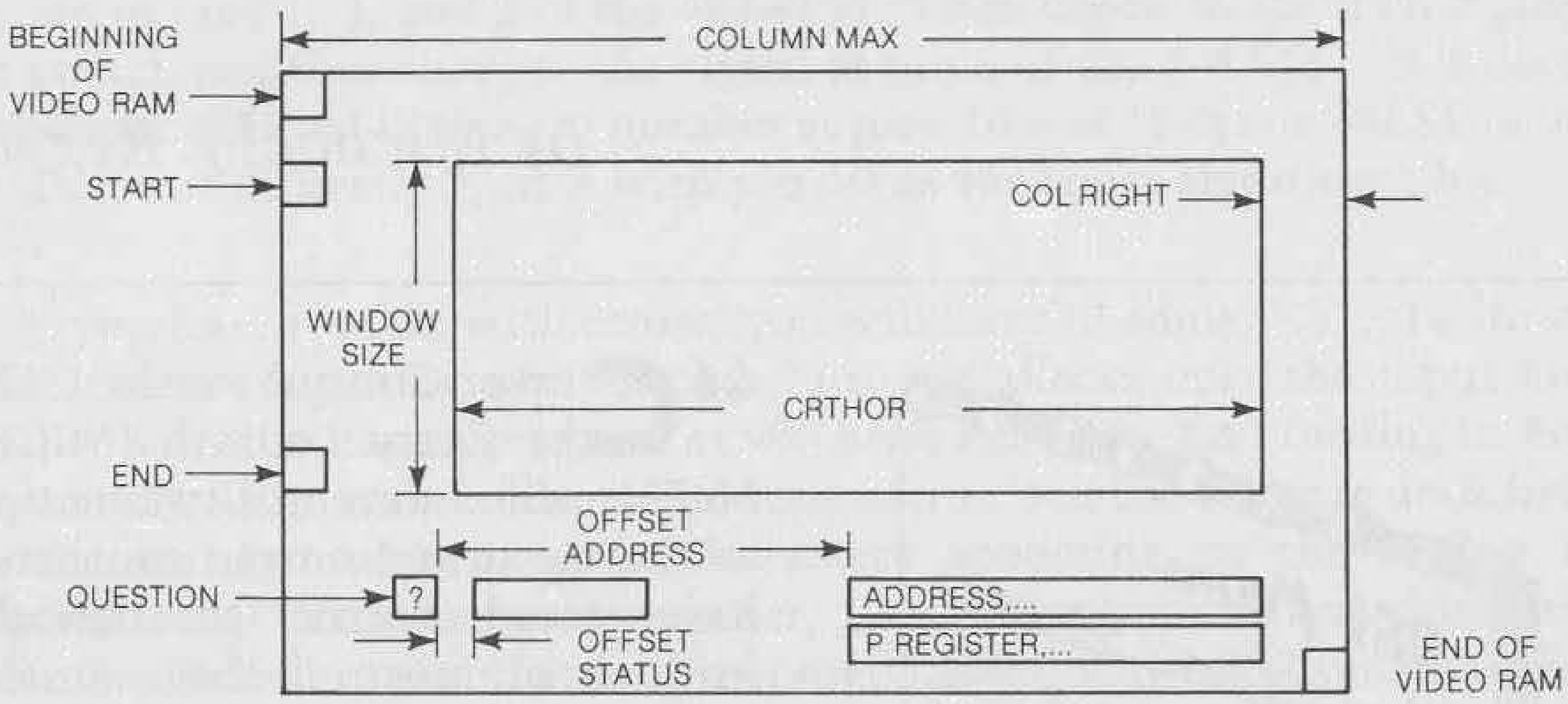


"Cursor Control for the C1P," by Kerry Lourash (MICRO 36:75), added nine utility functions to the input and output routines. I have pieced together the desirable features of most of these smaller programs and added a number of new ones, such as automatic line-number generation. In all, over thirty routines are now available for use during keyboard input, screen output, etc. User-supplied software/hardware additions for a printer, bell, and bug-free garbage collection are also supported. An improved monitor program is included, which can be called at any time. All the constants — screen parameters, subroutine vectors, and flags — were put into tables rather than imbedded into machine code, making changes relatively easy. The program originally was written for a C2-8P, but the version described here is for a C1P with 8K of memory. The 2K program is ROMable, assuming all the references to the high byte of subroutines ($18 through $1F) are translated to higher memory.

The Video Screen

Several screen parameters are stored in page zero memory, as shown in figure 1 and table 1. There are no restrictions on screen size or video memory location; 32, 64, or non-standard line widths can be supported, as well as video memory at locations other than $Dxxx. Figure 1 shows the window starting near the top of the screen and the flags and monitor fields (described later) near the bottom, but all locations can be modified. During initialization, the parameters are copied from tables within the program (default locations) to lower memory. The parameters can be changed by POKEing into pages zero and two, but the default values will be re-established on each warm start. Therefore, if the default values do not suit you, change them in the upper memory tables.

Figure 1: CRT Screen Parameters



COLMAX − 1 = COLUMN MAX − 1
CRTHOR + 1 = CRTHOR + 1
COL L − R = (COLUMN MAX) − (CRTHOR) − (COL RIGHT) + 2
OFFSET − P REGISTER = COLUMN MAX − 14

Table 1: Parameter Location and Values (for C1P)

PARAMETER	LOCATION	DEFAULT LOCATION	DEFAULT VALUE
CURSOR-LO	$00E0	$1C00	A0
CURSOR-HI	$00E1	$1C01	D0
START1-LO	$00E2	$1C02	A0
START1-HI	$00E3	$1C03	D0
END1-LO	$00E4	$1C04	C0
END1-HI	$00E5	$1C05	D2
QUESTION-LO	$00E6	$1C06	C5
QUESTION-HI	$00E7	$1C07	D3
OK SYMBOL	$00E8	$1C08	E5
CURSOR SYMBOL	$00E9	$1C09	A4
COLUMN MAX	$00EA	$1C0A	20
COLMAX-1	$00EB	$1C0B	1F
COL L-R	$00EC	$1C0C	07
COL RIGHT	$00ED	$1C0D	01
CRTHOR+1	$00EE	$1C0E	1A
STATUS FLAGS	$00EF	$1C0F	82
CONTROL C FLAG	$0212	$1C36	00
AUTOLINE-LO	$02D0	$1C10	90
AUTOLINE-HI	$02D1	$1C11	00
AUTOLINE INCREMENT	$02D2	$1C12	10
LINES/PAGE-CRT	$02D3	$1C13	12
LINES/PAGE-PRINTER	$02D4	$1C14	30
START2-LO	$02D6	$1C16	01
START2-HI	$02D7	$1C17	D3
END2-LO	$02D8	$1C18	80
END2-HI	$02D9	$1C19	D3
MOVE CURSOR-LO	$02DA	$1C1A	A5
MOVE CURSOR-HI	$02DB	$1C1B	D0
OFFSET-STATUS	$02DC	$1C1C	00
OFFSET-ADDRESS	$02DD	$1C1D	08
OFFSET-P REGISTER	$02DE	$1C1E	12

Cursor Movement

The cursor position is stored in locations $00E0 (low byte) and $00E1 (high byte). The cursor-movement functions print the character under the cursor, move the cursor, and print the cursor symbol (stored in location $00E9) at the new position. No other output to the CRT or printer is affected. The following control characters will cause non-destructive cursor movement to any screen location:

Up one line	Control-U	($15)
Down one line	Control-D	($04)
Left one space	Control-L	($0C)
Right one space	Control-R	($12)
Right eight spaces	Control-I	($09)

Using these cursor movements can put the cursor outside an active window. The following movement controls keep the cursor within an active window:

Return to the left of a line	Control-Q	($11)
Home cursor (to bottom of window)	Control-B	($02)
Backspace (like Control-L, but stays in margins)	Control-H	($08)
Move cursor (to a preset location)	Control-N	($0E)

Control-N moves the cursor to the location stored in $02DA (low byte) and $02DB (high byte). It is now set for the top left corner of the screen. Note that if the preset location is outside the window, Control-N causes the cursor to leave the window.

Window Controls

Active window boundaries are stored in START: $00E2, $00E3, and END: $00E4, $00E5. All CRT output, scrolling, etc., is maintained within these boundaries. An alternate window is stored in START2: $02D6, $02D7, and END2: $02D8, $02D9. The two windows could be equivalent, partially overlapping, or completely separate. The two windows can be switched by pressing Control-W ($17). In addition to toggling the windows, the cursor is homed in the new active window.

The window boundaries can be changed by POKEing into the appropriate locations, but are easily changed by using the Control-X ($18) key. To use Control-X, first place the cursor anywhere on the desired line by using Control-U or Control-D, then press Control-X. You will be prompted for another key with a question mark (at location $00E6, $00E7) and a beep (if this function is implemented), until either a T (for top of window) or a B (for bottom) is pressed. Control-X will change only boundaries of the active window; to change the other window's boundaries, first use Control-W.

If the cursor is placed above the window, it will naturally move down into (and be trapped in) the window. If the cursor is placed below the bottom boundary, however, it will not move by itself from that line. This can be used for a one-line non-scrolling window, but a two-line window is the minimum required to give readable text.

Scroll Controls

If the cursor is placed near the top of the window, it will move down the screen as lines of text are output. No scrolling occurs until the cursor attempts to move down when at the bottom of the window; the whole window then scrolls upward and the home line is blanked. An upward scroll can be forced at any time by pressing Control-Y ($19); similarly, a downward scroll is forced by Control-Z ($1A). These functions control only the location of the text, which is moved up or down on the screen; they do not move the cursor, which remains stationary. The scrolling functions are useful in editing and in game programs.

Clear Controls

To erase the entire screen, press either Control-T ($14) or ESCAPE ($1B). To erase only the active window, press RUBOUT ($7F); this also homes the cursor in the window.

Edit Text

Text can be entered by typing it in as usual, or by placing the cursor anywhere on the screen and pressing Control-E ($05). This causes whatever is under the cursor to be entered into BASIC; it has the same effect as typing the character. The cursor is then indexed one space to the right.

When entering a line of text, characters can be deleted with shift O ($5F); this moves the cursor one space backwards, deletes the character from BASIC, and erases it from the CRT. The function of shift P ($64) is not changed; it scratches from BASIC the line being worked on, but does not erase the line from the CRT.

To summarize, text is entered by typing characters (or spaces) or by using Control-E over text. Text can be deleted by typing spaces over text when using Control-E or with shift O. Text is not changed by using cursor controls; these are used only to position the cursor to allow use of a combination of Control-E, character input, or space input.

Autoline

To facilitate easy entry of text, an automatic line-entering system can be invoked by inputting Control-A ($01). Control-A toggles the autoline mode off or on at any time. Also it can be changed by POKEing the status flag. When the autoline mode is on, an A appears near the bottom of the screen. Then you enter a carriage return to activate autoline.

When the system is initialized, the starting line number is 100 and the increment is 10, resulting in lines numbered 100, 110, 120, ..., 9990. The line number and increment can be changed at any time by POKEing locations $02D0 and $02D1 (line number) and $02D2 (increment). These are packed BCD numbers, four bits per digit. The default values are re-established on warm start.

When the autoline mode is on, the input routine looks at both the character being entered and the last character. If the last character was a carriage return, you are now at the beginning of a new line, possibly in need of a new line number. Entering any character other than a space, a control character, a number from 0-9, a shift-O, or a rubout, automatically generates a new line number before the key is entered. These exceptions allow certain things to be done without getting a line number put on it: immediate mode commands are invoked by first typing a space, then the command; new line numbers can be inserted between or over existing lines; and all cursor and editing commands can be used. The autoline mode can be toggled off by using Control-A.

Flag Changes

To change a status flag, use Control-F ($06). You then get a prompt. You must enter the flag number (from 1 to 8), followed by either a 0 (for off) or 1 (for on). The flag code numbers are:

Flag Number	Code	Description
1	H	Hard copy (printer) mode
2	C	CRT output mode
3	I	Intermittent output (paging) mode
4	T	Trace mode
5	S	Step mode
6	A	Autoline mode
7	M	Monitor save mode
8	E	Extended I/O mode (all functions)

After the flag number and status is entered, the status of all flags are displayed near the bottom of the screen (these can be erased by escape or Control-T). The status can also be changed at any time (e.g., during execution of a BASIC program) by POKEing bits into location $00EF; the flag number corresponds to the bit number. Note that if the E flag is cleared, you can get back into the extended I/O mode by POKEing a number greater or equal to 128 ($80) into $00EF, or a warm start.

CRT and Hardcopy Flags

When these flags are set to 1, a corresponding output to the screen or printer is created. These flags are independent. To get printed output, a user-supplied printer subroutine must be included: change the NOPs at $1EF7 to JSR $YYXX (20 XX YY), where $YYXX is the address of your subroutine. Prior to this subroutine call, 16 page zero locations ($00EX) are freed for additional use by the print routine and are restored before returning to the CRT output.

Print Window

At any time, a Control-P ($10) from the keyboard causes the entire active window to be output to the printer, character by character. The H flag need not be set. The CRT display is not affected.

Intermittent Output

If the I flag is set, the number of lines output to the CRT/printer are counted and stored in locations $02F6/$02D5. These are compared to constants stored in locations $02D3/$02D4. If the line count is equal to the preset page size, the computer prompts you and waits for a keyboard entry before continuing. This allows you to copy (or read) CRT text before it scrolls off, or change to a new sheet of paper on the printer. These counts are independent; both are reset to zero on warm start.

Stop/Restart Output

In addition to the above intermittent output mode, a program or listing can be stopped at any time by pressing Control-S ($13) and then restarted by Control-R ($12). These commands are functional only during output. In many cases, the Control-S/R sequence is preferred over Control-C/CONT since no extraneous output is printed.

Step and Trace Modes

If the Step mode is invoked by setting the S flag, only one line of BASIC code is executed during RUN. You are then prompted for a keyboard entry, after which the next line is executed, and so on.

If the Trace mode is invoked by setting the T flag, the BASIC line number is printed when that line is executed. The output is then a mixture of line numbers with the normal program output. The program cannot be LISTed while in T mode.

The Step and Trace modes are independent, but for most purposes, are used together for debugging programs. The Control-C flag (at location $0212) must be cleared (enabled) to activate either Step or Trace: this is done on warm start.

View Tape

Pressing Control-V ($16) causes entry into the cassette-view mode, where BASIC tapes can be read and displayed on the CRT, but are not entered. To exit this mode, enter a space. This routine uses the old I/O vectors to eliminate accidental control-character routine activation during viewing.

Bell

An audible prompt is used in several of the above routines. This bell function is also used when a Control-G ($07) is either input or output. $07 is output if you attempt to enter more than 71 characters on a line. As an additional feature, the bell is also sounded once after the 64th character, like a typewriter, to warn you that the end of the line is near. To use the Bell feature, you must supply a subroutine at location $1CEC and the appropriate hardware modifications. (See MICRO 38:65, "A Typewriter Bell for Your Microcomputer.")

Carriage Return on BASIC Input

With OSI computers, if you respond to an input statement with only a carriage return, you will be kicked out of your program into the immediate mode. Usually you can jump back in with a CONT statement, but this is frustrating. On most large computers such a response is legal. This feature has been added to the input routine. A carriage return is accepted as a zero for numeric inputs, such as INPUT A, or as a space ($20) for string inputs, such as INPUT A$.

Other Jumps

An input of $1D causes a jump to the menu ($FF00). This duplicates the function of the Break (Reset) key and makes it easy to jump there from inside a BASIC program. Inputs of $1C, $1E, or $1F are not used. You can add your own functions by adding your vectors to the tables located at $1800-$183F.

Escape Sequence on Output

Most of the functions are accessed by entering a control character ($01-$1F) from the keyboard, either in immediate mode or in response to an INPUT statement. These functions also can be accessed on output, either in immediate mode or by a BASIC program. An escape sequence is used. The escape code ($1B, decimal 27) is output, followed by the control code. For example, to toggle windows, execute:

```
PRINT CHR$(27);CHR$(23);
```

The last semicolon is used to keep the display from scrolling. To output the graphic character for $1B, output two consecutive escapes:

```
PRINT CHR$(27);CHR$(27);
```

Not all functions are suitable for use during a BASIC run but many are, including cursor movements, scrolling, window toggles, screen clear, bell, print, etc. A summary of control functions is shown in table 2.

Table 2: Summary of Control Key Functions

CONTROL KEY	HEX	DECIMAL	FUNCTION	LOCATION
-	00	0	NONE-NULL	$185C
A	01	1	AUTOLINE TOGGLE	$1DF5
B	02	2	BOTTOM CURSOR (HOME)	$19F7
C	03	3	NONE-CONT C	$185C
D	04	4	DOWN CURSOR	$193D
E	05	5	EDIT	$187E
F	06	6	FLAG CHANGE	$1DD0
G	07	7	BELL	$1CEC
H	08	8	BACKSPACE CURSOR	$1905
I	09	9	INCREMENT CURSOR 8 SPACES	$1924
J	0A	10	NONE-LINE FEED	$185C
K	0B	11	MONITOR	$1A48
L	0C	12	LEFT CURSOR	$18F2
M	0D	13	NONE-CARR. RETURN	$185C
N	0E	14	MOVE CURSOR	$1946
O	0F	15	NONE-CONT O	$185C
P	10	16	PRINT WINDOW	$1EA3
Q	11	17	RETURN CURSOR	$1919
R	12	18	RIGHT CURSOR / RESTART	$190E
S	13	19	STOP OUTPUT	$185C
T	14	20	CLEAR SCREEN	$1CA0
U	15	21	UP CURSOR	$192D
V	16	22	VIEW TAPE	$1885
W	17	23	WINDOW TOGGLE	$18A8
X	18	24	SET WINDOW	$18BC
Y	19	25	SCROLL UP	$1894
Z	1A	26	SCROLL DOWN	$189E
ESC	1B	27	CLEAR SCREEN	$1CA0
-	1C	28	--	$185C
-	1D	29	JMP TO $FF00 (MENU)	$1C7A
-	1E	30	--	$185C
-	1F	31	--	$185C

New Monitor

An improved machine-language monitor routine is accessed by inputting Control-K ($0B). This monitor is significantly better than OSI's minimal monitor but not as versatile as commercial monitors. The advantage of this monitor is that it can be called at any time — in immediate mode, in the middle of a BASIC program, or by a JSR machine-language call.

Once the monitor is entered, data appears at the bottom of the screen, as shown in figure 1. The screen locations of this data are set by constants stored at $00E6 (low byte) and $00E7 (high byte), and offsets $02DD and $02DE. There are eight fields shown:

L — Location (four character address)
H — Hexadecimal data stored in L
C — ASCII character stored in L
S — Stack pointer
P — Processor status register (flags)
A — Accumulator
X — X register
Y — Y register

The "cursor" in the monitor mode is controlled by the keys "," and ".". These keys were chosen because the symbols for the left arrow and right arrow appear on these keys. The "," moves the cursor left, the "." moves it right. The cursor actually changes the lower-case letters l, h, c, etc., to the upper-case letter to be changed. Any field is changed by typing new data into it. The C field allows any character (except "," and ".") to be entered; the other seven fields allow only hexadecimal (0-9, A-F) characters.

Machine-language programs thus can be entered, or memory reviewed or changed, one byte at a time. The space bar is used to step forward through memory; the carriage return key is used to step backwards. Type R to return to where you were before you entered the monitor.

To jump to a subroutine (whose location is shown in L) type J; if the subroutine executes correctly and is terminated by an RTS ($60), control returns to the monitor. All flags and registers (S, P, A, X, and Y) are changed to what was shown on the screen just before the jump occurred. When returning to the monitor, the contents of S, P, A, X, and Y shown on the screen reflects their status at the time of return. No provisions are made for single step, trace, trap, etc.

When the monitor mode is entered, several things happen; all flags and registers are saved, and the P field is initialized to $04 (ignore interrupts and clear decimal mode). The S field is adjusted to prevent change to the stack. If the P register is changed, it will be restored automatically on return. However, if the stack is disturbed, you may run into problems when returning, unless the original page one ($01XX) was saved. If the M flag of $00EF is set, the first three pages of memory — page zero (BASIC constants and routines), page one (the stack), and page two (BASIC and Extended I/O constants) — are saved in the top three quarters of screen memory ($D000-$D2FF). This allows you to use these lower memory locations for your machine-language programs. They will be restored from the screen memory when exiting the monitor mode (R). If the M flag is clear, these three pages are not saved. Leave the M flag cleared if you merely want to examine or change a few memory locations or if you don't want the screen display disturbed.

Garbage Collector

A bug in OSI's BASIC-in-ROM may cause your program to bomb if you make extensive use of dimensioned strings. Provisions have been made to allow you to add a foolproof machine-language garbage-collection routine. This routine is called through the revised Control-C routine if fewer than 512 bytes of free memory are available; this keeps OSI's defective routine from being called. To use this function, insert $20 XX YY at $1D72, where $YYXX is the location of your new garbage-collection routine. In addition, the approximate number of free pages can be monitored at any time by PEEKing at $02F8. This can be used in lieu of FRE(X); never call FRE(X) when using dimensioned strings, as this forces a fatal garbage collection by the defective routine.

Initialization

First cold start, then Break-M, load the tape containing the Extended I/O routines, Break-M, then type .1D1FG. The initialization routine will then be run. The input, output, and Control-C vectors are pointed to new routines. The warm start and OK routines are replaced by new ones. Tables are copied from within the program to page zero and page two, where they are used by the new routines. The memory size is adjusted to keep BASIC from overwriting the new routines. The stack is adjusted to prevent an OM error after a warm start, then a message is written to the screen.

Odds and Ends

A subroutine that decodes a byte into two ASCII characters is located at $1CF7. Place the byte to be decoded into $0055. A JSR $1CF7 leaves the high-nibble character in $0053, the low one in $0054. An example of this routine is shown in listing 1. The simple program generated the hexadecimal dumps of table 3. Lines 100 and 200 turned the printer on and off. Line 160 set the USR vector to $1CF7.

A dump of the entire 2K program is shown in table 3; the underlined bytes are those that require changing if the program is relocated. Here are the locations that require changing if your OSI computer is not a C1P:

Location	**Function**	**C1P Location (low, high)**
$1C7E	Old Output Routine	69FF
$1C81	Old Output + 3	6C FF
$1C84	Old Input	BA FF
$1D5D	Old Control-C Routine	9B FF

However, you *must* have a support ROM (or EPROM) containing indirect vectors for these routines, which vector through page two of memory.

The control keys can be redefined any way you see fit by changing the pointers shown in table 2; these are stored at the beginning of the program ($1800-$183F). You may want to eliminate some functions (such as printer routines) and add others. You may want to let some keys generate predefined strings that can be entered into BASIC, such as DATA, or FOR I=1TO, etc. For hints on how to do this, study the autoline code. You may want to make some changes. I have yet to use a program that didn't need a few alterations.

Table 3: Hex Dump of Complete Program

```
$1800

5C F5 57 5C 3D 7E D0 EC 05 24 5C 48 F2 5C 46 5C
A3 19 0E 5C A0 2D 85 A8 BC 94 9E A0 5C 7A 5C 5C
18 1D 19 18 19 18 1D 1C 19 19 18 1A 18 18 19 18
1E 19 19 18 1C 19 18 18 18 18 18 1C 18 1C 18 18
A9 AD 8D 07 02 A9 8D 8D 0A 02 A9 60 8D 0D 02 A5
E3 8D 09 02 8D 0C 02 A5 E2 8D 0B 02 60 A9 20 48
20 40 18 A4 E4 A6 E5 68 20 0A 02 EE 0B 02 D0 03
EE 0C 02 CC 0B 02 D0 F0 EC 0C 02 D0 EB 60 AD 01
02 8D 02 02 60 20 F4 FF 20 83 1C 20 7D 1C AD 03
02 D0 F5 60 20 94 19 20 A0 19 20 89 19 60 20 94
19 20 52 1E 20 89 19 60 A2 03 B5 E2 48 BD D6 02
95 E2 68 9D D6 02 CA 10 F1 4C 57 19 20 6C 19 48
A5 E1 48 20 D9 1C C9 54 D0 09 68 85 E3 68 85 E2
18 90 0A C9 42 D0 EC 68 85 E5 68 85 E4 60 20 5D
18 20 CF 19 A5 E5 85 E1 A5 E4 20 6E 19 85 E0 18
90 61 20 94 19 A5 EC 48 A9 00 85 EC 20 7B 19 68
85 EC 18 90 4E 20 94 19 20 74 19 18 90 45 20 94
19 E6 E0 D0 3E E6 E1 D0 3A 20 94 19 20 6C 19 85
E0 18 90 2F A2 08 20 0E 19 CA D0 FA 60 20 94 19
A5 E0 38 E5 EA 85 E0 B0 1A C6 E1 D0 16 20 94 19
20 5D 19 18 90 0D 20 94 19 AD DA 02 85 E0 AD DB
02 85 E1 20 89 19 60 20 94 19 4C E4 18 A5 E0 18
65 EA 85 E0 90 02 E6 E1 EA EA EA 60 A5 E0 05 EB
38 E5 EE 60 20 6C 19 C5 E0 D0 0B A5 E0 38 E5 EC
85 E0 B0 02 C6 E1 C6 E0 60 A0 00 B1 E0 8D 01 02
A5 E9 D0 03 AD 01 02 A0 00 91 E0 60 A5 E8 D0 F7
20 40 18 18 65 EA 90 03 EE 09 02 8D 08 02 A6 E4
A4 E5 20 07 02 EE 08 02 D0 03 EE 09 02 EE 0B 02
D0 03 EE 0C 02 EC 0B 02 D0 E8 CC 0C 02 D0 E3 A4
EB A9 20 91 E4 88 10 FB 60 AD 02 02 C9 20 B0 F8
AA BD 00 18 8D F1 02 BD 20 18 8D F2 02 6C F1 02
EE D5 02 EA EA EA 18 90 1B A9 20 20 97 19 20 2D
```

(continued)

Table 3 *(continued)*

```
$1A00

1A 18 90 10 C6 0E A9 20 8D 01 02 20 97 19 20 74
19 20 90 19 4C F6 1F EA EA EA 8D 01 02 20 97 19
E6 E0 A5 E0 05 EB 38 E5 ED C5 E0 D0 17 EE F6 02
20 6C 19 85 E0 C5 E4 A5 E1 E5 E5 B0 04 4C 40 19
EA 20 A0 19 20 89 19 60 08 48 8A 48 98 48 BA CA
CA 86 43 24 EF 50 1C A2 00 B5 00 9D 00 D0 CA D0
F8 BD 00 01 9D 00 D1 CA D0 F7 BD 00 02 9D 00 D2
CA D0 F7 A5 E6 18 6D DD 02 85 49 A2 00 8A 20 98
1A A2 04 86 44 6D DE 02 20 98 1A A5 E7 85 4A 20
A8 1A 20 22 1B 18 90 F7 95 4B 69 06 95 4C 69 04
95 4D 69 04 95 4E 60 EA A2 07 B4 4B BD 1A 1B 91
49 CA 10 F6 A6 42 B4 4B BD 1A 1B 29 DF 91 49 A2
07 B5 40 20 FE 1A B4 4B 20 0F 1B CA E0 02 D0 F1
A0 00 B1 40 A4 4D C8 91 49 20 FE 1A A4 4C 20 0F
1B A5 41 20 FE 1A A4 4B 20 0F 1B A5 40 20 FE 1A
20 0F 1B 60 29 0F F8 18 69 90 69 40 D8 60 48 4A
4A 4A 4A 20 F4 1A 85 53 68 20 F4 1A 85 54 60 C8
A5 53 91 49 C8 A5 54 91 49 60 6C 68 63 73 70 61
78 79 20 83 1C C9 2C D0 09 C6 42 10 04 A9 07 85
42 60 C9 2E D0 0D E6 42 A6 42 E0 08 90 04 A9 00
85 42 60 A6 42 E0 02 D0 05 A2 00 81 40 60 C9 0D
D0 09 A5 40 D0 02 C6 41 C6 40 60 C9 20 D0 07 E6
40 D0 02 E6 41 60 C9 4A D0 27 BA 86 48 A6 43 9A
A6 46 A4 47 A5 44 48 A5 45 28 20 8E 1B 08 85 45
68 85 44 84 47 86 46 BA 86 43 A6 48 9A 60 6C 40
00 C9 52 D0 2E 68 68 AD 00 D0 C9 4C D0 1E EA EA
A2 00 BD 00 D0 95 00 CA D0 F8 BD 00 D1 9D 00 01
CA D0 F7 BD 00 D2 9D 00 02 CA D0 F7 68 A8 68 AA
68 28 60 48 20 93 FE 10 02 68 60 A8 68 98 A6 42
D0 0E A2 03 06 40 26 41 CA 10 F9 05 40 85 40 60
E0 01 D0 0F 85 48 A2 00 A1 40 0A 0A 0A 0A 05 48
81 40 60 16 40 16 40 16 40 16 40 15 40 95 40 60
```

```
$1C00

A5 D0 A0 D0 C0 D2 C5 D3 E5 A4 20 1F 07 01 1A 82
90 00 10 12 30 00 01 D3 80 D3 A5 D0 00 08 12 25
A9 A2 A0 1D 20 C3 A8 60 EA 4C 1F 1D 4C 9A 1E FE
1E 5E 1F 42 1D A9 00 8D 12 02 A9 00 8D F6 02 A9
AA 8D F4 02 A2 0F BD 00 1C 95 E0 CA 10 F8 A2 0F
BD 10 1C 9D D0 02 CA 10 F7 A9 18 85 86 A9 00 8D
D5 02 EA EA A2 05 BD 29 1C 95 00 CA 10 F8 A2 05
BD 2F 1C 9D 18 02 CA 10 F7 60 4C 00 FF 4C 69 FF
4C 6C FF 4C BA FF EA EA 48 8A 48 A2 0F BD E0 02
48 B5 E0 9D E0 02 68 95 E0 CA 10 F1 68 AA 68 60
48 98 48 A0 00 A9 20 99 00 D7 99 00 D6 99 00 D5
99 00 D4 99 00 D3 99 00 D2 99 00 D1 99 00 D0 C8
D0 E5 68 A8 68 60 20 D0 1C C9 0A B0 F9 48 68 60
20 D9 1C 20 93 FE 30 F8 60 20 EC 1C A2 00 A9 3F
81 E6 20 83 1C 48 A9 20 81 E6 68 60 48 EA EA EA
EA EA EA EA EA 68 60 A5 55 4C FE 1A 48 20 16 1D
C9 BE D0 10 A9 F7 20 18 1D C9 7F D0 07 20 D9 1C
C9 12 D0 F9 68 60 A9 FE 8D 00 DF AD 00 DF 60 20
A0 1C AD F4 02 C9 AA F0 00 A9 00 8D D5 02 8D F6
02 EA EA EA 20 35 1C EA EA 20 20 1C A2 FE 9A 4C
74 A2 AD 12 02 D0 1E A5 EF 29 10 F0 03 20 D9 1C
A5 EF 29 08 F0 03 20 5A B9 20 66 1D 4C 9B FF EA
EA EA EA EA EA 60 A5 82 38 E5 80 8D F8 02 C9 02
B0 03 EA EA EA 60 A2 08 A9 01 8D F0 02 8A 18 6D
DC 02 A8 AD F0 02 25 EF D0 04 A9 20 D0 03 BD 99
1D 91 E6 0E F0 02 CA D0 E4 60 45 4D 41 53 54 49
43 48 45 78 74 65 6E 64 65 64 20 49 2F 4F 20 50
72 6F 63 65 73 73 6F 72 0D 0A 43 31 50 20 56 65
72 73 69 6F 6E 20 63 2E 4D 4B 20 31 39 38 31 00
20 C6 1C AA A9 01 CA F0 03 0A D0 FA 48 20 C6 1C
D0 0A 68 49 FF 25 EF 85 EF 18 90 05 68 05 EF 85
EF 20 76 1D 60 A9 20 45 EF 85 EF 4C 76 1D AD 02
```

(continued)

Table 3 *(continued)*

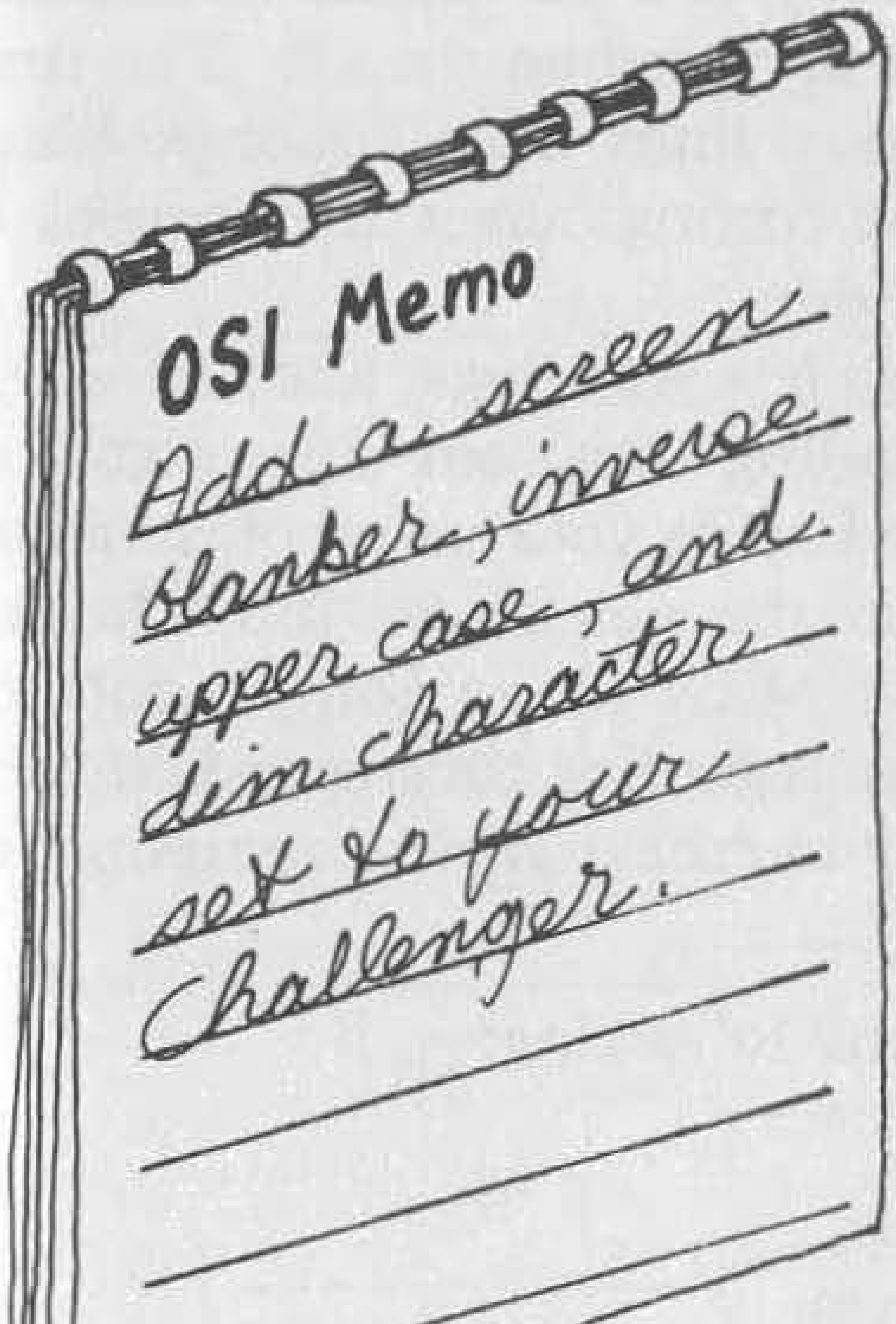
```
$1E00

02 C9 5F F0 44 C9 7F F0 40 C9 21 90 3C C9 3A B0
04 C9 30 B0 34 18 F8 AD D0 02 6D D2 02 8D D0 02
48 AD D1 02 69 00 8D D1 02 48 D8 A2 00 4A 4A 4A
4A 20 4A 1E 68 29 0F 20 4A 1E 68 48 4A 4A 4A 4A
20 4A 1E 68 29 0F 20 4A 1E 60 09 30 95 13 E8 4C
1A 1A 20 40 18 A5 E5 8D 09 02 8D 0C 02 A5 E4 05
EB 8D 0B 02 38 E5 EA B0 03 CE 09 02 8D 08 02 A6
E2 A4 E3 20 07 02 CE 08 02 D0 03 CE 09 02 CE 0B
02 D0 03 CE 0C 02 EC 0B 02 D0 E8 CC 0C 02 D0 E3
A4 EB A9 20 91 E2 88 10 FB 60 A9 0D 20 5E 1F 4C
9C 19 EA 48 8A 48 98 48 A5 E0 48 A5 E1 48 EA EA
EA EA A5 E2 85 E0 A5 E3 85 E1 A0 00 A9 0D 20 F4
1E A6 EA B1 E0 20 F4 1E E6 E0 D0 02 E6 E1 CA D0
F2 A9 0D 20 F4 1E A5 E1 C5 E5 90 E5 A5 E0 C5 E4
90 DF F0 DD EA EA EA EA 68 85 E1 68 85 E0 68 A8
68 AA 68 60 20 88 1C EA EA EA 20 88 1C 60 A5 EF
30 03 4C 83 1C 20 83 1C 8D 02 02 98 48 8A 48 A9
40 C5 0E D0 03 20 EC 1C A9 0D CD F3 02 D0 1B CD
02 02 D0 08 A9 20 20 4C 1E 18 90 09 A5 EF 29 20
F0 08 20 FE 1D 8A A8 68 98 48 AD 02 02 C9 7F D0
03 20 DE 18 20 D9 19 EA EA EA EA EA EA EA EA EA
AD 02 02 8D F3 02 68 AA 68 A8 AD 02 02 60 48 8D
02 02 A5 EF 30 04 68 4C 7D 1C 8A 48 98 48 20 FC
1C A5 EF 29 01 F0 1C A5 EF 29 04 F0 10 AD D5 02
CD D4 02 90 08 20 D9 1C A9 00 8D D5 02 AD 02 02
20 F4 1E A5 EF 29 02 F0 5F A5 EF 29 04 F0 10 AD
F6 02 CD D3 02 90 08 20 D9 1C A9 00 8D F6 02 AD
02 02 C9 0A D0 03 4C F0 19 C9 0D D0 03 4C F9 19
C9 5F D0 03 4C 04 1A C9 07 D0 03 20 EC 1C C9 1B
D0 11 AD F5 02 D0 07 A9 01 8D F5 02 D0 1A A9 00
8D F5 02 AD F5 02 F0 0A 20 D9 19 A9 00 8D F5 02
F0 06 AD 02 02 20 1A 1A 68 A8 68 AA 68 4C 80 1C
```

Listing 1: BASIC Program to Print a Hex Dump

```
100 POKE239,131
105 PRINT:PRINT"$1800"
110 FORI=6144TO6655STEP16
120 PRINT
130 FORJ=0TO15
140 K=I+J
150 A=PEEK(K):POKE85,A
160 POKE11,247:POKE12,28
170 X=USR(0):C=PEEK(83):D=PEEK(84)
180 PRINTCHR$(C);CHR$(D);" ";
190 NEXTJ,I
200 POKE239,130
```

Enhanced Video for C1P

by David Cantrell and Terry Terrance



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

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

First we will discuss OSI's video as implemented on the C1P. Even though you may have spent the past couple of years squinting at your C1P's screen almost daily, some of its subtleties may have escaped you.

When the screen is filled with CHR$(161) (OSI's solid white block character) and is viewed from about two feet away, all but the poorest TV or video monitor will show faint dark vertical lines on character-cell boundaries. You may have attributed these lines to a one-dot-wide inter-cell space. Closer inspection reveals that the whole screen is filled with evenly spaced dots — no blank spaces appear between cells. As the rows of dots of each character are clocked out of the shift register U42, the first dot in each row is held only one-third as long as the others in that row. Since this happens for the first dot of each row and for each character, the end result is faint dark bars when viewed from a distance. This is the subtle video defect alluded to earlier. It's so subtle that most OSIers do not notice it, or pass it off as intercell spacing. If C4 users are wondering why this effect can't be seen, the effect is reversed on the C4. The first dot is accentuated giving rise to bright vertical lines. This minor problem wouldn't be worth mentioning except the timing defect that causes it must be fixed if you are to add modified video.

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

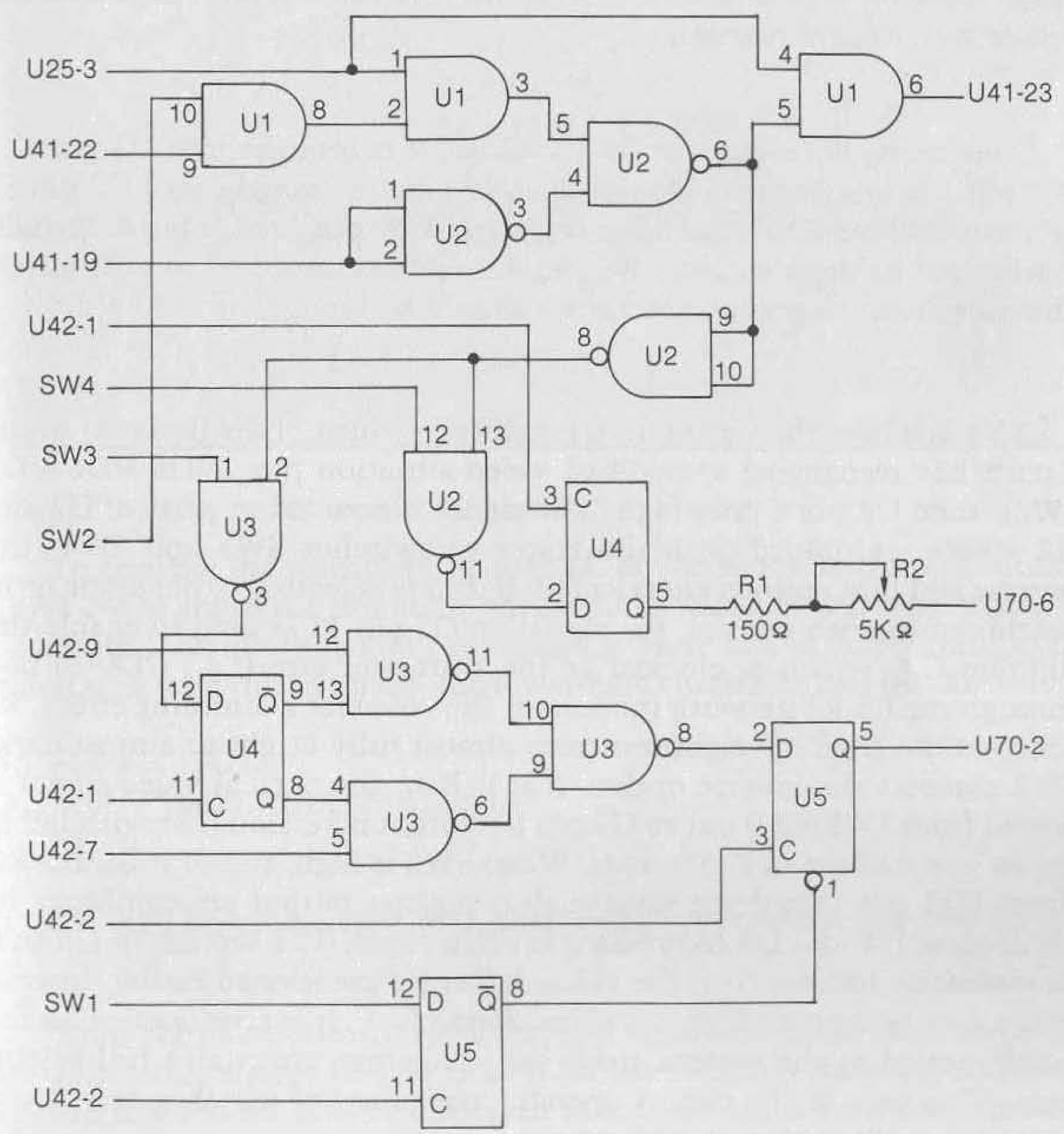
The parts list is short. You will need the following:

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

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

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

Figure 1: Schematic for Enhanced Video



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

We'll stop for a moment and explain what this part of the circuit does. U25 pin 3 is VD5 and U41 pin 22 is VD6, the data bits that the circuit keys on to know whether or not to output modified video. U41 pin 19 is VD7. Three gates of U1 and two gates of U2 perform logic to accomplish the following functions: if VD5 and VD6 are high and SW2 is high and VD7 is low, U1 pin 6 is low, causing lower-case characters to be read as upper case and activating the rest of the circuit *via* U2 pins 9 and 10; if either VD6 or VD5 is low or SW2 is low, U1 pin 6 will be high and the screen will behave normally.

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

This is where the circuit starts modifying video. If the first part of the circuit has recognized a modified video situation (i.e., VD5 VD6 VD7 SW2), then U2 pin 8 goes high. The signal is now fed to parts of U2 and U3 where, combined with the states of switches SW3 and SW4, the inverse and dim options are selected. If dim is selected, either alone or in combination with inverse, the signal on U2 pin 11 is used to enable the flip-flop U4, which is clocked at the shift-load rate (i.e., CLK/8) and through the R1-R2 network modulates the video for a dimming effect. R2 controls the level of brightness from almost fully bright to almost dark. SW3 controls the inverse option. If it is low, the normal video signal is passed from U42 pin 9 out to U5 pin 5 without inversion (but with latching as you will see in a moment). When SW3 is high, the shift-load clock (from U42 pin 1) and the inverse shift register output are combined by sections of U4 and U3 to produce inverse video. The section of U5 that immediately follows fixes the video defect we mentioned earlier. Instead of the dots being cut off by the video chain clock, it is now latched for the whole period of the system clock and, therefore, maintains full brightness. This part of the circuit operates regardless of whether or not any modified video options are selected.

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

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

Table 1: Operation of Switches SW1-SW4

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

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

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

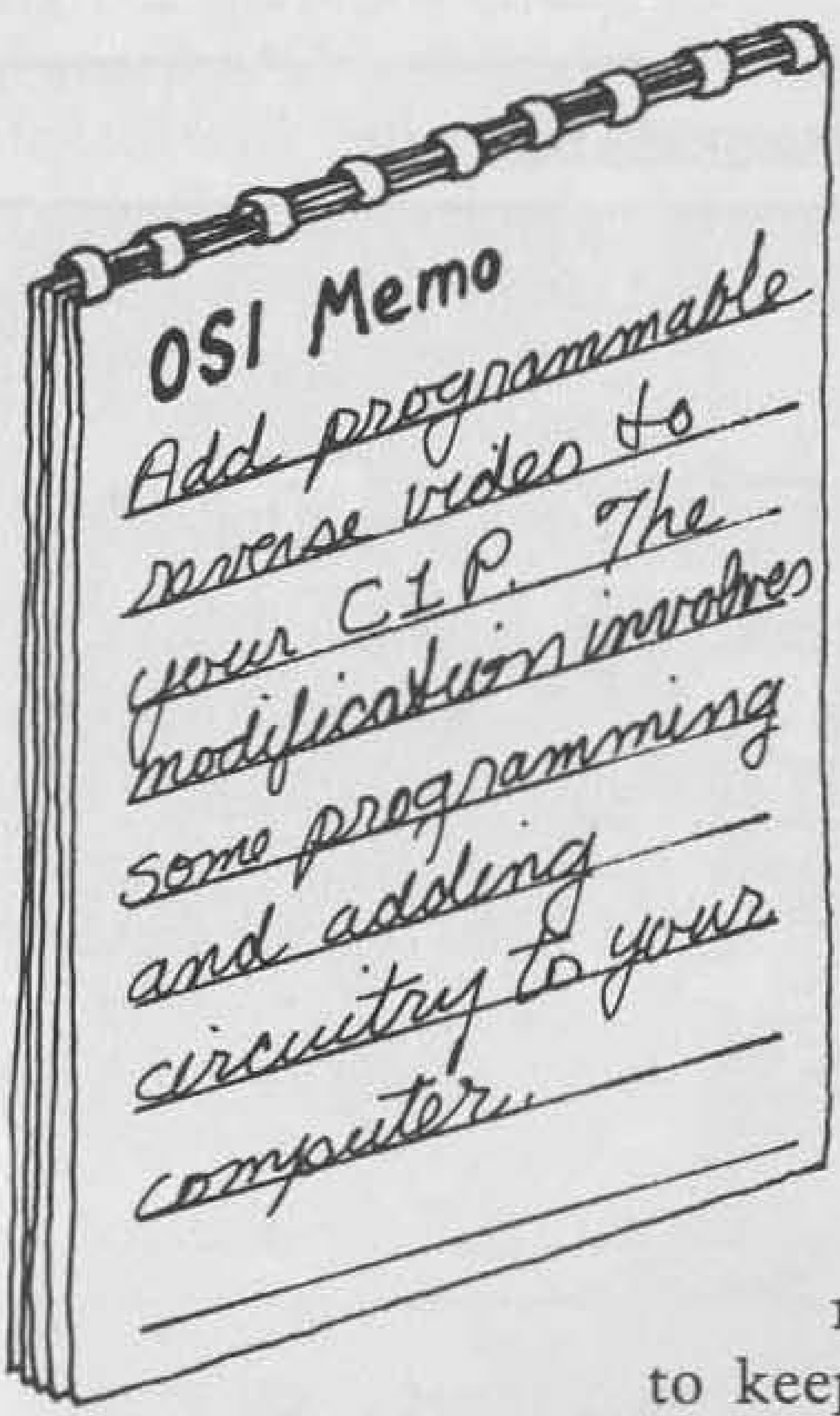
```
10 FORX=1TO12
20 PRINT"AaBbCcDdEeFfGgHhIiJj"
30 NEXT
```

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

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

Programmable Reverse Video

by Charles L. Stanford



The reverse video option requires modification to your C1P, some additional circuitry, and some software. You need above-average skills in electronic construction, as well as substantial programming ability to do this modification. While I've tried to make the actual changes on the main board as easy and risk-free as possible, it's still close to the equivalent of minor brain surgery on your best friend.

OSI's Video System

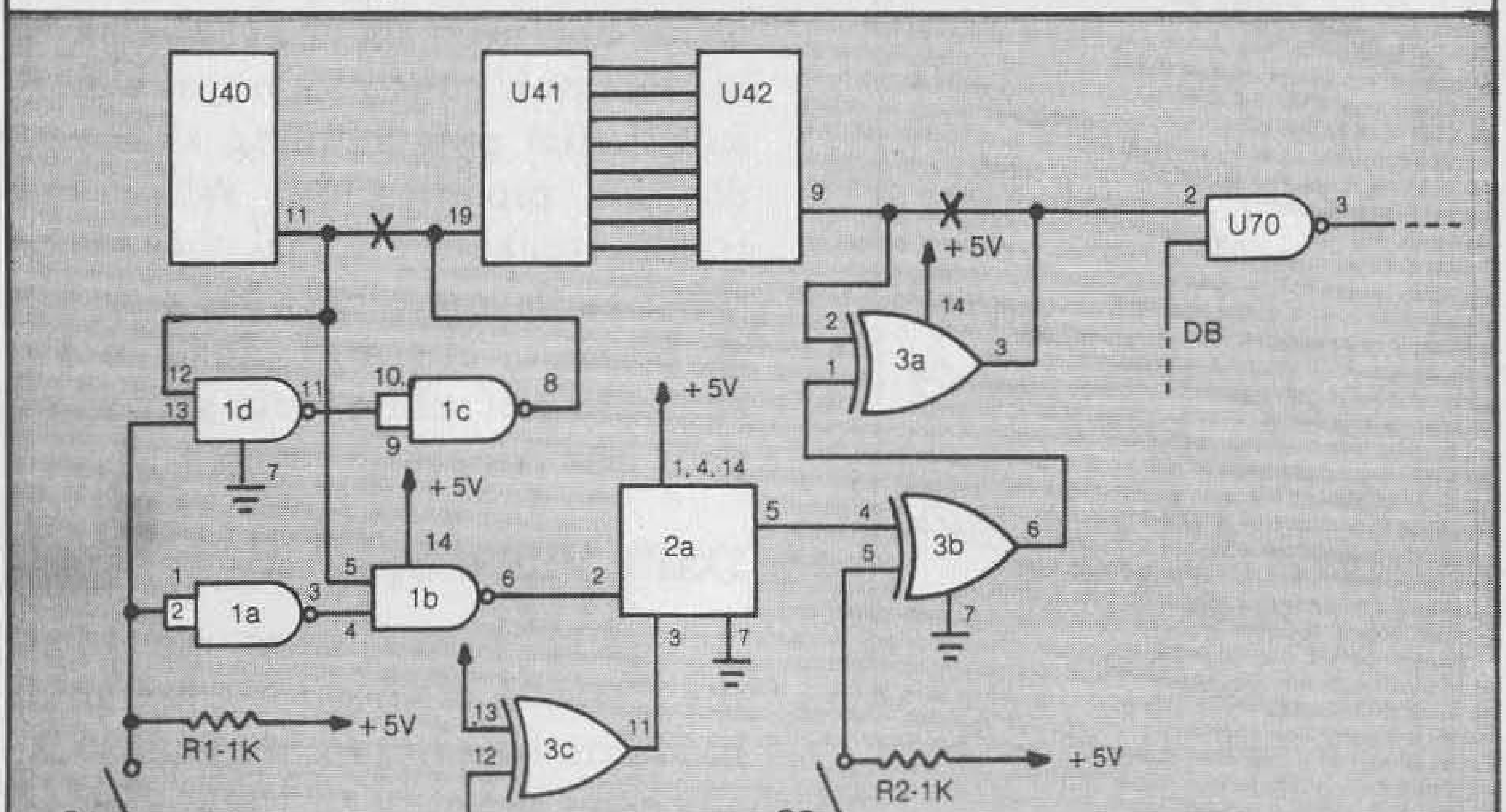
Unlike many other machines, the C1P video refresh is completely hardware-based. In other words, the microprocessor devotes no time or effort to keeping a proper display on the screen, but modifies the video RAM only when required to do so by the program. As a result, the video display has no undesirable streaks caused by software timesharing. You are, however, unable to make relatively simple program changes to achieve full control of the image.

Programmable Reverse Circuit Description

The circuit is relatively simple. It requires only three chips, can fit on a very small add-on board, and allows you to convert your computer back to its original hardware configuration almost instantly. It does cost a little in lost versatility: the upper 128 graphics characters are "lost" to use while the video reverse switch is closed. I have found that to be no inconvenience since the reverse video is generally used to enhance programs that employ alphanumerics only.

The add-on circuit consists of primarily three elements: the detecter, the latch, and the inverter. The detecter is connected, in series, with the most significant bit of the video data. As shown in figure 1, NAND gates 1b and 1d each detect the status of the bit. Treatment of the bit is also conditioned by the status of switch S1. IC1d either inverts it or ignores it; IC1b either detects it or ignores it. If S1a is open, the bit is passed along through IC1c and appears unchanged to character generator U41; also, IC1b ignores it and its output remains high. IC2a, half of a dual-D flip-flop, acts as a latch. It is clocked by the same latching signal used by U42, the parallel-serial shift register, and retains the status throughout the time needed to send one character to the screen.

Figure 1: Reverse Video Modification



The inverter uses two gates of a very versatile IC — the 7486 "exclusive OR" chip. In this circuit, it acts as both an inverter and a non-inverting gate. IC3a passes the serial video signal unchanged as long as pin 1 is held high, but pulling that pin low causes the signal to invert! In a similar manner, IC3b is used to condition the signal from the detecter and the latch circuits. Holding switch S2 high allows the signal from the latch to pass. Closing the switch inverts the output, effectively causing the image to be inverted constantly.

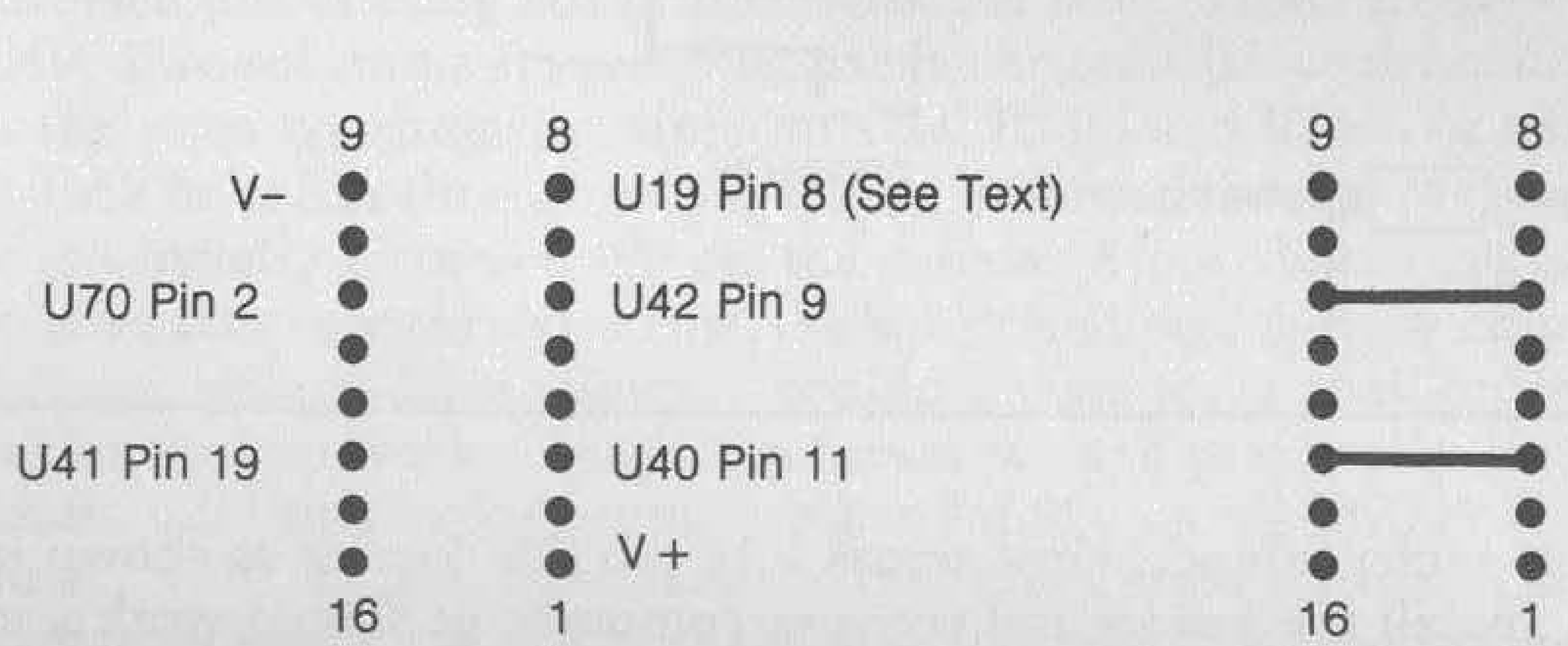
The net result of this circuit is to allow four conditions: when both switches are open, the computer acts normally; closing S1 inverts those characters that have a "1" in the left-most bit position (bit 7); closing S2 inverts the entire screen; and closing both causes the characters that have bit 7 high to be normal and the remainder to be inverted.

As I mentioned earlier, the price of this reverse video capability is the loss of the top 128 graphics characters. As long as switch S1 is open, the entire 256-character font of the character generator ROM is available. But closing that switch causes any character with a code greater than 127 ($7F) to detect the most significant bit and change it to low. Then the lower 128 graphics characters show up on the screen normally, and the upper half show up as their inverted complements. For example, POKEing the graphics character 51 ($33) to a screen location will cause the character "3" to appear. POKEing the character 179 ($B3) with switch S1 closed will cause an inverted "3" to show. Essentially, the top bit is checked, stripped off, and changed to "0". If the same sequence is performed with S1 open, the graphics character normally corresponding to 179 will appear.

Modifying the 600 Board

Since I am leery of damaging the PC board while making additions and modifications, I used an add-on board for this project. In addition, I devised a plug-in method that restores the main board almost instantly to its original configuration. Figure 1 shows the only two traces on the main board that need to be cut. These are marked by an "X". Then wires are run from either side of the cuts to prototype socket U28. By connecting the leads as shown in figure 2a, a properly jumpered DIP header can be used as a shunt in place of the plug from the add-on board, restoring normal operation.

Figure 2a: Socket U28 — Top View **Figure 2b: Jumper Header for U28 — Top View**



Start by installing a 16-pin soldertail IC socket at U28. Be sure to use a low-wattage pencil-type iron, and practice on an old board if you're rusty. Next cut the traces. You should use a jeweler's loupe or other magnifying lens and carefully scratch away about 1/8 inch of the trace with a sharp knife blade. Cut the line on the top of the board (component side) between U40 pin 11 and U41 pin 19. It starts at U40 but soon runs under U41's socket. Cut it about ¼ inch from pin 11 of U40.

Now find the trace that leaves U70 pin 2 and heads for the keyboard. It runs only one inch before passing through the board. (Remember the location of this plated-through hole. It is used later.) The trace now runs on the bottom toward the right and, again, passes through to the top. It runs from there toward the front again, ending at U42 pin 9. Cut the trace on the bottom of the board near the hole by U70.

Next connect the socket at U28. Using fine-gauge insulated wire, connect each pin as shown in figure 2. It's easier to connect U40 and U41 by slipping the wire down into the sockets at the proper pin than to try to solder to the small bit of PC board trace showing. If necessary, carefully remove the ICs. For the other jumpers, use the two holes where the trace passes to the bottom of the board for your wire connections. Note that a connection to U42 pin 1 is marked "see text." I suggest that you use figures 1 and 2 as they appear until the new display reveals timing problems serious enough to require the fourth IC shown in figure 3. So for now, hook U42 pin 1 (which also connects to U19 pin 8) to U28 pin 8. Connect the positive and negative buses to pins 1 and 9, respectively.

Figure 3a: Piggybacking ICs

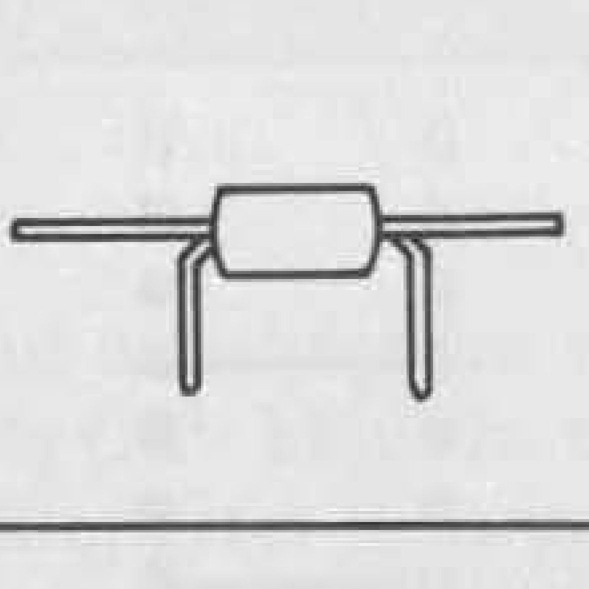
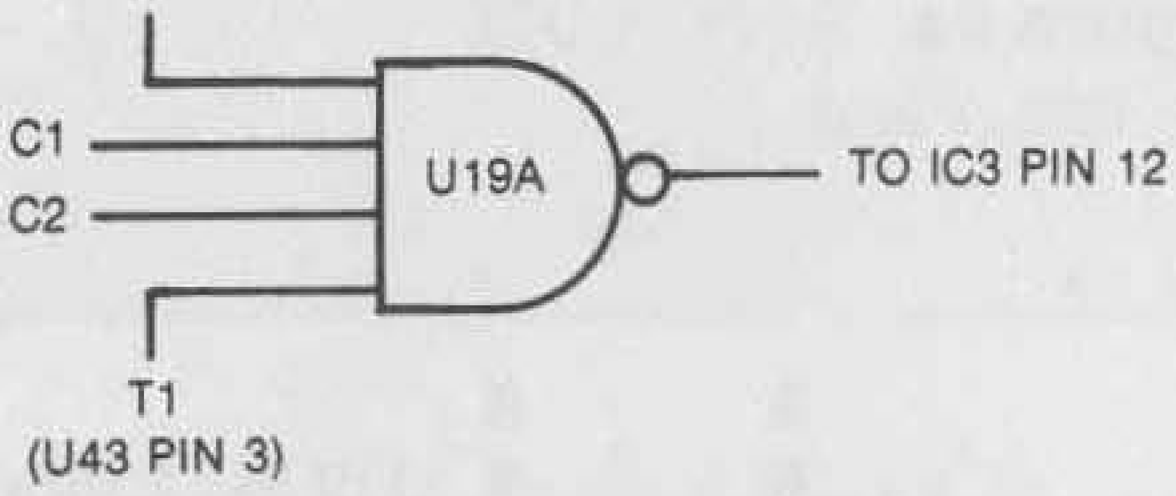


Figure 3b: Connections for Optional IC



Finally, solder jumper wires across a 16-pin DIP header as shown in figure 2b. Install the header and try your computer. It should work normally. If not, troubleshooting should be easy since you've made only minimal changes.

Building the PC Board

You can use one of several techniques to build your board. In this case, wirewrap is probably the best option. Equipment and supplies are readily available and easy to use. You must use a check list or schematic, and carefully check all connections when finished. Check the board under power, first *without* ICs and then *with* ICs, and measure current drain with a good volt/ohmmeter. Insert the ICs correctly. These TTL ICs will take a lot, but they cannot stand even a short period of inverse voltage, so make sure they are inserted properly.

The switch(es) can be mounted on your keyboard near either the left or right rear (just below the nameplate). When drilling, be careful not to mar the finish or get metallic cuttings in the works. Use stranded insulated wire to connect the small board with the switch and on the second IC header. You might want to use some sort of socket/plug in the leads to the switch if you expect to disassemble your machine often; this cuts down the stretching and bending of the wires.

Testing the Add-On

Warm up the TV or monitor before the computer is powered. Then, if the screen doesn't show a reasonable display, turn the power off immediately and check all wiring carefully. Using an ohmmeter, make sure every point is properly connected to, and *only* to, the other proper points. Since your machine will have been without power for some time, the RAM probably will be scrambled, and at least a few graphics characters will appear. Don't hit Break at this time; try the switches and get a feel for the way they work.

You should also look for timing problems now. Compare the reversed characters with the OSI *Graphics Reference Manual*. If the timing from U19 pin 8 is delayed too much by passing through ICs 2 and 3, the screen will reverse a bit late and change back a bit late. Reversal of characters in a row will be noticeable only at the beginning of the first row and the end of the last row. This phenomenon occurs when the signal from U42 is reversed just slightly out of sync with the latch trigger from NAND gate U19. Two solutions are possible: use faster gates (since the cause of delay is the extra transmission time in IC2a, IC3b, and IC3a); or use 74S-ICs (which have fast throughput) to reduce differential delay to the point that it is virtually unnoticeable on the screen. A few disadvantages to this modification are the extra cost, difficulty finding Schottky chips, and additional power drain. Since I couldn't wait for a mail-order delivery taking several weeks, another solution seemed practical — equalize the delay. I did this by installing another 74LS20 on top of U19 with all but pins 7, 9, 10, 12, and 14 bent out so they don't make contact. This is called "piggybacking" and is a neat and effective way to add additional circuits to an existing board.

Listing 1

```
10 PRINT "VIDEO REVERSE DEMO": PRINT: PRINT
30 INPUT "ENTER A STRING"; X$
40 A$=X$: GOSUB 190 :X$=A$: PRINT X$
60 INPUT "ENTER A NUMBER";X
70 A=X: GOSUB 170 :X$=A$: PRINT X$
90 END
160 REM --REVERSE NUMBERS--
170 A$=STR$( A)
180 REM --REVERSE STRINGS--
190 B$=""
200 FOR X= 1 TO LEN (A$)
210 C$=CHR$( ASC( MID$( A$,X,1)) OR 128)
220 B$=B$ + C$
230 NEXT
240 A$=B$: RETURN
```

Listing 2

```
0000                ;***************************
0000                ;*                         *
0000                ;*  REVERSE VIDEO ROUTINE  *
0000                ;*                         *
0000                ;*   BY CHARLES STANFORD   *
0000                ;*                         *
0000                ;***************************
0000                ;
0000                    LF=$0A          LINE FEED
0000                    CR=$0D          CARRIAGE RETURN
0000                   ESC=$1B          ESCAPE CHARACTER
0000                 CTRLI=$09          CONTROL I CHARACTER
0000                BRANCH=LBLC+1       SELF-MODIFIED BRANCH
0000                OUTPUT=$FF69        MONITOR OUTPUT ROUTINE
0000                GETCHR=$FFBA        GET CHARACTER ROUTINE
0000                                    ;
00D8                *=$00D8
00D8                                    ;
00D8 20BAFF               JSR GETCHR    GET A CHARACTER
00DB C909                 CMP #CTRLI    IS IT A CONTROL I ?
00DD D005                 BNE LBLA
00DF A200                 LDX #0        IF YES, MODIFY BRANCH
00E1 86F7                 STX BRANCH    TO REVERSE CHARACTERS
00E3 60                   RTS
00E4                                     ;
00E4 C91B          LBLA   CMP #ESC      IS IT ESCAPE ?
00E6 D004                 BNE LBLB
00E8 A202                 LDX #2        IF YES, RESET BRANCH TO
00EA 86F7                 STX BRANCH    DISPLAY NORMAL CHARACTERS
```

(continued)

U19 uses the gating of C0, C1, C2, and T3 to trigger the latch in the parallel-serial shift register U42 (see the 600 board schematic). T3 is merely the clock signal delayed through three gates to match delays already present in the video circuits. It would seem that a lesser delay in the trigger to latch IC3 might even things out. Accordingly, U19A piggy-backed to U19 can use three of the signals, and pin 13 can be connected to U43 pin 1, the T1 signal (clock with only one gate of delay). Use pin 8 of U19A instead of pin 8 of U19 to trigger latch IC2a. U43 has some solder pads that make the jumper connection very convenient. To prevent damage to the ICs, be sure to put a dab of solder on each of the pins common to U19 and U19A. Again, a good magnifying glass is invaluable. Pins 1 through 6 are left unconnected.

When you test the computer again, carefully check the reversed characters to be sure that they are completely in sync with the reversing circuit. You may need to use the clock itself, or T2, but T1 seems to be just about right.

Programming Techniques

There are at least a half dozen ways to use BASIC or machine-language software to capitalize on your new character-reversing capability. Using the CHR$, ASC, LEN, and MID$ functions, entire strings can be readily inverted by a relatively short and straightforward subroutine. The demonstration program in listing 1 also can be used in a game or financial planning program to highlight certain inputs or headings. Either inputs or internal strings will reverse, and numeric variables also can be reversed by using the STR$ function.

The machine-language program in listing 2 is more sophisticated. It can reside in the unused (by BASIC) RAM at the top of page zero, but remember that the monitor does use the space when you break. The program intercepts both the "character-get" and the "screen-write" routines of BASIC by changing the indirect addresses at $0218 and $021A. Then the data can be processed as needed for reverse video.

When the routine is in place, the first five lines get the character from the keyboard as usual and act only if either the control-I or escape key is detected. The control-I causes the routine starting at $00E4 to force a "1" into the left bit of the character. Once the control-I is pressed, every character coming from either the keyboard or the ACIA will be inverted before passing to the screen output or program storage. Hitting the escape key will return action to normal. Notice that the routine ignores carriage returns and line feeds. All other characters get the "reverse" treatment. Therefore, be careful to use the routine only for those items that go to the screen or are within quotes. Trying to invert characters involved in program entry will confuse the BASIC interpreter and lead to a program crash.

Listing 2 *(continued)*

```
00EC 60       LBLB   RTS
00ED                            ;
00ED  C90D           CMP #CR    CARRIAGE RETURN ?
00EF  F009           BEQ LBLD   YES, DO NOT CHANGE
00F1  C90A           CMP #LF    LINE FEED ?
00F3  F005           BEQ LBLD   YES, DON'T CHANGE IT
00F5 18              CLC
00F6  9002    LBLC   BCC LBLD   BRANCH ALWAYS (MODIFIED ABOVE)
00F8                            ;
00F8 0980            ORA #%10000000 SET HIGH BIT
00FA  4C69FF  LBLD   JMP OUTPUT  TO MONITOR OUTPUT ROUTINE
```

Listing 3

```
3000 PRINT "MACHINE LANGUAGE"
3010 PRINT "REVERSE VIDEO ROUTINE"
3020 REM --SET INPUT VECTOR--
3030 POKE 536,216 : POKE 537,0
3040 REM --SET OUTPUT VECTOR--
3050 POKE 538,237 : POKE 539,0
3060 FOR M=216 TO 252: READ D: POKE M,D: NEXT
3070 DATA 32,186,255,201,9,208,5,162,0,134,247,96
3080 DATA 201,27,208,4,162,2,134,247,96,201,13,240
3090 DATA 9,201,10,240,5,24,144,2,9,128,76,105,255
3100 PRINT "VECTORS SET & LOADED"
```

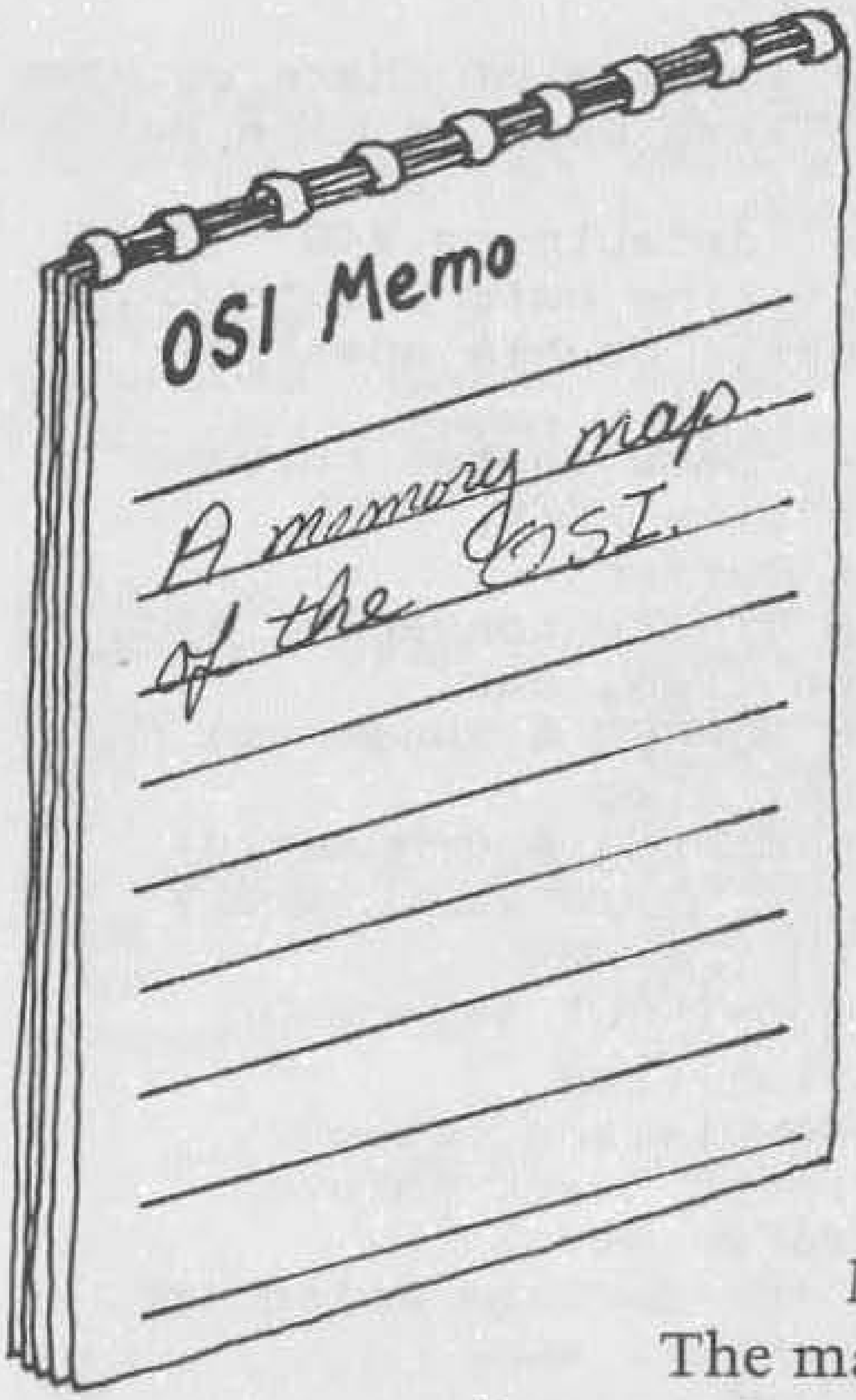
If you are familiar with the method Microsoft uses to store BASIC Source Code starting at $0300, you will be able to devise methods for actually changing the characters by modifying the program itself. It isn't too hard to write a BASIC program that will scan the source code for a particular line number and then invert any characters between quotation marks within that line. I'm sure you will find many creative ways to use this new capability.

Parts List

R1, R2 — 1KOhm ¼ watt
IC1 — 74LS00
IC2 — 74LS74 (option 74S74, see text)
IC3 — 74LS86 (option 74S86, see text)
IC4 — (optional — 74LS20)
S1, S2 — SPST miniature toggle switches (Radio Shack 275-324)
S1A — optional in place of S1 and S2
  SPDT center off min toggle switch (Radio Shack 275-325)
Misc. — PC board, IC sockets, IC header, Molex connector, wire, etc.

OSI C1/C2 ROM BASIC Memory Map

by Michael M. Mahoney



This map is a compilation of data collected from a variety of sources, including but not limited to:

M/A COM-OSI
  Ron Fial
Aardvark Technical Services
  Stan Murphy
CREATIVE COMPUTING
  Gordon Cannady
MICRO
  Ed Carlson
COMPUTE!
  T.R. Berger
Earl Morris
and my own investigation

My thanks to all.

The map is not represented as complete or error-free, so please feel at liberty to send any corrections or additions to:

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

Memory Map

```
 HEXLOC    NAME    DESCRIPTION
------------------------------------------------------------------
0000-00FF          HARDWARE PAGE 0

0000-0002  WARM    Initially Jump to Cold Start ($BD11) then
                      becomes Jump to Warm Start ($A274)

0003-0005  JPRNT   Jump to Message printer at $A8C3
0006-0007  USRINP  USR input argument vector
0008-0009  USROUT  USR output argument vector
000A-000C  USRVEC  USR Vector
000D       NULFLG  NULL FLAG-number of NULL's output to ACIA
                      in addition to BASIC's normal 10.

000E       TRMCNT  Terminal character count-# of chars printed
                      since last CR - also used as Line Buffer Ptr

000F       LINLEN  Output line length (default is $48 - 72)
                      Length of output line before auto CR/LF
                      '0' gives automatic double spacing.

0010       TRWDTH  Terminal width for comma spaced columns
0011-0012  BINARG  Misc Args of stmts like PEEK, POKE, etc.
0013-005A  BUFFER  BASIC's Line Input Buffer
005B       DELIM1  Used by decimal to binary converter etc.
005C       DELIM2  Scan between quotes flag, etc.
005D       CHRCNT  # of characters in BUFFER & Subscript Flag
005E       DIMLET  Default DIM flag,LET flag
005F       VARTYP  Variable Type ($FF=String $00=Numeric)
ØØ6Ø       MFLAG   Misc flag (DATA, LIST, QUOTE FLAG, Etc.)
ØØ61               Subscript Flag=Ø, FN Flag=$8Ø
0062       IRFLAG  READ/INPUT flag ($00=INPUT $98=READ)
0063       COMPFL  Comparison evaluation flag
0064       CTRLO   Control O flag ($80=Discard output)
0065-0067          Temp String Descriptor Stack Pointer
0068-0070          Temp String Descriptor Stack
0071-0072  TMPTR1  Temporary Pointer for Garbage Collector,
                      Dec to Bin converter, etc.


ØØ73-ØØ74  TMPTR2  Temporary Pointer for NEW LINE, DELETE LINE,
                   VAL, Etc.
0075-0078  RESACC  Reserve FP Accumulator
                      Staging area for MULTIPLY

0079-007A  TXTTAB  Start of BASIC Text Pointer ($0301)
007B-007C  VARTAB  Start of Variable Table pointer
                      (End of Program + 1)

007D-007E  ARRTAB  Start of Array Table pointer
ØØ7F-ØØ8Ø  ENDTAB  End of Array Table Pointer
0081-0082  STRSPC  Start of String Space pointer
                      goes from here to end of memory.
```

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
0083-0084	STRPTR	Work pointer into String Space
0085-0086	MEMSIZ	End of memory + 1
0087-0088	XQTLIN	Current Line # - ($88=#$FF if Direct Mode)
0089-008A	OLDLIN	Line # at 'BREAK' or 'STOP'
008B-008C	OLDPTR	Pointer to BASIC Code for 'CONT'
008D-008E	DATLIN	Line # for Current DATA statement
008F-0090	DATPTR	Address of next DATA statement
0091-0092	INPPTR	Address of next value in Current DATA stmt
0093-0094	VARNAM	ASCII name of present variable
0095-0096	VARPNT	Address of present variable
0097-0098	FORPTR	Address of Variable to be assigned value by LET - also FOR/NEXT pointer
0099-00A0		Various work pointers etc.
00A1-00A3	JUMPER	General purpose JMP instruction - Put target Address in $A2-$A3
00A4-00A9		Various work and storage area
00AA-00AB	VARPTR	Pointer to next line after LIST
00AC-00B0	ACCUM1	Floating Point Accumulator # 1 - Format is 1 byte Exponent-3 bytes Mantissa-1 byte Sign
00AD-00AE		Contents are printed in Decimal by $B962
00AE-00AF	FAC	Where INVAR Rtne at $AE05 puts it's Argument
00B0		Sign of Floating Accumulator #1
00B1		Series evaluation Constant pointer
00B2		ACCUM1 high order (overflow) word
00B3-00B7	ACCUM2	Floating accumulator #2 (Format = EMMMS)
00B8		ACCUM1/ACCUM2 sign comparison result flag
00B9		ACCUM1 low order (rounding) word
00BA-00BB		Series pointer
00BC-00D3	CHRGET	PARSER subroutine-gets next byte from ($C3) Returns with character in 'A' CARRY clear if value is ASCII 0-9 else CARRY set. 'A' will equal zero if end of line. Ignores spaces. Copied from $BCEE at Cold Start.
00C2	CHRGOT	Entry to get current character
00C3-00C4	CHRPTR	Code Address pointer for CHRGET routine
00D1-00D7		Used by both BASIC & Extended Monitor
00D4-00D7	RNDX	Random number Seed for RND
00D8-00FA		Not used by BASIC or MONITOR
00FB		ACIA / KEYBOARD flag for MONITOR
00FC		Temporary storage for MONITOR
00FD		Temporary Data storage for MONITOR
00FE-00FF		Temporary Address storage for MONITOR
0100-01FF		HARDWARE PAGE 1
0100-010C		Number conversion to ASCII storage area
0130-0131	NMI	NMI interrupt vectored here
01C0-01C1	IRQ	IRQ interrupt vectored here
0133-01FC	STACK	BASIC's stack area

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
0200-02FF		HARDWARE PAGE 2
0200	CURSOR	Current Cursor offset
0201	SAVER	Character to be printed
0202	CRTWRK	CRT emulator work byte
0203	LOADFL	LOAD flag (0=OFF 1=ON)
0204		CRT temporary
0205	SAVEFL	SAVE flag (0=OFF 1=ON)
0206	BAUD	CRT Emulator BAUD rate (0=FAST 255=SLOW)
0207-020E	VEB	Volatile Execution Block For screen scroll NOT re-entrant
020F-0211		Not used by BASIC
0212	CFLAG	CTRL C flag <>0 is disable
0213	KBWORK	Keyboard Poll work byte
0214	OLDKEY	Keyboard Poll last key
0215	NEWKEY	Keyboard Poll this key
0216	DBOUNC	Keyboard Poll debounce
0217		apparently un-used
0218-0219	C1INP	C1 INPUT vector (=$FFBA)
021A-021B	C1OUT	C1 OUTPUT vector (=$FF69)
021C-021D	C1CTLC	C1 CTRL C vector (=$FF9B)
021E-021F	C1LOAD	C1 LOAD vector (=$FF8B)
0220-0221	C1SAVE	C1 SAVE vector (=$FF69)
0222-02FF		Not used by BASIC
0300-09FF		BASIC workspace (User RAM)
A000-BFFF		** ROM BASIC **
A000-A037	INIDIS	Initial Keywords Dispatch table (= Address of routine minus 1)
A038-A065	FUNDIS	Functions Dispatch table (= actual address of routine)
A066-A083	ARITHD	Arithmetic Operation Table 3 bytes (1=precedence 2 and 3=Address)
A084-A163	KEYTBL	Keyword Tables - in Token order in ASCII with last byte of each Keyword bit 7 on. End of table marked by Null.

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
A084-A0F0	INITLK	Initial Keywords
A0F1-A114	SECNDK	Secondary Keywords
A115-A163	FUNCTK	Functions Keywords
A164-A185	ERRTBL	Error message table-High Bit of 2nd character set
A186-A18C	ERRMSG	ASCII Msg-' ERROR ',$00
A18D-A191	INMSG	ASCII Msg-' IN ',$00
A192-A198	OKMSG	ASCII Msg-CR,LF,'OK',CR,LF,$00
A199-A1A0	BRKMSG	ASCII Msg-CR,LF,'BREAK',$00
A1A1-A1CE		Look back thru BASIC Stack for most recent GOSUB or FOR
A1CF-A211		Open space in memory
A212-A21E	CKSTAK	Check for 'OM' and Stack overflow
A21F-A24B	CKMEM	Check free memory available
A24C	OMERR	Out of Memory error
A24E	ERRPRT	Error message printer-enter with X=offset of message from table at $A164
A274	WARMST	Warm start location
A27D	INMAIN	BASIC input routine
A284-A34A	INSERT	Inserts tokenized line into Text area and Adjusts all forward ptrs exits by JMP INMAIN
A295		Tokenize Buffer and store in text area
A2A2	DELLIN	Deletes line from Text area
A31C-A34D	CHAIN	Rebuild chaining of BASIC lines in Memory
A357	LININP	Input and fill buffer-put null at end
A386	CHRINP	Input a character - calls $FFEB
A399	TOGGLO	Toggle CTRLO - Output flag
A3A6-A431		Tokenize in Buffer
A432-A460	FINDLN	Find BASIC line whose # <= contents of BINARG and place address in $AA-AB Carry Flag Set if line found, Clear if not ending with a NULL ($ØØ) or a colon
A461-A479	NEWCMD	NEW routine
A463		Put $0 in Start Text ($0301-0302) - then
A46B		Reset VARTAB to TXTTAB+2 - then
A477		Reset CHRPTR - then
A47A	CLEAR	Reset STRSTC to equal MEMSIZ - then
A482		Reset ARRTAB & ENDTAB to = VARTAB - then
A48E		Do RESTORE - then
A491		Put #$68 into Address $65 - then
A495		Reset BASIC Stack to #$FC - then
A4A0		Disable 'CONT' & zero Subscript Flag
A4A7		Initialize Code Pointer (CHRPTR) to the beginning of program ($0301)

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
A4B5	LSTCMD	LIST routine
A556-A5FE	FORCMD	FOR routine
A5C2-A619		Main BASIC execution loop
A5FC	XQT	Entry to BASIC execute loop
A61A-A628	RSTCMD	RESTORE routine
A629	CTLC	CTRL C routine
A636	CTLCMD	CTRL C entry point
A638	STPMCD	STOP routine
A63A	ENDCMD	END routine
A661-A67A	CNTCMD	CONT routine
A67B	NULCMD	NULL routine
A68C	CLRCMD	CLEAR routine
A691-A69B	RUNCMD	RUN routine
A69C-A6B8	GSBCMD	GOSUB routine
A6B9-A6E5	GTOCMD	GOTO routine
A6E6-A70B	RETCMD	RETURN routine
A6F4	RGERR	Return W/O Gosub error
A6F7	USERR	Undefined Statement error
A70C-A719	DATCMD	DATA routine
A71A-A73B	NXSTMT	Scan for next BASIC Statement
A71D-A73B	NXLINE	Scan for next BASIC Line
A73C-A74E	IFCMD	IF routine
A74F-A75E	REMCMD	REM routine
A75F-A77E	ONCMD	ON routine
A77F-A7B8	EVAL	Evaluate expression whose beginning address is in CHRPTR. Convert ASCII to fixed with result appearing in BINARG.
A785		Same as EVAL without zeroing the result field first (BINARG)
A7B9-A828	LETCMD	LET routine
A829-A8C2	PRTCMD	PRINT routine - Entry at $A82F
A866		Puts null at end of buffer - then
A86C	DOCRLF	Output CR/LF - then
A878	DONULL	Output Nulls from NULFLG and RTS
A88B	COMCOL	Handle comma separators in PRINT routine
A8A2	SPCTAB	Do SPC and TAB in PRINT routine
A8C3-A8DF	MSGPRT	Print ASCII message. Enter with ADDR HI in Y,ADDR LO in A. Message is ASCII ending with a NULL ($00)
A8E0	SPCOUT	Outputs one space (' ')
A8E3	QMOUT	Outputs question mark ('?')
A8E5	AOUT	Outputs character in A updates TRMCNT and checks for Line Length overflow
A904	BADINP	Handle bad input data
A923-A945	INPCMD	INPUT routine - Clears CTRL O
A925		INPUT without clear CTRL O
A946-A94E	QINPLN	Prompt with '? ' then receive INPUT via Jump to LININP ($A357)

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
A94F	REACMD	READ routine
AA1C-AA2C	XTRMSG	ASCII Msg-'?EXTRA IGNORED',CR,LF,$00
AA2D-AA3F	RDOMSG	ASCII Msg-'?REDO FROM START',CR,LF,$00
AA40	NXTCMD	NEXT routine
AAAD	FRMEVL	Formula Expression Evaluator Gets Value from BASIC line (Evaulates Literals, Variables or Expressions). Puts value in ACCUM1, does TM check.
AABC	TMERR	Type Mis-match error
AAC1	FRMEV2	Same as FRMEVL without TM check
ABA0	NUMEVL	Numeric Expression Handler
ABAC	STREVL	String Expression Handler
ABD8	NOTCMD	NOT routine
ABF5-ABFA		Evaluate expression within parentheses
ABFB	CKRPAR	SN Error if next character not ')'
ABFE	CKLPAR	SN Error if next character not '('
AC01	CKCOMA	SN Error if next character not ','
AC03	CKCHR	SN Error if next char not = contents of A
AC0C	SNERR	Syntax Error
AC18	GETVAR	Find Variable, put addr in $AD-AE, check Variable type, and if Numeric transfer Value to ACCUM1.
AC27-AC65		Setup and JMP to Functions
AC66-AC93	ORCMD	OR routine
AC69-AC93	ANDCMD	AND routine
AC96-ACFD		Perform Comparisons
AD01-AD0A	DIMCMD	DIM routine
AD0B-AD80	FNDVAR	Get variable name from BASIC line, put name in VARNAM, find and put address of variable in VARPTR and in A & Y
AD53	FNDSIM	Find or Create Simple Variable Expects variable name in VARNAM, finds and puts address in VARPTR and A & Y
AD81-AD8A	CKLETR	Check if char in A is A-Z, Set carry if yes
AD8B-ADE5	CRESIM	Create Simple variable in table
ADE6-ADF6		Array pointer subroutine
ADF7-ADFA		FP Constant (-32768)
ADFB-AE16	INTEVL	Evaluate Integer Expressions
AE05	INVAR	Convert ACCUM1 to integer (+/-32768) in $AE,$AF

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
AE17-AFAC	FNDARR	Find or Create Array
AE85	BSERR	Bad Subscript Error
AE88	FCERR	Function Call Error
AEA1	CREARR	Create Array in Table
AF7C		Compute array subscript size
AFAD-AFC0	FRECMD	FRE(X) routine - Calls Garbage Collector
AFC1-AFCD	OUTVAR	Assigns value to Variable ???
AFCE-AFD3	POSCMD	POS routine
AFD4-AFDD	IDCHK	Check for ID Error
AFD9	IDERR	Illegal Direct command error
AFDE-B00A	DEFCMD	DEF routine
B00B-B01D	FNCHK	Check FN syntax
B01E-B08B	FNEVAL	Evaluate FNx and Store in Stack
B08C-B114	STRCMD	STR$ routine
B0AE-B114		Scan and set-up String & Find Length
B0F3	STERR	String Too complex error
B115-B146		Allocate room in String storage area and build Descriptor for String
B147-B24C	GARCOL	Garbage Collector routine
B1D4-B217		Check for string most eligible for collection
B218		Collect a string
B24D-B289		Perform Concatenation
B268	LSERR	String too Long error
B28A-B2B2		Store String in String Area
B2B3-B2EA		Discard unwanted String
B2EB-B2FB		Clean String Descriptor Stack
B2FC-B30F	CHRCMD	CHR$ routine
B310-B33B	LFTCMD	LEFT$ routine
B33C-B344	RGTCMD	RIGHT$ routine
B347-B38B	MIDCMD	MID$ routine
B38C-B391	LENCMD	LEN routine
B392-B39A		Find String & Length
B39B-B3A7	ASCCMD	ASC routine
B3AB-B3BC	GETBYT	Evaluate Integer (<256) from line into X
B3BD-B3FB	VALCMD	VAL routine
B3FC-B407		Gets 16 bit value from line-puts in BINARG, checks for comma, then gets 8 bit argument in X and then RTS
B408-B41D	FIX	Convert contents of ACCUM1 to two byte Fixed binary number and put in BINARG
B41E-B428	PEKCMD	PEEK routine
B429-B431	POKCMD	POKE routine
B432-B44D	WAITCD	WAIT routine
B44E-BCED		6 DIGIT FLOATING POINT MATH PACKAGE
B44E-B454		Add 0.5 TO ACCUM1
B455	MINUS	Perform Subtraction
B467	PLUS	Perform Addition
B4BB		Subtract ACCUM2 from ACCUM1
B4D0		Arithmetic to normalize floating point
B4F1		Set ACCUM1 to zero
B4F8		Add ACCUM1 to ACCUM2

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
B537		Complement ACCUM1
B564-B568	OVERR	Overflow Error
B569-B59B		Multiply a byte
B59C-B5BC		LOG Coefficients Constants
B5BD	LOGCMD	LOG routine
B5FB	MULT	Perform Multiplication ('*')
B62D		Add RESACC to ACCUM1
B64D-B672		Unpack Memory into ACCUM2
B673-B68F		Test and Adjust ACCUM1 and ACCUM2
B690-B69D		Handle underflow and overflow
B69E-B6B4		Multiply ACCUM1 by 2
B6B5-B6B8		Floating Point constant (2)
B6B9-B6C7		Divide by 2
B6CA	DIV	Perform Divide by
B6CF		Perform Divide into
B711		Subtract ACCUM1 from ACCUM2 result in ACCUM1
B737	D/OERR	Divide by zero error
B74B-B76A		Unpack memory into ACCUM1
B76B-B79A		Pack ACCUM1 into memory
B79B-B7AA		Move ACCUM2 to ACCUM1
B7AB-B7B9		Move ACCUM1 to ACCUM2
B7BA-B7C7		See if need to Normalize ACCUM1
B7CA-B7D7		Get sign of ACCUM1
B7D8-B7F4	SGNCMD	SGN routine
B7E8	FLOAT	Convert contents of $AD(Hi)-$AE(Lo) from Fixed Binary to floating point and put in ACCUM1. Enter with X=#$90 and Carry Set
B7F5-B7F7	ABSCMD	ABS routine
B7F8-B830		Compare ACCUM1 to Mem at (A,Y)
B831-B861		Convert Floating to Fixed
B862-B886	INTCMD	INT routine
B887-B946		Convert ASCII String to Floating Point
B947-B952		constants used with ASCII conversion
B953-B96D		Prints ' IN ' and the Line Number
B95A		Prints current line number
B95E-B96D	NUMPRT	Print decimal integer whose value is in A (lo) and X (hi)
B962		Print contents of $AD(Hi)-$AE(Lo) as Dec. integer
B96E-BA95	ASCII	Convert floating number in ACCUM1 to ASCII string at $0100-0107. $0100 is sign (space or -) - String terminated by NULL
BA96-BAAB		Constants
BAAC-BCED		FUNCTIONS PACKAGE
BAAC	SQRCMD	SQR routine
BAB6	POWER	Perform ^ (exponentiation)
BAEF-BAF9		Perform negation

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
BAFA-BB1A		Constants EXP Coefficients
BB1B-BB6D	EXPCMD	EXP routine
BB6E-BBB7		Series Summations
BBB8-BBBF		RND Constants
BBC0-BBFB	RNDCMD	RND routine
BBFC	COSCMD	COS routine
BC03	SINCMD	SIN routine
BC4C	TANCMD	TAN routine
BC78-BC98		SIN & COS Constants & Coefficients
BC99-BCC8	ATNCMD	ATN routine
BCC9-BCED		ATN Coefficients
BCEE-BD09	PARSER	CHRGET - Transferred to $BC
BD0A-BD10		Prints Author's Name
BD11	COLD	Cold Start routines
BE39-BE4D		ASCII msg-'WANT SIN-COS-TAN-ATN',$00 Left over from when Tape BASIC
BE4E-BE71		ASCII msg- Author's Name
BE72-BE7D		ASCII msg'MEMORY SIZE',$00
BE7E-BE8C		ASCII msg-'TERMINAL WIDTH',$00
BE8D-BEE1		ASCII msg-Version & Copyright Notice
BEE4		UART input routine (430 Board) Contains error (BF05 should be FB05)
BEF3		UART output routine (430 Board)
BEFE		UART initialization routine (430 Board)
BF07		C2 ACIA at $FC00 input routine
BF15		C2 ACIA at $FC00 output routine
BF22		C2 ACIA at $FC00 initialization set to 8 bits data, 2 stop bits no parity, divide by 16
BF2D-BFF2	CRTEMU	CRT emulator - prints char in 'A' to screen, scrolls etc.
BFC2		Prints char in 'A' to screen - but doesn't update Cursor pointer
BFF3-BFFA		Code transferred to VEB for Scroll
C000-C01F		Disk Controler PIA and ACIA
CE00-CEFF	MULTIP	C3 Multi User Ports
CF00-CFFF	PORT8	CA-10-X Board ACIA's
D000-DFFF		C3 Hard disk buffer
D000-D3FF		C1 Video memory
D000-D7FF		C2 Video memory
DE00		C2 Screen size/Color/Sound Latch
DF00	KBPORT	Polled Keyboard Port
E000-EFFF		C3 Level 3 & CP/M Memory
E000-E7FF		Color Memory (Low 4 bits)

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
E800-EFFF		Not presently used in C1/C2
F000-F001		C1 ACIA Port #1
F800-FBFF		C1 65V ROM extra pages for other machines
FC00-FC01		C2 ACIA Port #1
FC00-FCFF		C1 Floppy Bootstrap Routines
FC00		C1 Auto bootstrap entry
FC06		C1 Load track zero into $2200 up
FC8B		C1 Unload Floppy head
FC91		C1 Time delay routine - delay equals 1.25 MS times value of X at 1 MHZ
FC9C		Load next byte from disk to A
FCA6		C1 ACIA at $F000 initialization
FCB1		C1 ACIA at $F000 output routine
FCBE		Write complement of A to keyboard
FCC6		Load complement of keyboard into X
FCFF		Load complement of keyboard into A
FD00-FDFF	KBPOLL	Keyboard Polling Routine - Polls keyboard and returns with the ASCII value of key depressed in A
FE00-FEFF		65V PROM MONITOR
FE00	MONITR	Entry-clears screen, reset ACIA and Stack
FE0C	MONWRM	Entry-without Stack initialization
FE43	MONADR	Entry to address mode
FE80	OTHER	Input ASCII character, returns result in A, bit 7 cleared
FE93	LEGAL	Convert ASCII hex to binary, result in A (A=$80 if not ASCII Hex 0-F)
FEAC	MONOUT	Output Address & Data in Monitor format (ADDR from $FE-FF DATA from $FC)
FEB0	MOUT1	Output X bytes from $FC+X to screen at $D0C6+Y. Set X and Y before entering X decreases and Y increases.
FECA	DIGIT	Output LSD (HEX) from A to screen at $D0C6+Y. Set Y before calling.
FEDA	ROLL	Move LSD (HEX) in A to 2 byte number at ($FC) +X. Set X before calling.
FEE9	MONINP	Return Character in A from Keyboard or ACIA Port 1, depending on LDFLAG

(continued)

Memory Map *(continued)*

HEXLOC	NAME	DESCRIPTION
FF00-FFFF		BASIC I/O SUPPORT
FF00	RESET	Reset Entry Point
FF67	BASOUT	BASIC's output vector. Outputs 1 byte to screen, and if SAVEFL on, outputs to Port #1 (also ouput 10 NULLS & <LF> if char is <CR>)
FF89	LOADRT	LOAD flag routine
FF94	SAVERT	SAVE flag routine
FF99	CTLCRT	Control C check routine
FFB8	BASINP	BASIC's Input character routine
FFE0	HOME	Home position of cursor (C1=$64 C2=$40)
FFE1	LEN	Line Length default value
FFE2	SIZE	Screen Size type (C1=0 C2=1)
FFE3-FFEA		Misc work pointer default values
FFEB-FFED	INPUT	BASIC INPUT vector C1=JMP($0218) C2=JMP $FFB8
FFEE-FFF0	OUTPUT	BASIC OUTPUT vector C1=JMP($021A) C2=JMP $FF67
FFF1-FFF3	CTRLC	Control C check vector C1=JMP($021C) C2=JMP $FF99
FFF4-FFF6	LOAD	BASIC LOAD vector C1=JMP($021E) C2=JMP $FF89
FFF7-FFF9	SAVE	BASIC SAVE vector C1=JMP($0220) C2=JMP $FF94
FFFA-FFFB	NMIVEC	NMI Vector (= $0130)
FFFC-FFFD	RESETV	RESET Vector (= $FF00)
FFFE-FFFF	IRQVEC	IRQ Vector (= $01C0)

BASIC Enhancements
Machine Language Aids
I/O
Hardware
Reference

MICRO on the OSI

Technical Editor: Kerry Lourash

24 Articles by 18 Authors

About the Book

MICRO on the OSI is a compilation of articles that have appeared in MICRO magazine as well as newly written material that appears for the first time in this book. Categories covered are BASIC Enhancements, Machine-Language Aids, I/O, Hardware, and Reference.

Chapter topics that provide BASIC Enhancements include a program to help you recover from crashes intact, a utility for deleting blocks of lines as well as single lines with just a few keystrokes, and two fixes for ROM BASIC — an Error Message patch and a Garbage Collection patch. There are programs to add extra capabilities to OSI's Microsoft BASIC and to allow the AT keyword to be recognized. Elementary line editing is explained as well as auto line numbers for OSI disk BASIC and an autonumber program for cursor control. Also presented are a runtime utility that enables you to trap certain non-fatal errors and continue program execution, a cross-reference generator, an extended OSI BASIC, and two trace routines.

Machine-Language Aids provided include a routine for listing the symbol table generated by the OSI C1P Assembler, a short program that inserts spaces to create an improved LIST command, and a search-and-change utility. You will also find a debugger for machine-language programs and a polled keyboard for the C1P/Superboard that generates both upper-and lower-case characters by continuously interrogating the keyboard.

I/O Enhancements include a routine that allows you to add extra keyboard functions, a load-and-save program for tape at 300, 600, or 900 baud, and an extended I/O processor.

Two hardware fixes show you how to add a screen blanker, inverse upper case, and a dim character set to your Challenger and to make a modification to add programmable reverse video to your C1P.

And, finally, for your reference we present a C1/C2 ROM BASIC Memory Map.

You will find *MICRO on the OSI* is an informative and useful adjunct to your OSI microcomputer.

About the Editor

Technical Editor Kerry Lourash owns a Superboard II and is interested in both hardware and software. Among his special interests are deciphering BASIC-in-ROM and designing utilities. Mr. Lourash has contributed many articles to MICRO magazine.

MICRO INK
P.O. Box 6502
Amherst, New Hampshire 03031