JAN/FEB $1.50

ohio scientific's SMALL SYSTEMS JOURNAL

VOLUME 2 NO.1

features page

*SPECIAL NEXT ISSUE * DON'T MISS IT!*

The next issue will be a special issue on disk operating systems, comparing the features and applications of OS-65U, OS-65D V2.2, OS-65D V2.0, OS-65D V2.0 9-DIGIT, DMS-1, and OS-65D V2.4.

The magazine for 6502 computer enthusiasts!

INDEX TO VOLUME 1, July-December 1977

Introduction

This issue of the journal finds itself with a new editor. Namely; myself, Rick Whitesel. I hope to always have something for everyone in each issue of this journal. In order to do this I ask that you, the readers, will drop me a note on what you did or did not like in this issue as well as the previous issuses.

This issue contains an index to the previous journals, a long awaited explanation of 8K BASIC's USR FUNCTION, and a article showing how to adapt OS-65D V2.0 disc files to the 74-MEGABYTE HARD DISC. In the quickies corner, chessboard, a video BASIC game that displays a conventional chessboard on the screen and permits each player to move thier pieces by entering the from and to coordinates. Further into this issue is a "how to" article on converting 9-Digit BASIC into and end-user BASIC. Next is a outline of the new OS-65U disc operating system. This is followed by two debugging programs. The first may be used on 500 or 510 systems while the second is designed to be used on a 510 system with the software processor select switch.

Ohio Scientific's Small Systems Journal is issued bimonthly by Ohio Scientific, Inc., P.O. Box 36, Hiram, OH 44234. The subscription rate is six dollars per six issues. Individual copies are $1.50. Published in Twinsburg, OH by the Twinsburg Bulletin.

Vol. 2, No. 1 Jan./Feb., 1977
Editor-in-Chief Rick Whitesel
Production Manager Don Muchow
Contributing Editors Mike Cheiky
Bob Coppedge
Eric Davis
Jim Halverson

What's a USR Function?

In the real world of computer applications, BASIC has proved to be quite adequate. However, there are applications where it would be nice to have BASIC'S number crunching capability with machine code's speed. That is where OSI's USR function in BASIC comes into play. Via the USR function, one can have a BASIC program which works in conjunction with one or several machine code programs. When BASIC executes the USR function, it goes to VECTOR and VECTOR +1 (defined in the table below). There BASIC "picks up" the address of the machine code program and jumps to it. Once in the machine code program, one may execute two separate routines. These routines allow variable passing to and from BASIC. To pass a variable from BASIC, the routine pointed to by the contents of memory locations 6 and 7 must be executed. The 15 bit signed number can then be picked up at FACLO and FACHI. To pass a variable back to BASIC the low byte is placed in the Y - register and the high part is placed in the accumulator. The routine pointed to by the contents of memory locations 8 and 9 must be executed. Therefore, the following lines in BASIC would pass the value of X to the machine code program and upon returning to BASIC, X would be equal the value passed back.

```
 10 X=10
 20 X=USR(X)
 30 PRINT "X NOW EQUALS";X
```

Below are the steps required to implement the USR function.
1) Set BASIC's memory size so it does not overlap the machine
 code program.
 2) Set up VECTOR (low) and VECTOR +1 (high) to point to the
 machine code program.
 3) In the machine code program, insert the following code to
 allow variable passing:

```
 BEGIN    JSR GETVAR

 GETVAR   JMP (INVAR)
```

 4) To pass a variable back, execute the following machine code.

```
  FINISH  LDY LOWBYT
          LDA HIGHBY
          JMP (OUTUAR)
```

After execution of "FINISH", BASIC will continue with (in this case) X=the number passed in Y and A. Examples:

C2-4P or C2-8P
 1) Set memory size=3000
 2) Load the following machine code at $0FD0

```
A*RA

INIZ?N
 A

   10 0000                  USR SUB
   20 0000                  FACLO=$AF
   30 0000                  FACHI=$AE
   40 0000                  INVAR=$06
   50 0000                  OUTVAR=$08
   60 0000                  LOWBYT=$01
   70 0000                  HIGHBY=$00
   80 0000
   90 0FD0                  *=$0FD0
  100 0FD0
  110 0FD0
  120 0FD0 20E40F BEGIN     JSR GETVAR
  130 0FD3 A5AF             LDA FACLO
  140 0FD5 8DE70F           STA TEMPL
  150 0FD8 A5AE             LDA FACHI
  160 0FDA 8DE80F           STA TEMPH
  170 0FDD A001             LDY #LOWBYT
  180 0FDF A900             LDA #HIGHBY
  190 0FE1 6C0800           JMP (OUTVAR)
  200 0FE4 6C0600 GETVAR    JMP (INVAR)
  210 0FE7 00     TEMPL     .BYTE $00
  220 0FE8 00     TEMPH     .BYTE $00
```

 3) Load $000B with $D0 and
  Load $000C with $0F
 4) Execute the following BASIC lines:

```
 10 X=10
 20 X=USR(X)
 30 PRINT X
```

 5) X now equals the value passed and TEMPL contains the low part
 passed from BASIC while the high part is in TEMPH. Note the value
 of the variable passed ranges from - 32268 to +32268 (Bit 15 is the
 sign Bit, 1=negative).

DISC BASED BASIC

 1) Set memory size=15000
 2) Load the following machine code at $3FD0

```
 .GET
FROM DRIVE, TRACK: B,71

 A

   10 0000                  USR SUB
   20 0000                  FACLO=$B2
   30 0000                  FACHI=$B1
   40 0000                  INVAR=$06
   50 0000                  OUTVAR=$08
   60 0000                  LOWBYT=$01
   70 0000                  HIGHBY=$00
   80 0000
   90 3FD0                  *=$3FD0
  100 3FD0                ;
  110 3FD0                ;
  120 3FD0 20E43F BEGIN     JSR GETVAR
  130 3FD3 A5B2             LDA FACLO
  140 3FD5 8DE73F           STA TEMPL
  150 3FD8 A5B1             LDA FACHI
  160 3FDA 8DE83F           STA TEMPH
  170 3FDD A001             LDY #LOWBYT
  180 3FDF A900             LDA #HIGHBY
  190 3FE1 6C0800           JMP (OUTVAR)
  200 3FE4 6C0600 GETVAR    JMP (INVAR)
  210 3FE7 00     TEMPL      BYTE $00
  220 3FE8 00     TEMPH      BYTE $00
```

 3) Load VECTOR with $D0 and VECTOR +1 with $3F
 4) Same as for C2-4P and C2-8P
 5) Same as for C2-4P and C2-8P

Below are the various memory locations for the various BASIC's.

	6-digit	9-digit	ROM BASIC
VECTOR	$023E	$023E	$000B
VECTOR+1	$023F	$023F	$000C
OUTVAR	($08)	($08)	($08)
INVAR	($06)	($06)	($06)
FACLO	$AF	$B1	$AF
FACHI	$AE	$B2	$AE

Vector refers to the pointer to the machine code routine. OUTVAR is the location whose contents point to the routine which passes a variable to BASIC. INVAR contains a pointer to the routine which passes a variable from BASIC. FACLO and FACHI are the locations where a machine code can pick up the variable passed from BASIC. One final note, if you have any special uses of the USR function, please send us a letter on how you are using it.

QUICKIE!

This montns quickie is a decimal to binary number converter. The program uses some rather clever tricks to implement the connversion routine. Be sure to follow how the connversion is done.

```
50 PRINT
60 PRINT
70 PRINT "DECIMAL TO BINARY"
80 PRINT "   CONVERTER"
90 PRINT
93 PRINT
95 PRINT
100 INPUT X
101 IF X<0 THEN GOTO 330
102 IF X>32767 THEN GOTO 330
104 PRINT
105 PRINT "X=";
110 Y=16384
120 A=INT(X/Y)
130 IF A=0 THEN GOTO 200
140 PRINT "1";
150 X=X-Y
160 GOTO 300
200 PRINT "0";
300 Y=Y/2
310 IF INT(Y)=0 THEN GOTO 320
315 GOTO 120
320 GOTO 90
330 END
```

Bugs&Fixes

If you are having trouble with the 4K on board RAM on the 500 CPU board, check the foil runs above IC-F9 (A 7404). There have been cases of these runs overlaying each other causing loss of bit 2 in the memory.

Two bugs have been found in 9 Digit BASIC. The first causes the SPC function to act like the TAB function and the second is a bug in the string manipulation routines. The corrections are as follows, respectively:

Locations	Old	New Contents
$0A3E	$9F	$2C
$0A3F	$F0	$18
$0A40	$68	$F0
$0A41	$C9	$50
$0A42	$2C	$49
$0A43	$18	$9F
$0A45	$4C	$63
$0A47	$3B	$A4

AND

$0B70	$C4	$98
$0B72	$00	$99
$0BD6	$4C	$98
$0BD7	$0B	$99

C2-8Pers and C2-4Pers please make note that Basic's USR function vector is at decimal locations 10 and 11 and not as stated in the BASIC Manual.

If you are having any trouble with your 430B cassette interface, try the following. Double the 555's clock frequency and insert a 7474 in series with its output to obtain a symetrical clock. One might also try inserting a RC noise filter to eliminate any D.C. offset coming from the tape recorder's output.

Contributed Program CHESSBOARD

This is, in effect, a computer chess board, no more, no less. It moves pieces and displays the new board. In most cases, it will allow a player to cheat, as will a real chess board.

The board is a standard 8 x 8 board with locations given by the values A,1 through H,8 (see diagram).

```
   A - B - C - D - E - F - G - H
1      *       *       *       *
2  *       *       *       *
3      *       *       *       *
4  *       *       *       *
5      *       *       *       *
6  *       *       *       *
7      *       *       *       *
8  *       *       *       *
   A - B - C - D - E - F - G - H
```

Initially, the board is set in the standard way with white occupying the top two rows.

Moves are achieved through giving the initial position of the piece and the final position of the piece separated by a dash ("-"). For example, to move the white king's pawn from its initial position to the fourth row in the same column, (P-K4) would take a command "D2-D4". Castleing is achieved by the FC and QC commands, KC being king side castle and QC being queen side castle. The program automatically checks for white or black in these commands.

If a pawn achieves the home row of the opponent, the player may change the pawn into a queen by the Q switch at the end of the command string, for example: E2-E1Q
would cause the black pawn on E2 to move to E1 and become a queen.

Unfortunately, the 4K capacity of the BASIC Challenger II is insufficient to handle an error correction routine or a game record, however, the game record is easily added. A return without an input will end the game.

Suggestions: A routine that could interpret standard English chess notation would be nice.

10-100	Initialize Board
140-160	Input Command
170-179	Options
180-190	Interpreting of command string
190-210	Black or white
220-230	Move piece
235	Change piece?
240	GOTO print section
250	If black then GOTO 280
260-270	White king side castle
280	Black king side castle
300	If black, then 330
310-320	White queen side castle
330	Black queen side castle.
799-855	Print updated board
900-912	Change pawn to queen
1000-1040	Initialization data
2000	End

```
   1 REM**DANIEL GLASSER
   2 REM**CHESS BOARD
  10 DIM B1(8,8),B2(8,8),P$(16)
  30 FOR X=1 TO 2:  FOR Y=1 TO 8:  READ S
  42 B1(X,Y)=S:  B1(X+2,Y)=S:  B1(X+4,Y)=S:  B1(X+6,Y)=S
  50 NEXT Y:  NEXT X:FOR X=1 TO 2:FOR Y=1 TO 8:READ S:B2(X,Y)=S
  90 NEXT Y:NEXTX:FORX=7TO8:FOR Y=1 TO 8:READ S:B2(X,Y)=S:NEXT Y
 100 NEXTX:FORX=1TO16:READS$:P$(X)=S$:NEXTX:GOTO800
 140 IF W=1 THEN 160
 141 INPUT"W>";M$:GOTO 170
 160 INPUT"B>";M$
 170 IF M$="END"THEN2000
 177 IF M$="KC" THEN 250
 178 IF M$="QC" THEN 300
 179 IF M$="BOARD" THEN 799
 180 A$=LEFT$(M$,1):B$=MID$(M$,2,1):A1$=MID$(M$,4,1):B1$=MID$(M$,5,1)
 190 B=VAL(B$):C=VAL(B1$):IFW=1THEN210
 200 W=1:GOTO220
 210 W=0:W2$(K)=M$
 220 P1=ASC(A$)-64:P2=ASC(A1$)-64
 230 U=B2(B,P1):B2(B,P1)=0:B2(C,P2)=U
 235 IF RIGHT$(M$,1)="Q" THEN 900
 240 GOTO 799
 250 IF W=1 THEN 280
 260 W=1
 265 B2(1,1)=0:B2(1,4)=0:B2(1,2)=7:B2(1,3)=4:GOTO799
 280 W=0:B2(8,1)=0:B2(8,4)=0:B2(8,2)=14:B2(8,3)=11:GOTO799
 300 IFW=1THEN330
 310 W=1
 320 B2(1,8)=0:B2(1,4)=0:B2(1,6)=7:B2(1,5)=4:GOTO799
 330 W=0:B2(8,8)=0:B2(8,4)=0:B2(8,6)=14:B2(8,5)=11:GOTO 799
 799 FOR M5=1 TO 4:PRINT:NEXT M5
 800 PRINT"-A----B----C----D----E----F----G----H--"
 801 PRINT
 810 FOR X=1 TO 8:  FOR Y=1 TO 8
 820 S1=B2(X,Y)
 830 IF S1=0 THEN S1=B1(X,Y)
 840 S$=P$(S1):PRINTS$;"  ";
 850 NEXTY:PRINT"    ";X:PRINT:PRINT:NEXT X
 851 PRINT"-A----B----C----D----E----F----G----H--"
 855 GOTO 140
 900 IF C=8 THEN B2(C,P2)=8:GOTO 799
 910 IF C=1 THEN B2(C,P2)=15
 912 GOTO 799
1000 DATA 1,2,1,2,1,2,1,2,2,1,2,1,2,1,2,1
1010 DATA 4,5,6,7,8,9,5,4,3,3,3,3,3,3,3,3
1020 DATA 10,10,10,10,10,10,10,10,11,12,13,14,15,16,12,11
1030 DATA "   "," ##"," WP"," WR"," WN"," WB"," WK"," WQ"," WB"
1040 DATA " BP"," BR"," BN"," BB"," BK"," BQ"," BB"
2000 END
```

~ ~ ~

DOS CNTRL

This subroutine in BASIC may be used to perform transfers to or from Ohio Scientific's new hard disk drive. The transfer are set up as single sector transfers which are 3584 bytes in length. The parameters that must be specified are minimal and are listed below ---

DC -- Disk cylinder	(0 through 338)
DT -- Disk track	(0 through 11)
DS -- Disk sector	(0 through 4)
DD -- Disk direction	(0 = read / 1 = write)

All transfers are done into or from $E010 up for 3584 bytes. If the drive is not powered up when this sub is executed, the program will terminate. Please note that this "kluge" method is not required under OSI's OS-65U. This program is merely an example of how to modify programs presently running under OS-65D V2.0

The following is a line by line description of the subroutine.

Line(s) Function

63000 Bypasses Initialization of the control port after first pass
63005 Initializes the control port on first pass only
63020 Time delay to allow the disk to get ready
63020 If the disk is not ok by this time then error
63030 This line waits for the disk ready signal
63040 Sets DF=1 if cylinder >255
63050 Pokes flag bit and cylinder vaule
63060 Track is or'ed with flag bit and poked
63070 Pokes header for disk
63080 These lines define the absolute disk address of
63090 the sector (start and end)
63100 a requirement of the system is such that a read
63110 must offset the start address by three (3)
63140 This line pokes the sector start address
63150 This line pokes the sector end address
63160 This line pokes the direction flag
63170 This line waits for the drive to get ready
63180 This line is oring the direction flag with the GO bit
63190 This line waits for the transfer complete signal
63200 And of course this line simply returns from this sub

TELEPHONE DIRECTORY EXAMPLE PROGRAM

This program is a modified version of the program found on all version 2.0 diskettes. The only modifications required were to create a new subroutine at line 1000, modify the "end of file" values, and to set the memory I/O pointers to point at the $Exxx address of the file. The actual start of file is at E010 (57360) and the actual end of file address is at $EEOF (60943).

The following lines describe the modifications to the original program

lines 100,110 changed to point at the new start of file address
lines 165, 167 changed to accommodate the new end of file address
line 200 changed to set the direction of transfer (DD)
line 400 changed to set the direction of transfer
lines 410, 420 changed to accommodate the new start of file address
lines 460, 470 changed to accommodate the new end of file address
lines 620, 630 changed to accommodate the new start of file address
lines 705, 707 changed to accommodate the new end of file address
line 1000 now sets value of disk cylinder, disk track, and disk sector
line 1010 now is a return from subroutine
lines 1020-1080 deleted from program

```
 10 PRINT"DISK BASED PHONE DIRECTORY
 20 PRINT"COMMAND";
 30 INPUT A$
 40 IF A$="NEW" THEN GOTO 100
 50 IF A$="ADD"THEN GOTO 400
 60 IF A$="FIND"THEN GOTO 600
 65 IF A$="EXIT" THEN GOTO 9999
 70 GOTO 20
100 POKE 11860,16
110 POKE 11861,224
120 PRINT"NAME";
130 INPUT B$
140 PRINT"NUMBER";
150 INPUT C$
152 POKE 8708,16
154 PRINT B$
156 PRINT C$
158 POKE 8708,1
160 IF B$="END" THEN GOTO 200
165 Z=PEEK(11860)+(PEEK(11861)*256)
167 IF Z>60900 THEN PRINT"OVERFLOW": GOTO 20
170 GOTO 120
200 DD=1: GOSUB 1000
210 GOTO 20
400 DD=0: GOSUB 1000
410 POKE 11879,16
420 POKE 11880,224
```

```
  430 POKE 8707,8:POKE 8708,128
  440 INPUT K$
  445 POKE 8707,1:POKE 8708,1
  450 IF K$="END" THEN GOTO 500
  460 Q=PEEK(11879)+(PEEK(11880)*256)
  470 IF Q>60900 THEN PRINT"EDIT OVERRUN": GOTO 20
  480 GOTO 430
  500 R=PEEK(11879):S=PEEK(11880)
  510 R=R-4
  520 IF R<0 THEN S=S-1:R=R+256
  530 POKE 11860,R:POKE 11861,S
  540 GOTO 120
  600 PRINT"NAME";
  610 INPUT N$
  620 DD=0: GOSUB 1000
  630 POKE 11879,16
  640 POKE 11880,224
  650 POKE 8707,8:POKE 8708,128
  660 INPUT E$
  670 INPUT F$
  680 POKE 8707,1:POKE 8708;1
  690 IF E$=N$ THEN PRINT "THE NUMBER IS ";F$:GOTO 20
  700 IF E$="END" THEN PRINT "WHO?":GOTO 20
  705 Y=PEEK(11879)+(PEEK(11880)*256)
  707 IF Y>60900 THEN PRINT"FILE OVER RUN ERROR": GOTO 20
  710 GOTO 650
 1000 DC=300: DT=0: DS=0: GOSUB 63000
 1010 RETURN
 9999 END
63000 IF DK=1 THEN GOTO 63020
63005 POKE 49666,0: POKE 49666,16: POKE 49666,0: DK=1
63010 FOR DZ=1 TO 500: NEXT DZ
63020 IF PEEK(49666) <> 217 THEN PRINT "ERROR": END
63030 WAIT 49671,128,128
63040 DF=0: IF DC>255 THEN DC=DC-256: DF=1
63050 POKE 49664,DF*128: POKE 49665,DC
63060 DT=((DT)  OR  (DF*128)): POKE 49664,DT
63070 FOR DE=0 TO 14: POKE 57344+DE,0: NEXT DE: POKE 57359,01
63080 IF DS=0 THEN DU=00: DV=16: DX=07: DY=37: GOTO 63130
63090 IF DS=1 THEN DU=07: DV=80: DX=14: DY=101: GOTO 63130
63100 IF DS=2 THEN DU=14: DV=144: DX=21: DY=85: GOTO 63130
63110 IF DS=3 THEN DU=21: DV=208: DX=28: DY=229: GOTO 63130
63120 IF DS=4 THEN DU=29: DV=16: DX=36: DY=37
63130 IF DD=0 THEN DV=DV+3
63140 POKE 49667,DV: POKE 49668,DU
63150 POKE 49669,DY: POKE 49670,DX
63160 POKE 49671,DD*64
63170 IF PEEK(49666) <> 217 THEN GOTO 63170
63180 DD=((DD*64)  OR  128): POKE 49671,DD
63190 WAIT 49671,128,128
63200 RETURN
```

~ ~ ~

Track Zero Writer

As the tech rep for OSI, I hear whatever the programmers in the field are screaming for. Some of the loudest screams have been for a method to modify track zero. So, with my time being as limited as it is and my personal need for a quick way to change track zero, I set out to find a solution. I was sitting in front of my Challenger III when the idea hit me. The disk copy utility found on every diskette had to be the solution. So, with all of that out of the way, here is how to do it. The changes are very minor and the copy program can still be used normally. The only operational difference is that "DONE" is no longer printed after a track zero restore. In order to modify track zero, follow this procedure. Call the disk copy program in to $0200 as usual. Then, go at $0200. After the message is printed, type an "E". Track zero will then be loaded into memory off the diskette. Now, instead of typing a control P, hit the space bar. The computer is now in the system monitor. Track zero is now in memory at $3200. That is to say, what normally would reside at $2200 is now at $3200. After you have made whatever modifications you desire, execute a "GO" at $031B. Track zero (in its modified form) will now be written to the diskette. Below is a step by step patch for the track zero writer.

First, call in extended monitor. Then, call in the disk copy program. Next, return to the extended monitor, then change the memory locations as listed below. Finally, save the modified program back on to the diskette. The track zero re-entry point is at $031B.

```
A*CØ2ØØ=Ø1,1          Ø32Ø/Ø5 Ø6
A*RE                  Ø321/ØD EA
                      Ø322/ØA EA
:#Ø679/7E 8Ø          Ø323/ØA EA
Ø67A/Ø6 31            Ø324/44 EA
                      Ø325/4F EA
:#Ø316/DØ FØ          Ø326/4E EA
Ø317/DC Ø3            Ø327/45 EA
Ø318/2Ø 6C            Ø328/2E EA
Ø319/49 FC            Ø329/ØØ EA
Ø31A/26 FE            Ø32A/4C 4C
Ø31B/2Ø 2Ø            Ø32B/ØØ ØØ
Ø31C/15 49            Ø32C/25 25
Ø31D/Ø6 26
Ø31E/2Ø 2Ø            :D
Ø31F/8B 15            A*SØ1,1=Ø2ØØ/5
```

9 Digit BASIC

There have been enumerable requests for an end-user 9 Digit BASIC. Therefore, this article's purpose is to present a concise method for modifying OSI's 9 Digit BASIC.

Normally, the machine code on Track 5 (address $34D5 when Track 5 is in memory) determines if the system is serial or video based. The I/O distributor is then set up accordingly. This code on Track 5 then jumps to cold start BASIC by jumping to $20E1.

The normal sequence of events then looks like this:

1) Is this a serial or video system?
2) Set up the I/O distributor according to the type of system that is being used (serial or video).
3) Jump to cold start BASIC (JMP $20E1).

Following the instructions contained within this article, Track 5 will be modified in the following manner:

Instead of determining if the system is serial or video, the I/O distributor is set up to input from memory without echoing anything to the screen.

Next the memory input pointer will be set up to point at the indirect command file which was loaded in as part of Track 4 (more on Track 4 shortly). The code on Track 5 then cold starts BASIC.

In summary, the modified code on Track 5 follows this sequence of events below:

1) Sets the I/O distributor to input from memory without echo.
2) Sets up the memory input pointer to point at the indirect command file (which has been made part of Track 4).
3) Jumps to cold start BASIC (JMP $20E1).

Now when BASIC has cold started and takes its first command, that command will come from the indirect command file. The indirect command file (INDCMD file) will "type" a "LOAD". It will then "type" a "L11". This will load Track 11 into BASIC's workspace. Track 11 contains the menu program which will be responsible for loading the user's selection. After Track 11 is loaded, the INDCMD file will "type" an "RB" (return to BASIC) and finally a "GOTO 61000". The menu program is now running and is in control.

Line 6100 in the menu program switches the I/O distributor to input from the keyboard and to output to the screen. Line 61000 then "RUNS" the menu program (that is, it goes to line number 1). Control C, Control O, LIST, NEW, and BASIC's immediate mode are then disabled by "POKES". The screen is then cleared and the menu is printed. After the user enters his selection, the corresponding track number is read from the DATA statement.

Notice that the DATA statement contains an "L" before the track number. This is required because the load command is in the form of "LTT" where TT is the track number. Now at this point, TRACK$ equals the appropriate track number. "RB GOTO 61000" is then "added" to TRACK$. This is so that when the I/O distributor is switched to input from the appropriate program will be loaded and then BASIC will "GOTO" line 61000 in the selected program.

To this point, then the user has selected his choice and TRACK$ equals the appropriate track number plus "RB GOTO 61000". All that remains to be done is to "PRINT" TRACK$ into memory and to switch the I/O distributor to input from memory. This then is the procedure - first, the memory output pointers are set up to point just beyond the first command file (it loaded the menu program). Then the I/O distributor is switched to output to memory. TRACK$ is then printed to memory. Now, the memory input pointer is set to point at what was just printed into memory. Finally, the I/O distributor is switched to input from memory without echoing to the screen.

The indirect command file (which was just printed into memory) "LOADS" the appropriate program and "GOES TO" line 61000 in it. Line 61000 in the "game" program swiches the I/O distributor to input from the keyboard and to output to the screen. Line 61000 then "RUNS" the game program (that is to say, it goes to the start of the program). When the "game" program is finished and it is time to reload the "MENU" program, the following steps must be taken:

First, the "game" program must set the memory input pointer to point at the first indirect command file (remember the file that originally "LOADED" the menu program). The I/O distributor must then be switched to input from memory without echoing it to the screen. Once this is done, the "MENU" program is reloaded and "RUN". At this point, the user may enter his next selection. One final note, the command file that "LOADS" the "MENU" program is brought into memory with Track 4. The assembly listings below show how to modify Track 4 and Track 5.

```
1 REM NORMAL VALUES LISTED BELOW
2 REM CNTRL - C =76 / CNTRL -O =255/ REDO FROM START =55,8
3 REM LIST =76/ NEW =78
10 POKE 2073,96: POKE 8981,0: REM   DISABLE CNTRL C & CNTRL O
20 POKE 2893,28: POKE 2894,11: REM   REDO FROM START
30 REM DIABLE LIST AND NEW
40 POKE 741,10: POKE 750,10
90 B=30
100 FOR SC=1 TO B: PRINT: NEXT SC
105 IF B=19 THEN FOR SC=1 TO 1000: NEXT SC
110 PRINT "OSI 9-DIGIT END USER SYSTEM":PRINT
120 REM LINES 120 - 999 FOR DIRECTORY ENTRIES
130 PRINT "1>DEMO"
```

```
1000 INPUT"ENTER THE NUMBER OF THE DESIRED GAME";G$
1002 GT=1
1005 IF VAL(G$)>GT THEN PRINT"INVALID SELECTION":B=19: GOTO 100
1010 G=VAL(G$): FOR X=1 TO G: READ TRACK$: NEXT X
1015 TRACK$=TRACK$+"RBGOTO61000"
1020 REM LINES 1020 - 1099 FOR DATA STATEMENTS
1030 DATA "L12": REM THIS DATA STATEMENT CONTAINS TRACK NUMBERS
1100 REM 1100 - 1200 PRINT THE INDIRECT COMMAND
1110 POKE 11860,150: POKE 11861,33: REM SET MEM INP. PNTR L&H
1120 POKE 8708,16: REM SWITCH OUTPUT TO OUTPUT TO MEM
1130 PRINT: PRINT "LOAD": PRINT TRACK$
1140 POKE 8708,128: REM SWITCH OUTPUT TO NON-ECHO
1150 POKE 11879,150: POKE 11880,33: REM SET MEM PNTR AT START OF FILE
1160 POKE 8707,8: REM SWITCH INPUT TO INOPUT FROM MEMORY
59999 END
61000 POKE 8707,1: POKE 8708,1: RUN

59999 REM THESE LINES MUST  BE ADDED TO THE "GAME" PROGRAMS
60000 POKE 8708,128: POKE 11879,128: POKE 11880,33: POKE 8707,8
60010 END
61000 POKE 8707,1: POKE 8708,1: RUN
61010 END


A*RA

INIZ?N
 A

   10 0000          ;          POWER UP OVERLAY FOR OSI 9-DIGIT BASIC
   20 0000          ;          BEFORE ASSEMBLY CALL TRACK 5 INTO $4200
   30 0000          ;          ALSO SET THE M COMMAND M1000
   40 0000          ;
   50 0000          ;
   60 0000          ;          EQUATES:
   70 0000          ;
   80 0000                    INFLAG=$2203
   90 0000                    OTFLAG=$2204
  100 0000                    INPNTL=$2E67
  110 0000                    INPNTH=$2E68
  120 0000                    CSTART=$20E1
  130 0000          ;
  140 34D5                    *=$34D5
  150 34D5          ;
  160 34D5 A980     INSERT    LDA #$80     ; SET MEM. INPUT PNTR. ADL.
  170 34D7 8D672E             STA INPNTL
  180 34DA 8D0422             STA OTFLAG   ; SET OUTPUT TO NON-ECHO
  190 34DD A921               LDA #$21     ; SET MEM. INPUT PNTR. ADH.
  200 34DF 8D682E             STA INPNTH
  210 34E2 A908               LDA #$08     ; SET INPUT FLAG TO INPUT FROM MEM.
  220 34E4 8D0322             STA INFLAG
  230 34E7 4CE120             JMP CSTART   ; GO CLOD START BASIC
  240 34EA EA                 NOP          ; FILL UP EXTRA
  250 34EB EA                 NOP
  260 34EC EA                 NOP
  270 34ED EA                 NOP
  280 34EE EA                 NOP
  290 34EF EA                 NOP
  300 34F0

.EXIT
01T
A*001

A*RA

INIZ?N
 A

   10 0000          ;          9-DIGIT INDIRECT FILE OVERLAY
   20 0000          ;          BEFORE ASSEMBLY CALL TRACK 4 INTO $4800
   30 0000          ;          ALSO SET THE M COMMAND M1000
   40 0000          ;
   50 0000          ;
   60 4180                    *=$4180
   70 4180          ;
   80 4180 0D       BEGIN          BYTE $0D,'LOAD',$0D
   80 4181 4C
   80 4182 4F
   80 4183 41
   80 4184 44
   80 4185 0D
   90 4186 4C                      BYTE 'L11'
   90 4187 31
   90 4188 31
  100 4189 52                      BYTE 'RB'
  100 418A 42
  110 418B 47                      BYTE 'GOTO 61000',$0D
  110 418C 4F
  110 418D 54
  110 418E 4F
  110 418F 20
  110 4190 36
  110 4191 31
  110 4192 30
  110 4193 30
  110 4194 30
  110 4195 0D
```

OS-65U PERFORMS

The New Standard in Micro Computer Operating Systems

System design goals: Create a simple, concise crash proof operating system which is easy for business programmers to utilize and simple for office workers (and other non-computerists) to use. The system must have the highest performance in the microcomputer industry and must be able to support present day floppy and hard disks as well as tomorrow's CCD and bubble memories without any user program modifications.

This may sound outlandish but we developed just such a system and here's how:

First, we started with a fresh copy of Microsoft's super fast 9 1/2 digit BASIC for the 6502. (This BASIC out benchmarks every other microcomputer BASIC using the 7 Kilobaud benchmarks except for our own ultra fast 6 digit BASIC.)

We knew that all operating system commands and features should be an integral part of this BASIC language so we put them right in the BASIC itself. This means that all OS features can be accessed in the immediate or command mode and as part of BASIC programs. All syntax such as file names can be literal strings or BASIC variables.

We started out with some simple but powerful extensions to BASIC to make the business system programmer happy like $L, $R, Input pound sign (D), and print pound sign (D). $L and $R are PRINT subcommands which automatically output numeric data in dollars and whole cents in neat colums just like "PRINT USING" only simpler and quicker.

The optional pound sign specifier in LIST, INPUT and PRINT statements allows the user to route I/O directly to the console, 16 RS-232 ports, a cassette port, RS-232 and parallel printer ports and word processing printers not to mention video displays and parallel keybords.

We then added a continuous memory file system (the real achievement of OS-65U). This file system has no tracks or sectors or records. The user simply allocates storage copacity to each file when he creates it. (On a CD-74 Hard Disk this can be over 72,000,000 bytes or characters.) The user can then directly address every entry in the file with no awareness of any block, sector or track structures. Data files can simultaneously contain strings and pure numeric data files can be accessed sequentially and randomly.

Data files are handled with standard syntax including OPEN "File", CLOSE (File), PRINT % (File) and INPUT % (File) and the very special INDEX (File). INDEX is a special BASIC variable/function which specifies the file address of the next entry to be input or output to that file. If you leave it alone, it operates sequentially, however, you can change it at any time to force a random access. This remarkable function can be on either side of a BASIC equation and can take on anny value within the storage range of an opened file. For example, all of the following are legal in OS-65U:

Index (1) = Index (1) + 10 (Causes 10 characters to be skipped)
B = Index (1) (Sets B=current index)
Index (3) = Index (8) /2 (Equates two file positions, useful in sorts and merges.)
Index (5) = A*50 (Sets up a random access on an array with 50 character elements)

Where (N) is a channel number or shorthand notation for an open file, and is assigned by the OPEN command.

This may seem exotic but it is really super simple and incredibly powerful. Besides your files always automatically revert to simple equential operation if you chose to ignore indexes.

And, finally, for those of you who would really hate to give up plain old sequential files, we added a FIND command. FIND searches for up to a 32 character string with optional "don't care" characters and will automtically scan any file from the beginning or other specified index. The FIND command is implemented in straight line page zero 6502 code (the fastest programming technique on the fastest micro) and searches files at over 250,000 bits per second.

Only three statements are needed to support a sequential file in a BASIC program; only four to support a random file. A mere seven statements are required to use an indexed sequential file system as part of a program!

A Benchmark: A Challenger III equipped with a CD-74 running OS-65U can access any account entry in a 500 account one million byte randomly ordered ledger file by an alphabetic key string up to 32 chracters long in less than 40 milliseconds (typically) using a simple two level ISAM file structure supported by a total program only 10 statements only. That's performance!

OS-65U also hosts multilevel passwords, elaborate error checking, programmable error recovery and end user miceties like warnings and automatic recovery when an "off" or non-existent peripherial is accessed. Programs and files in OS-65U can be fully secured such that they cannot be listed, copied or even accessed if desired.

OS-65U is available now for use on any Ohio Scientific floppy or hard disk based computer with 32K of RAM or more. At $199, it's quite possibly the best computer investment you'll ever make.

500/510 Breakpoint Utilities

How many times have you been debugging a program and wished you knew where it was when it "went away". Well, your troubles are over. The 500/510 breakpoint utilities allow you to halt the program wherever you desire. Upon halting, the program counter plus two is printed out, along with the flags and the contents of the accumulator, X-register, Y-register and the stack pointer! There are actually two modes of operation. The first uses the 6502 BRK instruction. Whenever the 6502 executes a BREAK instruction, several things happen. The 6502 fetches the new value of the program counter from $FFFE low and $FFFF high.

$FFFE and $FFFF are set in prom to point to $01C0. The breakpoint utilities set up a jump to the interrupt request entry point (IRQENT line number 1830). The program then decides if a IRQ or BREAK occured and either sets or clears the carry flag, respectively. All registers are saved on the stack and are available for modification using the following commands:

A - print the contents of A and opens A for modification
X - same as for A except deals with the X-register
Y - same as above but affects the Y-register
C - print the proccessor status word (PSW) and open it
(commercial at) - opens the program counter for modification
R - return to the command mode
G - go from the address set up using the commercial at command

In actual use one would load the breakpoint utilities and go at $3E00 using the 65A PROM monitor. The next step would be to place a BREAK command over top one of the instructions in the program being debugged. One would then "go" to the program under test and when the 6502 executes the BREAK instruction, the utilities program will be entered. The second mode of operation involves a slight hardware modification. On 510 boards, all the parts are already there and they can probably be found on a 500 CPU board.

The 6502 microprocessor chip has a very special pin called the SYNC pin. This pin goes high whenever the 6502 is fetching an instruction. Using the SYNC pin in conjunction with the interrupt request pin (IRQ), the utilities program allows one to trace program flow. Before each instruction is executed, the program counter, the flags, the registers (A,X,Y) and the stack pointer are printed on the screen. Typing any key halts the trace and a go resumes the trace. The modifications required are extremely straight forward. All that need be done is to take the SYNC pin's output, invert it (a spare NAND gate on the 510 CPU) and feed the inverter's output to the 6502's interrupt request pin (IRQ).

Don't forget when using the trace function to clear the interrupt disable flag by executing a CLI instruction. The following is a description of the various modules of the 500/510 utilities source. Lines 310 through 420 set up the interrupt request and non-maskable interrupt request vectors at $01C0 and $0130 respectively. Lines 460 through 590 are responsible for character input and output of a standard serial system. Lines 610 through 750 are responsible for inputting a character and either converting it to hex, or jumping to back to input another character if other than a legal hex character is entered.

Lines 770 through 910 determine which command has been entered as well as initializing the ACIA. Lines 930 through 980 are the go command section. This section restores the registers and then returns interrupt. 1010 through 1040 simply output a carriage return/line feed. 1060 through 1170 perform the "L" or LOAD command as per the 65A monitor. Lines 1180 through 1330 are responsible for the"P" or PRINT command. 1350 through 1460 input one hex byte and store it at the address pointed to by PNL, PNH, +X. 1480 through 1530 simply build an address at PNL, PNH +X. Halfby (1550 through 1610) converts the LSD in A to ASCII and outputs it to the screen.

1630 through 1720 PRTBYT prints a hex byte pointed to by PNL,PNH+Y. 1770 through 1940, these lines contain the NMI, IRQ and break entry points. Take special note of lines 1910 - 1940. It is here that the "+" or "*" prompter is determined via the carry flag. The "+" indicates the trace function and "*" indicates the break command. 1960 - 2410 output the prompter, the program counter, the flags, the registers and the stack pointer. Lines 2430 - 2500 determine if this is a break or a trace. A break returns to the control loop, while a trace cnecks for a key depression and enters the control loop if a key is down. If no key is down, the trace then executes the next instruction. Lines 2550 - 2650 output a byte pointed to by $100+X - OUTSP simply outputs a space.

Lines 2700 - 2770 are used to index to the proper register and to output the registers. XCMD2 - EXIT01 make the registers available for modification and do so if the user desires.

A couple of final notes, first this program is aimed at the small system owner and, therefore, resides in the top two pages of a 16K system. Secondly, this program is definitely not minimumized and we at OSI will be glad to see your suggestions and ideas.

```
A*RA

INIZ?N
.A

  10 0000          ;500/510 BRK PNT
  20 0000          ;
  30 0000          ;
  40 3E00                *=$3E00
  50 3E00          ;
  60 3E00          Z=IRQENT/256*256
  70 3E00          IRQADL=IRQENT-Z
  80 3E00          IRQADH=IRQENT/256
  90 3E00          ;
 100 3E00          ;
 110 3E00          ZZ=NMIENT/256*256
 120 3E00          NMIADL=NMIENT-ZZ
 130 3E00          NMIADH=NMIENT/256
 140 3E00          ;
 150 3E00          ;
 160 3E00          IRQVCL=$01C1
 170 3E00          IRQVCH=$01C2
 180 3E00          NMIVCL=$0131
 190 3E00          NMIVCH=$0132
 200 3E00          ;
 210 3E00          STKBAS=$0100
 220 3E00          JUMP=$4C
 230 3E00          ;
 240 3E00          ;
 250 3E00          ACIA=$FC00
 260 3E00          PNL=$FC
 270 3E00          PNH=$FD
 280 3E00          ;
 290 3E00          ;
 300 3E00          ;
 310 3E00 A907     BEGIN   LDA #IRQADL
 311 3E02          ;SET UP THE IRQ VCT
 320 3E02 8DC101           STA IRQVCL
 330 3E05 A93F             LDA #IRQADH
 340 3E07 8DC201           STA IRQVCH
 350 3E0A A901             LDA #NMIADL
 351 3E0C          ;SET UP THE NMI VCT
 360 3E0C 8D3101           STA NMIVCL
 370 3E0F A93F             LDA #NMIADH
 380 3E11 8D3201           STA NMIVCH
 390 3E14 A94C             LDA #JUMP
 391 3E16          ;SET UP THE JMP INSTR.
 400 3E16 8DC001           STA IRQVCL-1
 410 3E19 8D3001           STA NMIVCL-1
 420 3E1C 4C543E           JMP CONTRO
 430 3E1F          ;
 440 3E1F          ;
 450 3E1F          ;
 460 3E1F AD00FC   INCH    LDA ACIA
 461 3E22          ;INPUT CHARACTER
 470 3E22 4A               LSR A
 480 3E23 90FA             BCC INCH
 490 3E25 AD01FC           LDA ACIA+1
 500 3E28 297F             AND #$7F
 510 3E2A          ;
 520 3E2A 48       OUTCH   PHA
 521 3E2B          ;OUTPUT CHARACTER
 530 3E2B AD00FC           LDA ACIA
 540 3E2E 4A               LSR A
 550 3E2F 4A               LSR A
 560 3E30 90F9             BCC OUTCH+1
 570 3E32 68               PLA
 580 3E33 8D01FC           STA ACIA+1
 590 3E36 60               RTS
 600 3E37          ;
 610 3E37 201F3E   INHEX   JSR INCH
 611 3E3A          ;INPUT HEX DIGIT
 620 3E3A C952             CMP #'R
 630 3E3C F015             BEQ INHEXX
 640 3E3E C930             CMP #'0
 650 3E40 30F5             BMI INHEX
 660 3E42 C93A             CMP #':
 670 3E44 300B             BMI IN1
 680 3E46 C941             CMP #'A
 690 3E48 30ED             BMI INHEX
 700 3E4A C947             CMP #'G
 710 3E4C 10E9             BPL INHEX
 720 3E4E E906             SBC #6
 730 3E50 18               CLC
 740 3E51 290F     IN1     AND #$F
 750 3E53 60       INHEXX  RTS
 760 3E54          ;
 770 3E54 A903     CONTRO  LDA #$03
 771 3E56          ;INIT THE ACIA
 780 3E56 8D00FC           STA ACIA
 790 3E59 A9B1             LDA #$B1
 800 3E5B 8D00FC           STA ACIA
 810 3E5E D8               CLD
 820 3E5F 78       CONTR1  SEI
 830 3E60 D8               CLD
 840 3E61 207C3E           JSR CRLF
 850 3E64 201F3E           JSR INCH
 851 3E67          ;CMD. LOOP
 860 3E67 C94C             CMP #'L
 870 3E69 F01B             BEQ LOAD
 880 3E6B C950             CMP #'P
 890 3E6D F02E             BEQ PRINT
 900 3E6F C947             CMP #'G
 910 3E71 D006             BNE XCMDJ
 920 3E73          ;
 930 3E73 68       GO      PLA
 931 3E74          ;RESTORE Y
 940 3E74 A8               TAY
 950 3E75 68               PLA
 960 3E76 AA               TAX
 961 3E77          ;X
 970 3E77 68               PLA
 971 3E78          ;A
 980 3E78 40               RTI
 981 3E79          ;CC, PC
 990 3E79 4C8D3F   XCMDJ   JMP XCMD
1000 3E7C          ;
1010 3E7C A90D     CRLF    LDA #$D
1020 3E7E 202A3E           JSR OUTCH
1030 3E81 A90A             LDA #$A
1040 3E83 4C2A3E           JMP OUTCH
1050 3E86          ;
1060 3E86 20D43E   LOAD    JSR BUILD
1061 3E89          ;"L" COMMAND
1070 3E89 B0D4             BCS CONTR1
1080 3E8B A203             LDX #3
1090 3E8D A000             LDY #0
1100 3E8F 20BF3E   L01     JSR HEXBYT
1110 3E92 B0CB             BCS CONTR1
1120 3E94 91FC             STA (PNL),Y
1130 3E96 C8               INY
1140 3E97 D0F6             BNE L01
1150 3E99 E6FD             INC PNH
1160 3E9B 90F2             BCC L01
1170 3E9D          ;
1180 3E9D 20D43E   PRINT   JSR BUILD
1181 3EA0          ;"P" COMMAND
1190 3EA0 B0BD             BCS CONTR1
1200 3EA2 A000             LDY #0
1210 3EA4 A209     PR0     LDX #9
1220 3EA6 207C3E           JSR CRLF
1230 3EA9 CA       PR1     DEX
1240 3EAA F00B             BEQ PR2
1250 3EAC 20EE3E           JSR PRTBYT
1260 3EAF C8               INY
1270 3EB0 D0F7             BNE PR1
1280 3EB2 E6FD             INC PNH
1290 3EB4 4CA93E           JMP PR1
1300 3EB7 AD00FC   PR2     LDA ACIA
1310 3EBA 4A               LSR A
1320 3EBB B0A2             BCS CONTR1
1330 3EBD 90E5             BCC PR0
1340 3EBF          ;
1350 3EBF 20373E   HEXBYT  JSR INHEX
1351 3EC2          ;GET 1 HEX BYTE
1360 3EC2 B00F             BCS HEXX
1370 3EC4 0A               ASL A
1380 3EC5 0A               ASL A
1390 3EC6 0A               ASL A
1400 3EC7 0A               ASL A
1410 3EC8 95FC             STA PNL,X
1420 3ECA 20373E           JSR INHEX
1430 3ECD B004             BCS HEXX
1440 3ECF 15FC             ORA PNL,X
1450 3ED1 95FC             STA PNL,X
1460 3ED3 60       HEXX    RTS
1470 3ED4          ;
1480 3ED4 A201     BUILD   LDX #1
1481 3ED6          ;BUILD 2 BYTE ADDRESS
1490 3ED6 20BF3E           JSR HEXBYT
1500 3ED9 B004             BCS BUILDX
1510 3EDB CA               DEX
1520 3EDC 20BF3E           JSR HEXBYT
1530 3EDF 60       BUILDX  RTS
1540 3EE0          ;
1550 3EE0 18       HALFBY  CLC
1551 3EE1          ;PRINT HEX DIGIT
1560 3EE1 290F             AND #$F
1570 3EE3 0930             ORA #'0
1580 3EE5 C93A             CMP #':
1590 3EE7 9002             BCC HA0
1600 3EE9 6906             ADC #6
1610 3EEB 4C2A3E   HA0     JMP OUTCH
1620 3EEE          ;
1630 3EEE B1FC     PRTBYT  LDA (PNL),Y
1631 3EF0          ;PRINT ADDR (2 BYTES)
1640 3EF0 4A               LSR A
1650 3EF1 4A               LSR A
1660 3EF2 4A               LSR A
1670 3EF3 4A               LSR A
1680 3EF4 20E03E           JSR HALFBY
1690 3EF7 B1FC             LDA (PNL),Y
1700 3EF9 20E03E           JSR HALFBY
1710 3EFC A920             LDA #$20
1720 3EFE 4C2A3E           JMP OUTCH
```

```
3F01            ;
3F01            ;
3F01            ;
3F01            ;
3F01 48         NMIENT PHA
3F02            ;TRACE ENTRY
3F02 8A                TXA
3F03            ;SAVE A, X
3F03 48                PHA
3F04 38                SEC
3F05            ;SET TRACE FLAG
3F05 B010              BCS BREAK
3F07            ;
3F07 48         IRQENT PHA
3F08            ; BREAK, IRQ ENTRY
3F08 8A                TXA
3F09            ;SAVE A, X
3F09 48                PHA
3F0A BA                TSX
3F0B E8                INX
3F0C            ;POINT TO CC
3F0C E8                INX
3F0D E8                INX
3F0E 18                CLC
3F0F            ;SET BREAK FLAG
3F0F BD0001            LDA STKBAS,X
3F12 2910              AND #$10
3F14            ;ISOLATE B BIT
3F14 D001              BNE BREAK
3F16            ;IT WAS A BREAKPOINT
3F16 38                SEC
3F17            ;SET C AS THE IRQ FLG
3F17            ;
3F17 D8         BREAK  CLD
3F18 98                TYA
3F19            ;SAVE Y
3F19 48                PHA
3F1A 08                PHP
3F1B            ;SAVE ENT FLG IN C
3F1B 207C3E            JSR CRLF
3F1E 68                PLA
3F1F 48                PHA
3F20            ;GET ENT FLG (C)
3F20 2901              AND #1
3F22 092A              ORA #'*
3F24            ;* = BREAK, + = TRACE
3F24 202A3E            JSR OUTCH
3F27 BA                TSX
3F28 8A                TXA
3F29 6906              ADC #6
3F2B            ;6 + C(FROM OUTCH) = 7
3F2B AA                TAX
3F2C            ;NOW X POINTS TO PCH
3F2C            ;
3F2C 20783F            JSR OTKDX
3F2F            ;OUT PCH & DEX
3F2F 20783F            JSR OTKDX
3F32            ;"     PCL
3F32            ;OUTPUT COND. CODES (CC)
3F32 20883F            JSR OUTSP
3F35            ;SPACE
3F35 A007              LDY #7
3F37            ;8 CC BITS
3F37 BD0001            LDA STKBAS,X
3F3A            ;GET CC FROM STACK
3F3A 48                PHA
3F3B            ;SAVE ON TOP OF STK
3F3B 68         BREAK1 PLA
3F3C 0A                ASL A
3F3D            ;SHIFT CC BIT TO C
3F3D 48                PHA
3F3E B96D3F            LDA CCTBL,Y
3F41            ;GET CORREXP. LETTER
3F41 B002              BCS BREAK2
3F43            ;CC BIT WAS SET
3F43 A930              LDA #'0
3F45            ;CC  "    "  RESET
3F45 202A3E     BREAK2 JSR OUTCH
3F48 88                DEY
3F49 10F0              BPL BREAK1
3F4B 68                PLA
3F4C            ;CLR CC FROM STK
3F4C CA                DEX
3F4D            ;PNT TO A IN STK
3F4D            ;OUTPUT A, X, Y
3F4D 20753F     BREAK3 JSR OSTKDX
3F50            ;OUT " ", REG, DEX
3F50 C8                INY
3F51 C002              CPY #2
3F53 D0F8              BNE BREAK3
3F55            ;OUTPUT STK PTR
3F55 20883F            JSR OUTSP
3F58 8A                TXA
3F59            ;ADJUST BACK
3F59 6905              ADC #5
3F5B            ;(5+C=6) FOR STK CONT
3F5B 207B3F            JSR OADX
3F5E            ;OUTPUT STK PNTR
3F5E            ;BACK TO US OR BACK PROG.
3F5E 28                PLP
3F5F            ;GET ENT FLG OFF STK
3F5F B003              BCS GOTEST
3F61            ;TRACE
3F61 4C5F3E     CONTRJ JMP CONTR1
3F64            ;BREAK
3F64            ;
3F64 AD00FC     GOTEST LDA ACIA
3F67            ;STOP ?
3F67 4A                LSR A
3F68 B0F7              BCS CONTRJ
3F6A            ;YES -
3F6A 4C733E            JMP GO
3F6D            ;
3F6D 43         CCTBL  .BYTE 'CZIDB0VN'
3F6E 5A
3F6F 49
3F70 44
3F71 42
3F72 30
3F73 56
3F74 4E
3F75                   ;" ", BYTE @ STKBAS,X
3F75            ;
3F75 20883F     OSTKDX JSR OUTSP
3F78            ;SPACE
3F78 BD0001     OTKDX  LDA STKBAS,X
3F7B 48         OADX   PHA
3F7C 4A                LSR A
3F7D 4A                LSR A
3F7E 4A                LSR A
3F7F 4A                LSR A
3F80 20E03E            JSR HALFBY
3F83 68                PLA
3F84 CA                DEX
3F85 4CE03E            JMP HALFBY
3F88            ;
3F88 A920       OUTSP  LDA #'
3F8A 4C2A3E            JMP OUTCH
3F8D            ;REG. DISP./CHG. LOGIC
3F8D A004       XCMD   LDY #4
3F8F            ;NO. OF REGS - 1
3F8F BA                TSX
3F90 E8         XCMD1  INX
3F91            ;FIND REG NAME IN
3F91 D9D43F            CMP REGTBL,Y
3F94            ;TABLE & ADJUST X
3F94 F006              BEQ XCMD2
3F96            ;ALONG THE WAY
3F96 88                DEY
3F97 10F7              BPL XCMD1
3F99 4C5F3E            JMP CONTR1
3F9C            ;INVALID COMMAND
3F9C            ;
3F9C 20883F     XCMD2  JSR OUTSP
3F9F            ;SPACE
3F9F 98                TYA
3FA0            ;@ ?
3FA0 D004              BNE XCMD3
3FA2            ;NO -
3FA2 E8                INX
3FA3            ;YES -
3FA3 20783F            JSR OTKDX
3FA6            ;OUTPUT PCH
3FA6 20783F     XCMD3  JSR OTKDX
3FA9            ;OUT SELECTED REG, DEX
3FA9 E8                INX
3FAA            ;POINT BACK TO THAT REG
3FAA 20883F            JSR OUTSP
3FAD            ;SPACE
3FAD 20373E     XCMD4  JSR INHEX
3FB0            ;ANY INPUT ?
3FB0 B01F              BCS EXIT01
3FB2            ;BACK TO CONTR1 VIA JMP
3FB2 48                PHA
3FB3            ;YES - SAVE NEW DIGIT
3FB3 98                TYA
3FB4            ;TEST Y ZERO OR NOT
3FB4 697F              ADC #127
3FB6            ;AND SAVE RESULT IN V BIT
3FB6 68                PLA
3FB7 A004              LDY #4
3FB9            ;SHIFT NEW DIG INTO REG
3FB9 0A         XCMD5  ASL A
3FBA            ;1ST LJ IT IN A
3FBA 88                DEY
3FBB 10FC              BPL XCMD5
3FBD 3E0001            ROL STKBAS,X
3FC0            ;SHIFT INTO REG IN STK
3FC0 7005              BVS XCMD6
3FC2            ;NO -
3FC2 E8                INX
3FC3            ;YES - SHIFT INTO PCH
3FC3 3E0001            ROL STKBAS,X
3FC6 CA                DEX
3FC7 C0FC       XCMD6  CPY #$FC
3FC9            ;DONE ?
3FC9 D0EE              BNE XCMD5
3FCB            ;NO -
```

510 Tracer

The 510 tracer contains all the features of the 500/510 breakpoint utilities plus a few extras. Tracer also prints a disassemble of the next instruction to be executed. In addition, this program "swaps" out the zero page locations it requires and restores them upon returning to the main program. Two important points - first, this program uses the software processor select switch found only on Challenger III's and it resides at $5C00. This program can be used to trace another program by running the sync line on the 6502 through an inverter and then to the 6502's interrupt request line (IRQ LINE).

```
3010 3FCB A8            TAY
3011 3FCC        ;Y=0
3020 3FCC 50DF          BVC XCMD4
3021 3FCE        ;Y=0 & Y MATCHES
3030 3FCE C8            INY
3031 3FCF        ;Y=1
3040 3FCF D0DC          BNE XCMD4
3050 3FD1 4C5F3E EXIT01 JMP CONTR1
3051 3FD4        ;BACK TO CONTRL
3060 3FD4        ;
3070 3FD4 40     REGTBL  BYTE '@CAXY'
3070 3FD5 43
3070 3FD6 41
3070 3FD7 58
3070 3FD8 59

.S0
```

```
A*RE
:U5C00,5FF0
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
5C00    A9 1C 8D FE F2 A9 5D 8D FF F2 A9 16 8D FA F2 A9
5C10    5D 8D FB F2 A9 04 8D 01 F7 A9 EF 8D 00 F7 A9 00
5C20    8D 01 F7 A9 FF 8D 00 F7 4C 60 5C AD 00 FC 4A 90
5C30    FA AD 01 FC 29 7F 48 AD 00 FC 4A 4A 90 F9 68 8D
5C40    01 FC 60 20 2B 5C C9 52 F0 15 C9 30 30 F5 C9 3A
5C50    30 0B C9 41 30 ED C9 47 10 E9 E9 06 18 29 0F 60
5C60    A9 03 8D 00 FC A9 B1 8D 00 FC D8 78 D8 20 91 5C
5C70    20 2B 5C C9 4C F0 24 C9 50 F0 37 C9 47 D0 0F A2
5C80    FC BD 1B 5D 95 00 D0 F9 68 A8 68 AA 68 40 4C B3
5C90    5D A9 0D 20 36 5C A9 0A 4C 36 5C 20 E9 5C B0 CB
5CA0    A2 03 A0 00 20 D4 5C B0 E2 91 FC C8 D0 F6 E6 FD
5CB0    90 F2 20 E9 5C B0 B4 A0 00 A2 09 20 91 5C CA F0
5CC0    0B 20 03 5D C8 D0 F7 E6 FD 4C BE 5C AD 00 FC 4A
5CD0    B0 99 90 E5 20 43 5C B0 0F 0A 0A 0A 0A 95 FC 20
5CE0    43 5C B0 04 15 FC 95 FC 60 A2 01 20 D4 5C B0 04
5CF0    CA 20 D4 5C 60 18 29 0F 09 30 C9 3A 90 02 69 06
5D00    4C 36 5C B1 FC 4A 4A 4A 4A 20 F5 5C B1 FC 20 F5
5D10    5C A9 20 4C 36 5C 48 8A 48 38 B0 21 48 8A 48 98
5D20    48 A2 FC B5 00 9D 1B 5D E8 D0 F8 20 FF 5D 68 A8
5D30    BA E8 E8 E8 18 BD 00 01 29 10 D0 01 38 D8 98 48
5D40    08 20 91 5C 68 48 29 01 09 2A 20 36 5C BA 8A 69
5D50    06 AA 20 9E 5D 20 9E 5D 20 AE 5D A0 07 BD 00 01
5D60    48 68 0A 48 B9 93 5D B0 02 A9 30 20 36 5C 88 10
5D70    F0 68 CA 20 9B 5D C8 C0 02 D0 F8 20 AE 5D 8A 69
5D80    05 20 A1 5D 28 B0 03 4C 6B 5C AD 00 FC 4A B0 F7
5D90    4C 7F 5C 43 5A 49 44 42 58 56 4E 20 AE 5D BD 00
5DA0    01 48 4A 4A 4A 4A 20 F5 5C 68 CA 4C F5 5C A9 20
5DB0    4C 36 5C A0 04 BA E8 D9 FA 5D F0 06 88 10 F7 4C
5DC0    6B 5C 20 AE 5D 98 D0 04 E8 20 9E 5D 20 9E 5D E8
5DD0    20 AE 5D 20 43 5C B0 1F 48 98 69 7F 68 A0 04 0A
5DE0    88 10 FC 3E 00 01 70 05 E8 3E 00 01 CA C0 FC D0
5DF0    EE A8 50 DF C8 D0 DC 4C 6B 5C 40 43 41 58 59 20
5E00    91 5C BA D8 18 8A 69 07 AA BD 00 01 85 FE E8 BD
5E10    00 01 85 FF 4C 1F 5E 00 00 00 00 00 00 00 00 20
5E20    EA 5E A1 FE A8 4A 90 0B 4A B0 17 C9 22 F0 13 29
5E30    07 09 80 4A AA BD 1B 5F B0 04 4A 4A 4A 4A 29 0F
5E40    D0 04 A0 80 A9 00 AA BD 5F 5F 8D 1B 5E 29 03 8D
5E50    1C 5E 98 29 8F AA 98 A0 03 E0 8A F0 0B 4A 90 08
5E60    4A 4A 09 20 88 D0 FA C8 88 D0 F2 48 B1 FE 20 0F
5E70    5F A2 01 20 F6 5E CC 1C 5E C8 90 F0 A2 03 C0 04
5E80    90 F1 68 A8 B9 79 5F 8D 1D 5E B9 B9 5F 8D 1E 5E
5E90    A9 00 A0 05 0E 1E 5E 2E 1D 5E 2A 88 D0 F6 69 BF
5EA0    20 36 5C CA D0 EA 20 F4 5E A2 06 E0 03 D0 14 AC
5EB0    1C 5E F0 0F AD 1B 5E C9 E8 B1 FE B0 1D 20 0F 5F
5EC0    88 D0 F1 0E 1B 5E 90 0E BD 6C 5F 20 36 5C BD 72
5ED0    5F F0 03 20 36 5C CA D0 D2 60 20 03 5F AA E8 D0
5EE0    01 C8 98 20 0F 5F 8A 4C 0F 5F 20 91 5C A5 FF A6
5EF0    FE 20 E3 5E A2 02 A9 20 20 36 5C CA D0 F8 60 AD
5F00    1C 5E 38 A4 FF AA 10 01 88 65 FE 90 01 C8 60 48
5F10    4A 4A 4A 4A 20 F5 5C 68 4C F5 5C 40 02 45 03 D0
5F20    08 40 09 30 22 45 33 D0 08 40 09 40 02 45 33 D0
5F30    08 40 09 40 02 45 B3 D0 08 40 09 00 22 44 33 D0
5F40    8C 44 00 11 22 44 33 D0 8C 44 9A 10 22 44 33 D0
5F50    08 40 09 10 22 44 33 D0 08 40 09 62 13 78 A9 00
5F60    21 01 02 00 80 59 4D 11 12 06 4A 05 1D AC A9 AC
5F70    A3 A8 C1 D9 00 D8 00 00 00 1C 8A 1C 23 5D 8B 1B
5F80    A1 9D 8A 1D 23 9D 8B 1D A1 00 29 19 AE 69 A8 19
5F90    23 24 53 1B 23 24 53 19 A1 00 1A 5B 5B A5 69 24
5FA0    24 AE AE A8 AD 29 00 7C 00 15 9C 6D 9C A5 69 29
5FB0    53 84 13 34 11 A5 69 23 A0 D8 62 5A 48 26 62 94
5FC0    88 54 44 C8 54 68 44 E8 94 00 B4 08 84 74 B4 28
5FD0    6E 74 F4 CC 4A 72 F2 A4 8A 00 AA A2 A2 74 74 74
5FE0    72 44 68 B2 32 B2 00 22 00 1A 1A 26 26 72 72 88
5FF0    C8 C4 CA 26 48 44 44 A2 C8 24 24 24 24 24 24 24
```

Do you HAVE THESE IMPORTANT publications FROM OSI?

*Full Line Catalog** ✓

This is the "complete" catalog - every OSI product is described in full. We even include articles, such as "An Introduction to Small Computers" in our Fall 77 issue. Our no nonsense approach allows you to get the facts. WE WANT YOU TO GET THE FACTS because we want you to know what you're buying. Who knows, when you're done reading our catalog you might have learned something.

Comprehensive Information Package

Designed for reference and service it contains all the technical information you need. 64 PAGES including

PARTS LIST
BOARD DESCRIPTIONS
EXTENSIVE SCHEMATICS

You can work your way up from the bare board to the system you want to configure. PLUS you get the Full Line Catalog with its complete product descriptions.

1977 Small Systems Journal Back Issues

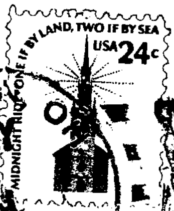
DID YOU MISS THE JOURNAL IN 1977? Now you can catch up on all that good reading you missed. For only $6.00 you can find out about "The Auto-Load Cassette System", "Understanding and Using the 6502 Assembler", "Getting the Most Out of BASIC", "Constructing a Fool-Proof End User System" plus much more. Start your collection from the beginning.

1978 Small Systems Journal

In its first six issues the Journal established itself as a publication dedicated to the serious exploration of microcomputer technology. If you want to continue the exploration send in now for your 1978 bi-monthly subscription. If you are not a subscriber now is the time for you to pick up on what the Small Systems Journal has to offer. Enjoyable reading that keeps you informed about what's going on in the small computer world. If you've missed the first six, don't miss number seven.
SUBSCRIBE NOW!

Ship To
NAME____________________
ADDRESS__________ PHONE__________
CITY__________ STATE________ ZIP__________
CATALOG $1. ☐ 77 JOURNAL $6. ☐
INFO PACKAGE $5. ☐ 78 JOURNAL $6. ☐
MASTERCHARGE ☐ No.__________ Exp.______
BANKAMERICARD ☐ No.__________ Exp.______
OHIO SCIENTIFIC 1333 S. CHILLICOTHE AURORA OH. 44202

*(SPRING EDITION AVAILABLE APRIL 15)



AKRON OH
MAY 3 PM 1978
443

OHIO SCIENTIFIC
11679 HAYDEN STREET HIRAM, OHIO 44234

SMALL SYSTEMS JOURNAL

Ohio Scientific 1333 S.Chillicothe Aurora, OH 44202