August 1977 $1.50

ohio scientific's
SMALL SYSTEMS JOURNAL VOLUME 1 NO. 2

features

departments

The magazine for 6502 computer enthusiasts!

Ohio Scientific advances the state-of-the-art of small computers.

- **Challenger II with our ultra-fast 8K BASIC in ROM. Now you can own a computer with full BASIC and plenty of user workspace for as little as $298.00. And the BASIC is there the instant you turn the machine on!**
- **Challenger III is the remarkable computer which has 6502A, 6800, and Z-80 processors. This computer system allows you to run all software published in the small computer journals, yet, it costs only about 10% more than comparable single processor computers.**
- **Challenger Single and Dual Drive Floppy Disks. These full size floppy disks are available in kit form or assembled at about the same prices as our competitors' mini floppies. Yet, they store three times as much data as the minies.**
- **Ohio Scientific's new 9 digit precision business BASIC is only slightly slower than our ultra-fast 8K BASIC. Still faster and more powerful than anyone else's 6 digit precision BASIC.**
- **Our incredible new 74 million byte disk drive. That's right, 74 million bytes is available for as little as $6,000.00 complete with interface for any Ohio Scientific computer. This new disk is quite possibly the world's highest performance data storage device. It features an unbelievable 34 milli-second average access time and an ultra-fast data transfer rate.**
- **Now is the time for you to dump your 1974 design vintage S-100 computer and move up to the state-of-the-art!**

For more specifics, send $1.00 for our new Fall Catalog.

OHIO SCIENTIFIC

11679 Hayden
Hiram, Ohio 44234

Introduction

This issue of Ohio Scientific's Small Systems Journal will bring you answers to many of your most urgent questions on our latest products. To all of you who have been eager to have a way to protect your carefully written programs from damage due to the actions of uninitiated keyboard operators, we refer you to our story on the end user system modifications on page 9. There you will see just which contents to change in which locations, in order to add a foolproof safeguard to your programs using the OS-65D. If you have been perplexed by the various mass storage devices available, you will be pleased to see a clear-cut breakdown of the advantages and disadvantages of each of the popular media in our article (part one of a three-part series) on page 15. As a postscript to this article, we are presenting an enlightening description of our new 74 million-byte disk drive for all those of you who have an interest in it. Our opening story offers you certain pointers on the PEEK and POKE functions in BASIC, a feature many of you have asked about. Here you will find a means to add considerably to your system's capabilities. You should find our descriptions of the Hamurabi and Mini-Graphics programs both amusing and challenging. In particular, in this issue, we are giving notice of a revised customer service policy below.

The splendid response from you, our readers, to the first issue of Ohio Scientific's Small Systems Journal was heartening, to say the least. We are endeavoring to live up to your expectations, and cordially invite you to let us hear from you. Send all correspondence and articles for publication to:

Ohio Scientific, Inc.
Small Systems Journal
Box 36
Hiram, OH 44234

New policy for Customer Service

OSI now has several thousand hobbyist customers. Most of these customers are very reasonable people who realize that we must make a profit (however small) to stay in business. However, we are getting (as can be expected) a small number of customers who expect the unreasonable, particularly in the area of repair and trouble-shooting service of bare boards which they purchase from us, and populate from their junk boxes and the "surplus" component vendors. This situation has forced us to specify a new customer service policy.

Bare Boards and Manuals

Boards are sold at a very low profit margin, which does not support customer service. Furthermore, since our fully assembled products are so inexpensive, bare boards should only be considered for custom applications. Since we have no control over the quality of the parts used on the board, we cannot provide service in repairing the board. In other words, when you buy a board, you are on your own!

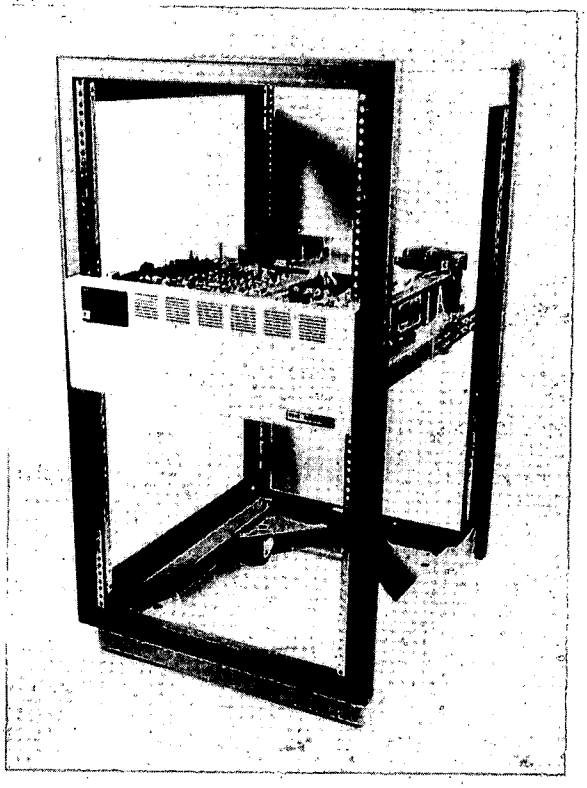
Kits

Ohio Scientific uses only 100% pre-tested components in kits, which should facilitate trouble-free assembly. We are willing to answer specific questions concerning trouble-shooting problems phoned in during business hours. We cannot afford to spend time jawboning about your computer system or OSI's latest developments.

Generally, if the problem cannot be cleared up in one or two telephone calls, we recommend that you send the board back for repair. Repair charges are $15.00 per hour for labor. We charge only for components damaged by improper hook-ups or other user induced failure.

Fully Assembled Computer and Boards

OSI fully-assembled products carry a 60-day limited warranty on materials and workmanship and a 1-year limited warranty on components. This warranty does not apply to items such as tape recorders and video monitors which are not manufactured by OSI. We recommend that the user contact OSI and discuss the problem before sending the unit back for repair. Generally, we would prefer that the user not attempt a repair himself. Out-of-warranty repair charges are currently $15.00 per hour and parts (where applicable). All returns require a return authorization number. OSI cannot be responsible for any items received without an R.A. number.



The Model C-D74 is the newest disk available from Ohio Scientific. It features a 74-million byte storage capacity with an average access time of 35 milliseconds. The unit is shown here mounted in a standard equipment rack (22" deep), which OSI will offer at a later time. A general description of Ohio Scientific's new big disk begins on page 11.

Part I

Get the most out of 8K BASIC

BASIC's performance and versatility can reach far beyond what is implied in the 8K BASIC User's Manual by effective use of the machine language reference instructions PEEK, POKE, and USR(X). In this article we will discuss the use of the PEEK and POKE instructions to program I/O operations on standard Ohio Scientific systems.

The POKE command can be executed in immediate or program mode of operation. The syntax is: POKE I,J where I is a location (decimal) between Ø and 65,535 and J is a value (decimal) between Ø and 255. The location I specifies the actual memory or I/O location which will be POKEd. The value J is a numerical value which will be inserted into that location. Numbers used in this function are automatically truncated to integers. If numbers out of the range specified are utilized, an error message will occur. The arguments of the POKE command I,J may be numbers, variables, or expressions.

The PEEK function should only be executed in program mode and must be equated to an expression or variable. The proper syntax is X=PEEK(I) where X is a variable and I is a location (decimal) between 0 and 65,535, and again, may be a numeric value, variable, or expression.

It is totally safe to use the PEEK function. However, extreme caution must be used with the POKE command, since it is possible for the user to inadvertently POKE a location in BASIC or the operating system, and effectively "POKE his system to death," causing a crash, and requiring a complete reload of the computer.

PEEK and POKE in conjunction with support FOR-NEXT loops, and other operations, can be executed at the rate of about 100 operations a second. That means that PEEK and POKE can be used to perform I/O operations adequately fast for human input, such as keyboard typing, and servicing of mechanical devices, such as relays, etc. PEEK and POKE operations are not fast enough to service high-speed I/O operations. For that, the USR(X) function must be used in conjunction with an Assembler language or machine code subroutine. We will discuss the servicing of high-speed I/O with the USR(X) function in a later issue of the journal.

To effectively utilize PEEK and POKE, one must know the locations in memory corresponding to the I/O ports of the computer. Since it is more common to use these I/O locations with Assembler and machine code, all standard documentation specifies these locations and the values residing there in hexadecimal, rather than decimal, notation. Therefore, the I/O programmer in BASIC must be aware of the method of conversion between these two notations.

The BASIC program on the following page performs the conversion of decimal to hexadecimal and vice versa with a numeric range adequate for PEEK and POKE functions. Following the program is a sample execution. This program can be used in conjunction with any of our kit manuals to convert the hexadecimal addresses and desired values into decimal for use in BASIC. The common OSI system ports which we will discuss are listed in the following table. This is not intended to be a complete list of I/O ports on standard OSI systems, but it does present a majority of ports usable with a low-to-medium speed I/O handler.

430 I/O Board

	hex	dec	
In	FB01	64257	in only
Out	FB02	64258	out only
A/D	FB00	64256	in only
D/A	FB00	64256	out only
D/A	FB01	64257	out only

Ports of Interest

PIA	hex	dec
500 CPU	F700	63232
510 CPU	F700	63232
450 PROM	F000	61440
455 PROM	EF00	61184

ACIA at FC00

440 Video Board

	hex	dec
Alpha	D000-D3FF	53248-54271
Graphics	D400-DBFF	54272-56319
Keyboard	DFFF	57343

The simplest I/O ports to service are those on the 430A and 430B Super I/O Board. This board can optionally have an eight-line parallel input located at FB01 (hex) and an eight-line parallel output at FBØ2, an A/D converter at FBØØ, a D/A converter at FBØØ, and a D/A converter at FBØ1. The ports on the 430 Board are uni-directional, i.e., there is an output address and an input address for the same location. That is why an A/D and a D/A converter share the same address. One is a Read Only Port, and the other is a Write Only Port.

The 430's parallel input can be simply read with a BASIC line such as:

```
2ØØ      X=PEEK(64257)
```

When this line is executed, the current bit patterns present on the pins of the parallel input port will be transported to the variable X in the BASIC program.

Outputs are serviced or generated by the POKE statement, for instance:

```
21Ø      POKE 64258,X
```

This will place the binary pattern corresponding to the value of X on the output port at FBØ2, which is latched so that that value will be maintained until it is changed by the program. By simply adding an additional line, such as:

```
22Ø      GO TO 2ØØ
```

you continuously monitor the input at FBØ1, and place these results at FBØ2. The user can acquire the digital value of an analog input if an A/D converter is present by simply reading the A/D converter located at FBØØ (hex) 64256 (dec) in the same manner. Analog outputs can be generated by using the D/A converter at FBØØ and optionally the second D/A conver-

```
10 REM HEX-DECIMAL CONVERSION
20 REM
30 DIM CH$(16)
32 PRINT "  --BASE 10, BASE 16 CONVERSIONS--" : PRINT : PRINT
40 FOR I=1 TO 10 : CH$(I)=CHR$(I+47) : NEXT I
50 FOR I=11 TO 16 : CH$(I)=CHR$(I+54) : NEXT I
60 INPUT "CONVERT TO HEX OR DEC";A$
61 IF A$<>"HEX" AND A$<>"DEC" THEN GOTO 60
62 PRINT
65 INPUT "THE NUMBER";NUM$
70 IF A$="DEC" THEN GOSUB 1000 : GOTO 90
80 IF A$="HEX" THEN GOSUB 2000
90 IF FLAG=0 THEN PRINT "IS";C$;" IN ";B$
100 GOTO 62
1000 B$="DECIMAL"
1005 FLAG=0
1010 ANS=0
1020 FOR I=1 TO LEN(NUM$)
1030 FOR J=1 TO 16
1040 IF CH$(J)=MID$(NUM$,I,1) THEN GOTO 1070
1050 NEXT J
1060 PRINT "ILLEGAL CHARACTER!" : FLAG=1 : RETURN
1070 ANS=ANS*16+J-1
1080 NEXT I
1085 C$=STR$(ANS)
1090 RETURN
2000 B$="HEXDECIMAL"
2010 FLAG=0
2020 C$=" "
2025 CC$=""
2027 IF LEN(NUM$)>5 THEN FLAG=1 : PRINT "TOO LARGE" : RETURN
2028 FOR I=1 TO LEN(NUM$) : D$=MID$(NUM$,I,1)
2029 IF D$<"0" OR D$>"9" THEN PRINT "ILLEGAL CHAR." : FLAG=1 :RETURN
2030 NEXT I
2035 V=VAL(NUM$)
2040 D=4096
2050 FOR I=1 TO 4
2060 ANS=INT(V/D)
2065 V=V-ANS*D
2070 D=D/16
2080 CC$=CC$+CH$(ANS+1)
2090 NEXT I
2100 FOR I=1 TO 4
2105 II=I
2110 IF MID$(CC$,I,1)<>"0" THEN GOTO 2130
2120 NEXT I
2130 C$=C$+MID$(CC$,II)
2140 RETURN
2150 END
OK
```

```
RUN
  --BASE 10, BASE 16 CONVERSIONS--

CONVERT TO HEX OR DEC? HEX

THE NUMBER? 16684
IS 412C IN HEXDECIMAL

THE NUMBER? 129
IS 81 IN HEXDECIMAL

THE NUMBER?

OK
```

```
RUN
  --BASE 10, BASE 16 CONVERSIONS--

CONVERT TO HEX OR DEC? DEC

THE NUMBER? F700
IS 63232 IN DECIMAL

THE NUMBER? FC00
IS 64512 IN DECIMAL

THE NUMBER? D223
IS 53795 IN DECIMAL

THE NUMBER?

OK
```

ter at FB01. These outputs are latched just as the parallel output is, so that the D/A will acquire the analog output value associated with its digital input by simply POKE-ing the desired value into that location. That value will remain there until changed by the program. Most applications of A/D and D/A's, such as voice recording, voice generation, music generation, etc., require high-speed servicing of these I/Os, but, there are some low-speed applications, such as driving a chart recorder, monitoring weather instruments, or driving small motors at different speeds, that can be serviced by the low-to-medium speed operation of BASIC.

The standard parallel interface used by the 6502 and 6800 systems is the PIA (peripheral interface

adapter). Several manufacturers produce this part as a 6520, 6820, 6821, or any of various other versions of the chip. The PIA has two eight-line bi-directional ports, plus additional hand-shaking lines and interrupt control lines. Each of the 16 I/O lines on the PIA can be an input or an output under program control. One eight-line port is referred to as the A-Port, and the other as the B-Port.

PIA Initialization

There is a total of six internal registers in the PIA, four of which must be set up before the PIA can be used for any specific tasks. This setup is called PIA initialization. In a dedicated application such as an industrial controller, the PIA would be set up on system power-up by code stored in a PROM or ROM. In a general purpose computer, the setup of undedicated ports is the responsibility of the user. Thus the user must set up the PIA, or initialize it, whenever he wants to use it.

The following program in BASIC is a PIA exerciser containing a general purpose subroutine starting at line 1000 for initializing PIAs for user programs. There are as stated above, six internal registers in the PIA, but only four can be accessed as memory locations at a given time. These registers normally occupy four consecutive memory locations. We consider the address of the PIA to be X, as it is stated in the program. Thus location X is the address of the peripheral interface register A, or its corresponding data direction register. Address X+1 is the status register in conjunction with the A-Port of the PIA. A bit in the status register specifies whether the data direction register of the peripheral interface register is available to the bus location. X+2 is the address of the peripheral interface register or its corresponding data direction register as specified by a bit in its corresponding control, or status register at location X+3. A logical 1 in a bit location in the data direction register specifies that the corresponding pin on its port will be an output. A Ø in that location specifies that that pin will be an input.

The BASIC program further demonstrates the use of the PIA in a simple application. The program goes to line 1000, which inputs the base address of the PIA into the variable X. It then asks if the A-side will be an input or an output, asks the same of the B-side, and stores the responses as strings A$ and B$. It then performs an initialization of the PIA by setting the PIA's control registers to Ø. These control registers are located at X+1 and X+3. By setting these registers completely to Ø, the data direction registers for Ports A and B are accessible at addresses X and X+2, respectively. The user must then specify whether he wants the ports to be inputs or outputs. Line 1040 will set the data direction register lines all low if A$=I, or if the user specifies I for input. Otherwise, it sets the data direction register lines all high, specifying that Port A will be an output (see line 1045). Lines 1050 and 1055 perform the same function for Port B. Line 1060 then POKEs Ø4 into the control register for both Port A and Port B, located at X+1 and X+3. This switches the peripheral interface registers into addresses X and X+2. The peripheral interface registers are the registers which actually input or output data to the pins on the port once the PIA is configured by the subroutine. This subroutine at 1000 to 1070 thus initializes the PIA by setting the control registers to Ø, then specifies the data direction of both Port A and Port B, and finally restores the peripheral interface register to addresses X and X+2, so that the PIA can be used as a simple I/O device.

The mainline program is then at lines 30 to 260. The program then asks whether you would like to work with Port A or Port B. It checks on the entry at A$ and B$ to find out whether A or B is an input or an output. If the port you have selected is an input, it then reports the current value on the input pins. If the port you select is an output, it asks you for the bit pattern variable K, and then outputs that bit pattern. When the output occurs, it is latched by the PIA and will not change until the program changes it or the PIA is reset. One important feature of PIAs is that they must be reset on, or immediately after, power-up by a master reset and then configured by software. The normal configuration for a PIA immediately after power-up shows all lines as inputs, having generally drifted to a logical high state. Thus if the PIA is read immediately after initialization, with nothing connected to its inputs, it will generally report all highs, and the data direction registers will specify that all lines are inputs. This program is very useful for test-

```
5 REM  PIA INITIALIZATION SUBROUTINE AT 1000
10 GOSUB 1000
20 INPUT "SIDE (A OR B)";C$
30 IF C$="A"GOTO 100
40 IF C$="B"GOTO 200
50 GOTO 20
100 IF A$="I"GOTO 150
110 INPUT "OUTPUT TO A";K
120 POKE X,K
130 GOTO 20
150 PRINT"INPUT TO A IS ";PEEK (X)
160 GOTO 20
200 IF B$="I"GOTO 250
210 INPUT "OUTPUT TO B";K
220 POKE X+2,K
230 GOTO 20
250 PRINT"INPUT TO B IS ";PEEK (X+2)
260 GOTO 20
1000 INPUT "STARTING ADDRESS OF PIA";X
1010 INPUT "A SIDE I OR 0";A$
1020 INPUT "B SIDE I OR 0";B$
1030 POKE X+1,0:POKE X+3,0
1040 IF A$="I" THEN POKE X,0:GOTO 1050
1045 POKE X,255
1050 IF B$="I" THEN POKE X+2,0:GOTO 1060
1055 POKE X+2,255
1060 POKE X+1,04:POKE X+3,4
1070 RETURN
OK
```

ing PIA ports in various portions of your system, and can also be used to test I/O devices which you connect to the PIA.

A simple test procedure for the PIA output is to connect a digital voltmeter or VOM between ground and the output pin, and via this program toggle that pin (or the entire port, for that matter) high and then low, and observe the voltage swing on the voltmeter. One PIA port can be used as a signal source to check inputs of another PIA port. For example Port A can be configured as an output, and then alligator clips from its connector can be joined to Port B which could then be configured as an input. You could then attempt to write a bit pattern out on Part A and observe the same bit pattern coming back on Port B. These ports, of course, can be used for simple applications such as a switch register and light register for games.

The PIA actually has many more features than are used here. For example, the 6820 PIA has complete interrupt controls for both Ports A and B plus handshaking lines and several other status bits in its control registers. A complete description of the PIA operation is in the Motorola M6800 Microcomputer System Design Data Book.

Serial Port Programming

It can be very advantageous to be able to program inputs from a serial port. The standard (6850) ACIA port on the 400, 500, or 510 CPU Board is located at address FCØØ. It is very simple to read the 6850 ACIA in BASIC for specialized input applications, particularly when an input character is desired but cannot be obtained through the normal INPUT statement. For example, no control characters are echoed into BASIC on the normal INPUT statement. They must be brought in by a special input statement. The following program is an example of a programmed input from the ACIA port. The output of this program is both the decimal value of the ASCII code inputted, as well as the actual ASCII code. It is necessary to disable the Control-C break-test function on the input routine if the ACIA port is also being used as the system input port, i.e., if you are using the same serial terminal to execute the programs in BASIC as you are using to run this program.

```
LIST

 5 Y=64512
 10 X=PEEK(Y)
 20 IF X<131 GOTO 10
 30 Z=PEEK(Y+1)
 40 PRINT Z,CHR$(Z)
 45 IF Z=24 THEN STOP
 50 GOTO 10
OK
LOAD
01T
A*N01
A*RB
OK
RUN
 212            T
 72             H
 201            I
 83             S
 160
 201            I
 83             S
 160
 65             A
 160
 212            T
 197            E
 83             S
 212            T
 24

BREAK IN  45
OK
LOAD
01T
A*N81
A*RB
OK
```

This is necessary because the Control-C test will interfere with your normal program input and will cause this program to miss characters sometimes. It is easy to disable a Control-C function on disk systems under OS-65D. Change the I/O Distributor Input Flag from 81 to Ø1 by using an NØ1 command in the DOS.

Unfortunately, it is impractical to attempt to disable the Control-C in paper tape systems, and of course, impossible in ROM versions. There a USR(X) function will be necessary to perform input character functions. This program assumes that the ACIA has already been initialized by the operating system. If another ACIA is used, then the user must also initialize the port. For a complete description of the ACIA operation, consult the Motorola M6800 manual, as mentioned above. Note that in the operation of the program, Control-C was disabled by the statement A*NØ1, and then after the run, it was enabled by the statement A*N81. Also note that the program has a test of whether the input is 24. Line 45 in the program is a test for Control-X, which allows you to exit the program. If this line were not present there would be no way to get out of the program without resetting the computer.

440 Video Display

The 440 Video Display is very easily programmed for interesting displays via the POKE instruction in BASIC. This allows you to write customized CRT routines as well as to perform a random access of the display for video games, etc. The following program is a simple routine that draws a box on the screen. The

```
 10 Y=53496
 20 A=40
 40 F(1)=-1
 42 F(2)=32
 43 F(3)=1
 44 F(4)=-32
 50 FOR Z=1 TO 4
 60 FOR X=1 TO 20
 90 POKE Y,A
 100 Y=Y+F(Z)
 110 NEXT X
 120 NEXT Z
 130 A=A+1
 140 IF A>255 THEN A=0
 150 GOTO 50
OK
```

box rotates counter-clockwise by changing the character used to construct the box. This simple example shows how to draw lines of characters on any of four axes. Diagonal lines can easily be drawn by incrementing or decrementing the address pointer by 31 or 33 instead of 32. Incrementing or decrementing by different values will yield different slopes on diagonal lines. A moving pointer can be easily generated by placing the pointer on the screen, then writing its new location and replacing its old location with a space after it moves. BASIC is fast enough to produce high-speed video displays on alphabetic screens, but not quite fast enough to produce real-time graphics, because many more bit-level operations are required to produce a dot within the graphics memory; also, the graphics memory is much larger, being 128 dots by 128 dots.

The next example is a modification of the box program to produce a more elaborate pattern. By adding statements which modify

pointers each time through the loop, this same BASIC program can produce a wide variety of interesting repetitive patterns on the screen.

```
LIST

 10 Y=53496
 20 A=40:B=0
 30 G=1:H=20
 40 F(1)=-1
 42 F(2)=32
 43 F(3)=1
 44 F(4)=-32
 50 FOR Z=1 TO 4
 60 FOR X=G TO H
 90 POKE Y,A
 100 Y=Y+F(Z)
 110 NEXT X
 120 NEXT Z
 127 H=H-1
 128 IF H>G GOTO 50
 130 B=B+.5
 132 IF A=32 THEN A=B:GOTO140
 134 A=32
 140 IF A>255 THEN A=0
 150 GOTO 30
OK
```

It is possible under some circumstances to input characters by a simple PEEK statement by means of the 440 keyboard. This is totally dependent on the actual configuration of the keyboard itself. The following example is a program that inputs a character from the 440 keyboard and displays it in reverse order on the 440 screen for a total of 20 characters across. This program will only work if you have a momentary keyboard strobe long enough to be caught by the polling routine of lines 10 and 15. Continuous strobe keyboards and keyboards with a short pulse will not work satisfactorily with this program. They will require a USR(X) subroutine for proper operation.

```
 5 Y=53496
 10 X=PEEK(57343)
 15 IF X>127 GOTO 10
 20 POKE Y,X
 30 Y=Y-1
 40 IF Y>53476 GOTO 10
 50 GOTO 5
OK
```


Ohio Scientific's Small Systems Journal is published monthly by Ohio Scientific Inc., P.O. Box 36, Hiram, Ohio 44234. The subscription rate is six dollars for six issues. Individual copies are $1.50. Published at Twinsburg, Ohio, by the Twinsburg Bulletin.

Vol. 1. No. 2	August 1977
Editor-in-Chief	Gary Deckant
Production Manager	Rob Spademan
Contributing Editors	Mike Cheiky
	Eric Davis
	Marcel Meier
Production Assistant	Cindy Warrick

Dumping Memory

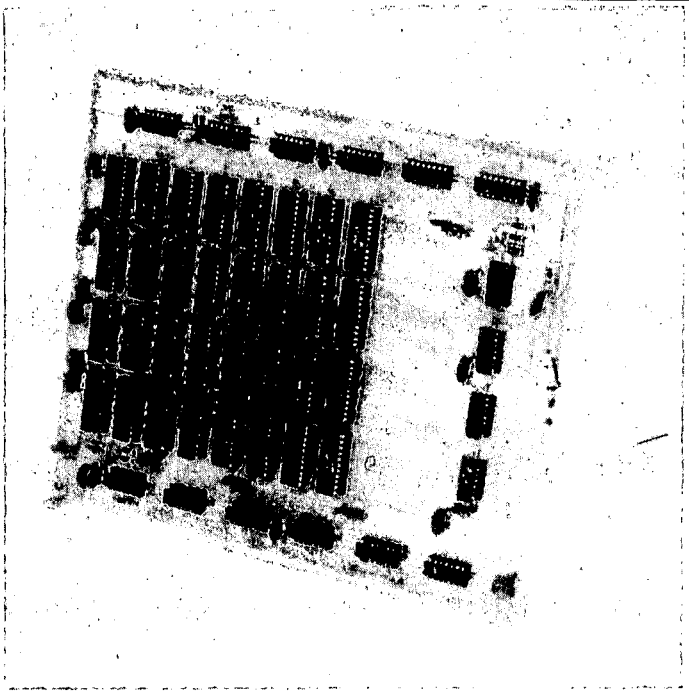
The PEEK and POKE functions are also very useful for examining, changing, and testing memory. For example, it is very simple to write memory test programs which are somewhat slow but nevertheless are conveniently generated with the PEEK and POKE functions. It is also possible to write memory compare programs, checksum programs, and a machine level monitor similar to Ohio Scientific's Extended Monitor in BASIC. A simple example of a program is shown here. This program lists memory within the range specified by the FOR-NEXT loop and prints ASCII values of the memory locations. It has been set here to dump the reserved word table of 8K BASIC. The entries in this table define which ASCII strings perform which functions and statements.

```
LIST

 10 FOR X=638 TO 920
 20 Z=PEEK(X)
 30 PRINTCHR$(Z);
 40 NEXT
OK
RUN
ENDFORNEXTDATAINPUTDIMREADLETGOTORUNIFRESTOREGOSUBRETURNREMST
OPONNULLWAITLOADSAVEDEFPOKEPRINTCONTLISTCLEARNEWTAB(TOFNSPC(T
HENNOTSTEP+-*/↑ANDOR>=<SGNINTABSUSRFREPOSSQRRNDLOGEXPCOSSINTA
NATNPEEKLENSTR$VALASCCHR$LEFT$RIGHT$MID$NFSNRGODFCOVOMUSBSDD/
0IDTMOSLSSTCNUF ERROR IN
OK
```

In page 19 we advise changing the reserve word entry "LIST" so that other users cannot access your program and possibly damage it. This is another of many practical applications of the PEEK and POKE functions.

We have briefly touched on some specific examples of how the use of the PEEK and POKE features of 8K BASIC can greatly expand your programming capabilities. We must warn you again, however, that extreme caution must be exercised when POKE-ing into memory, since POKE-ing a location in BASIC or operating system can cause a system crash, requiring a complete reload of the programs into the computer. The information here will allow you to write in BASIC interesting programs such as video games using switches on a parallel input port for control. Ohio Scientific would like to see examples of user programs utilizing the PEEK and POKE capabilities of BASIC and will consider publishing these in the journal.



The Model 520 16K RAM Board

Mass storage devices

Memory technologies for small computers

Every computer system requires a mass storage device to hold programs and data. Mass storage devices are necessary because the computer's main memory, RAM, has two disadvantages. It "forgets" once the power is turned off, and it has a small capacity, both because of the high cost of RAM memory, and the small address range of a computer (16 to 20 bits).

In most applications the performance of the mass storage device is actually much more critical than that of the computer itself. That is, the system performance is generally limited by the mass storage peripheral, and not by the CPU or RAM memory, etc. The mass storage peripheral is actually the most important part of most computer systems, but seldom receives that level of consideration when a computer is purchased.

Small computers generally have only one mass storage device. Devices and storage media available for use on small systems are shown in Table I, below.

TABLE I

Media	Type	OSI Product Number
Paper tape	sequential	
Audio cassette	sequential	CA-6
Digital cassette	sequential	
Mini-Floppy disk	random access	
Floppy disk	random access	C-D1, C-D2
Hard disk	random access	C-D74

The first three are sequential storage devices, that is, all data on the tape must be read until the desired data or program of interest is reached. Disk drives are at least partially random access, i.e., data on the diskette is accessible without the necessity of reading all data prior to its position on the diskette. The following is a brief discussion of each storage device.

I. Sequential devices

Paper Tape

Paper tape was one of the earliest forms of mass storage devices and is by far the most primitive. Paper tape systems require both a reader and a punch. The medium (paper tape) is not reusable and is very difficult and time-consuming to handle. Paper tape punches are very trouble-prone mechanical devices which are quite noisy and subject to high wear. They are also rather expensive. Some people have the mistaken impression that paper tape is more reliable and more permanent than other recording means because they can see the holes. In practice, paper tape readers and punches are far less reliable than any other mass storage device, and paper tapes wear out much faster than any other storage medium. Paper tapes are also easily destroyed by humidity, dust, oil, and mishandling. Actually, paper tape is being mentioned here only as a warning to uninformed hobbyists of its inferiority and low reliability.

Audio Cassette

By far the least expensive mass storage device for a computer is an audio cassette system. The most popular audio cassette systems use the "Kansas City Standard." This format uses two tones, 2400Hz and 1200Hz, to record data at about 30 bits per second on tape. Audio cassette systems generally can use very inexpensive cassette recorders and recording tape, and are generally very reliable. The severe limitation of audio cassette mass storage is that it is slow and requires manual operation of the cassette transport, i.e., rewind, fast forward, etc. Because of this, audio cassette systems should only be considered as an economical "starter" system for beginners. Cassettes can be used slowly but effectively for program storage and recall, but are not practical for data storage, although they are still much better than paper tape systems.

Digital Cassette

Digital cassettes store information on cassette tape as magnetic transitions instead of audio tones. This facilitates a much higher data density on the tape, which also allows much faster data transfer (4800 bits per second and up). However, this feature requires the use of expensive special recorders and tape. Most digital cassette systems also have remote transport control, so the computer can rewind the tape, run it on fast forward, etc. Digital cassettes can load and save programs in only a few seconds, but are as impractical as audio cassettes for data storage unless the transport has computer control. Digital cassette systems with remote transport controls are as expensive as floppy disk systems which have much higher performance. Manual control digital cassettes offer little advantage over much less costly audio cassettes, and computer control digital cassettes offer no cost advantages to the disk systems, which have higher performance.

II. Random access devices (disk systems)

Computer systems with disk are far more usable in virtually all applications of computers than systems using other mass storage devices. The important feature of disk systems is that the position of programs and data on the disk can be identified and used to access data very quickly.

Disk systems are available in two basic forms, floppy disks and hard, or "conventional," disks.

Floppy disks are so called because the floppy diskette is somewhat flexible. It is a disk of magnetic recording film eight or five inches in diameter, sealed in a permanent protective non-rotating jackette. The diskettes are placed in floppy disk drives which rotate the diskette and position and load the record/playback head. The drive is generally operated by a floppy disk controller board in the computer.

Typical floppy diskettes are shown in Diagram 1, right.

Data is stored on floppy disks as magnetic transitions. The data is arranged in concentric circles called tracks. The start of each track is specified by the index hole which is optically read as the disk rotates. Tracks can be further subdivided into sectors, either by additional holes (sector holes or hard sectoring) or by software and data on the disk itself (soft sectoring). Ohio Scientific uses soft sectoring because it provides the flexibility to vary sector lengths on tracks to match the user's file size requirements, and therefore, offers much higher performance than hard-sectored or "fixed" sector systems. Data is accessed by moving the head over the proper track, loading the head, checking for index, and then reading or writing on that track. If the track is further divided into sectors, the read/write operation is delayed until the proper sector rotates under the head. A functional diagram of a typical floppy is shown in Diagram 2, at the right.

Disk performance is specified by the total storage capacity of a disk (excluding additional capacity produced by flipping the disk or changing disk), the data transfer rate, the rotational time in revolutions per minute (RPMs), the time the head takes to move from one track to another (step time), the head-setting time, and the number of tracks on a disk.

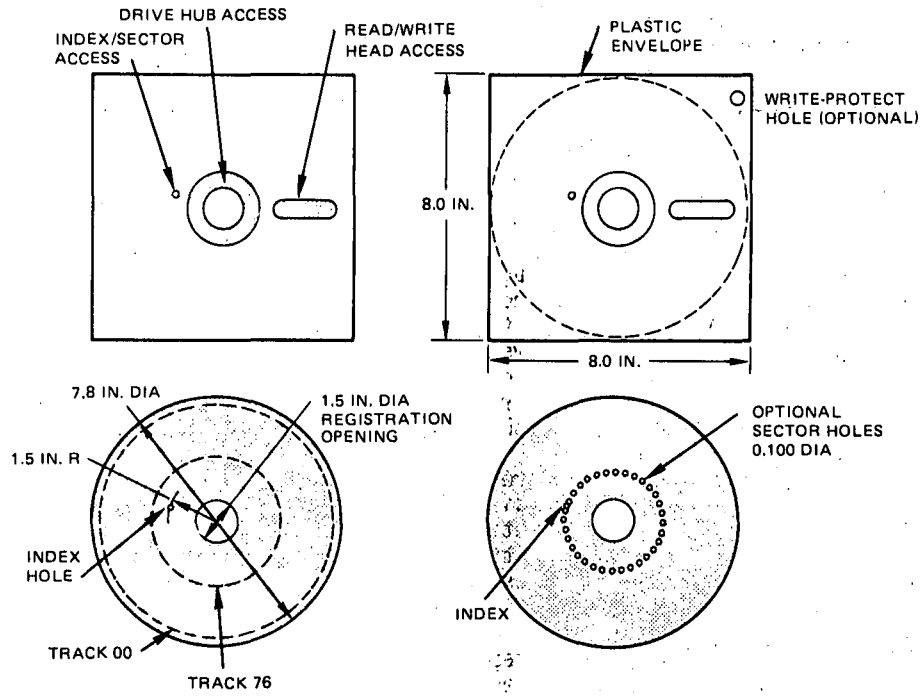


Diagram 1.

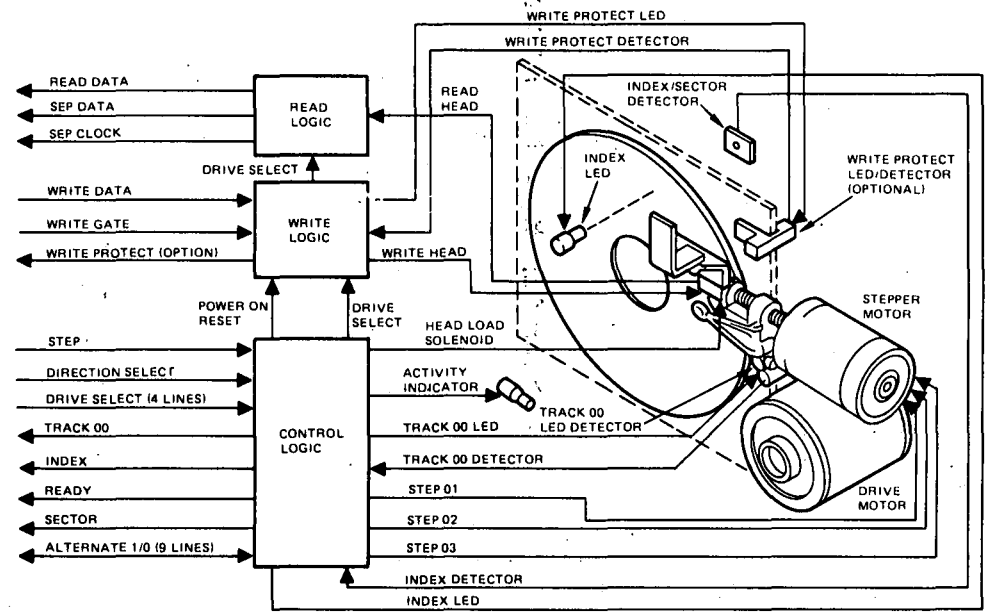


Diagram 2.

The important parameters specify how much data the disk holds and how long it typically takes to access any data. This parameter is usually specified at one-half the slowest access time, which is the amount of time required for the head to step all the tracks, load, and wait almost one complete revolution.

There are two types of floppy disks available for small computers--mini-floppies and full-size floppies. Mini-floppies use a 5.25" diskette and store almost 80,000 bytes per diskette. Full-size floppies use an 8" diskette and store about 250,000 bytes. Full-size floppies access data about twice as fast and transfer data twice as fast as minis. Mini-floppy disk drives cost only about forty dollars less than full-size drives, which is fairly insignificant compared to the retail cost of full-size floppy disk systems ($600 for the kits, or $1,000 fully assembled). Why would any engineer sacrifice a factor of 2 in speed and 3 in storage capability for a $40 savings on a product which sells for $600 or more?

The answer is that mini-floppies transfer data at only 125K bits per second (which is all the faster a 6800 or 8080 system can handle or process). On the other hand, only the 6502 system is capable of handling the 250K bits per second rate at which the full-size floppies transfer data. Consequently, 8080 and 6800 users can use only mini-floppies while 6502 users can enjoy the benefits of a full-size floppy at about the same retail cost.

Floppy disk systems provide extremely rapid program and data storage and retrieval for small computers. In most situations they operate about as fast as the user can type commands. Floppies are generally available as single-drive or dual drive units. A single-drive floppy disk is an excellent mass storage

device for a personal computerist, industrial development system or educational system. Small business users can use single floppies, but dual floppies are more convenient when large data files are required. They provide 500,000 bytes of disk storage, fast diskette copying and fast file sort, pack and merge capabilities.

Big disks

Until now, the floppy disk was the ultimate storage device for a small computer. Recent technological developments have made big system disk technology affordable and reliable enough for the small system which is not under maintenance contract. Hard disks have been in use much longer than floppies in big computer installations, but have been out of the realm of most small computer users because of their high cost ($20,000 and up) and low reliability. The new IBM "Winchester" technology disk drives solve these problems.

Hard disks are quite different from floppies. The actual disk is one or more precisely machined aluminum platters coated with magnetic material. There are one or more heads per platter side, and there can be as many as twelve heads in the disk drive. The heads are positioned with a linear motor or voice coil (much like a speaker) or rotary positioner which steps the head from track to track much faster than the stepping motor drives of floppies. Also because there are usually several heads, several tracks are in position simultaneously. Each head position is called a cylinder. There are usually as many tracks per cylinder as there are heads. (On floppies cylinders are the same as tracks, because there is only one head.)

The disks are generally 12" to 18" in diameter and are rotated about ten times faster than floppies (3000-3600 RPM). Because of the high surface velocity of the disk under the heads, the heads are designed to ride on the thin layer of air generated by the high velocity of the disk surface.

The performance of the big disks depends on several features: their size results in a large storage capacity; their ultra-fast access is due to their multiple heads, fast head actuators (no head loading), high rotational speed, and virtually no disk wear because the disk or heads never touch anything (while in floppies, the head is in contact with the diskette, which is constantly in contact with its stationary jackette). In theory, large disks would run forever if they weren't turned off.

The expensive problem with big disks has been head crashes and disk damage (sometimes destroying invaluable data). A head crash is the act of the head touching the disk surface. This can occur if dust or other foreign matter gets on the disk, or if the heads are not properly retracted during a power outage. High density disk drives (50 megabytes and above) also have had head alignment problems with disk cartridges, particularly when a cartridge at room temperature is installed in a warm drive.

These problems have been solved in the so-called "Winchester" technology first developed by IBM (which also developed the floppy disk). This technology is used in the 3340, the IBM system 32 and IBM's newest drive for the 370, the 3350.

The technology uses disks which are housed in a sealed clean-air environment with the heads, eliminating the major cause of crashes--contamination. The heads are always precisely aligned to the disk allowing ultra-high density formats. The disks have special landing and launching areas for the heads so that when the power is off, the heads deliberately come to rest on the disk surface. As the disk comes up to speed, the heads lift off the disk by their natural air cushion and are moved into position over the magnetic area of the disk. This feature eliminates power cycling head crashes and the costly and less precise head retractors of other disks.

New Big Disk from OSI

Ohio Scientific proudly announces the first Winchester technology disk for small computers, the Model C-D74.

Specifications

74 million bytes storage (unformated)
18,560 bytes per track
12 tracks per cylinder
339 cylinders
10 millsec. single track seek
35 millsec. average access
75 millsec. maximum access
7.3 megabits/sec data transfer rate
7" X 17 3/4" X 23 1/2" Rack Mount
110VAC 5amps running
30amps starting
Drive, cable, interface for OSI Challenger and OS-74 operating system software
$6,000 F.O.B., Hiram, OH

The C-D74 uses a new non-removable sealed chamber drive with a unique rotary arm positioner to provide the highest performance disk drive available. Besides providing an unbelievable 35 millsec. average access time to any of 74 million bytes, this is the first drive to offer twelve tracks on a cylinder without reseeking. That means that any of 220,000 bytes can be accessed typically in 5 milliseconds!

The C-D74 interface and controller are designed with the same philosophy as our floppy disk controller. That is, as much of the interface as possible consists of software and not hardware. The main controller difference between the big disk and the floppy is that the C-D74 has a 7 million-bit-per-second transfer rate, which is much too high for any microcomputer to handle, whereas the floppy uses programmed data transfers. One possible approach would be to have the big disk load and read memory by DMA (Direct Memory Access), but this would require that the processor be stopped during transfers, which is undesirable. Therefore, OSI has developed a new dual port memory board for use with the C-D74. This new 16K static board, the Model 525 has two standard 48-line buses, the main system bus and a memory channel. The on-board memory can be operated on from either port, so that the disk can transfer data without stopping the processor. This high-speed data channel or memory channel is a well-established feature on large computers, such as the IBM 370s, as well as large minis such as the PDP-11/45 and 11/70. The processor specifies cylinder, track, and sector, and specifies the limits of the memory transfer. It then activates the hardware that performs the transfer between disk and memory. After transfer, the memory can be accessed by the processor, normally to retrieve data. C-D74 comes complete with OS-74, a named file operating system which maintains both a table and a linked list for reliability, and includes our nine-digit precision BASIC and Assembler. OS-74 also supports one or two standard floppy drives primarily for archival storage and transferring programs between machines.

The recommended minimum hardware for the C-D74

is an OSI Challenger with 32K RAM and at least 8K on a Dual Port 525 board, and a single or dual-drive floppy disk.

The C-D74 must be mounted in a 22"-deep standard equipment rack. OSI Challengers and floppy disks are 17" wide and conveniently fit on rack shelves. OSI will be offering a complete rack mount system in the future.

The C-D74 costs $6,000, which includes a one-day user orientation at the plant. The price is F.O.B. Hiram and the delivery is 120 to 150 days A.R.O.

The C-D74 has important applications in both business computing and research in computing itself. It makes small computers practical for much larger jobs than formerly thought feasible, particularly since most business computing is disk-bound and not compute-bound. The C-D74 can store all the records of a medium-size company for instant access. The Winchester technology of the C-D74 means that the drive can be run 24 hours a day without worry of disk wear. And the sealed chamber and non-removable medium protects data against operator errors.

The huge storage capabilities and fast access open up new areas of computer research, even to those on a very limited budget. Image recognition, voice recognition and playback, and English language processing are just a few areas of exploration possible with the C-D74.

To summarize this entire article on mass storage devices, consider the following comparison tables.

Table II compares the data or program transfer rate for the six mass storage devices discussed above, and the time required to find a specific entry in a 100,000-byte-long file.

Table III compares the performance of the three disk drives discussed here. The performance factor was obtained by dividing the storage capacity by the average access time and then normalizing the result such that the mini-floppy has a performance factor of 1. The table shows that the C-D74 is rated 10,000 times more powerful than a mini-floppy at about seven times the cost!

Table IV is a cost comparison for fully assembled units, including the cost of the interface in the computer.

Table II

media	bit rate per second	average random access time for 100K byte file
Paper Tape	100 to 3000	manual to 250 sec.
Audio Cassette	300	manual to 2500 sec.
Digital Cassette	4800	manual to 160 sec.
Mini-Floppy Disk	125K	.55 sec.
Floppy Disk	250K	.286 sec.
Hard Disk	7.33M	.038 sec.

Table III

random access storage devices	formated capacity per disk	average seek time for entire disk	performance factor
Mini-Floppy Disk	70-90 Kbytes	.55 sec.	1
Floppy Disk (OSI C-D1, C-D2)	250 Kbytes	.286 sec.	5
Hard Disk (OSI C-D74)	70 Megabytes	.038 sec.	10,000

Table IV

media	cost
Paper Tape	$ 485.00 up
Audio Cassette (OSI CA-7 plus cassette recorder)	128.00 up
Digital Cassette	410.00 up
Mini-Floppy Disk (partially assembled)	877.00 up
Floppy Disk (single-drive OSI C-D1	990.00
(dual-drive OSI C-D2)	1,590.00
Hard Disk (OSI C-D74)	6,000.00

Bugs&Fixes

Many OS-65D users are overlooking the fact that the Control-C Flag bit of the input distributor must be set high on serial systems in BASIC. The proper flag code for serial input with Control-C enabled is 81 (hex) 129 (dec) and not Ø1. Remember that the high order bit specifies Control-C test from either the ACIA port (if high) or the 440 keyboard (if low).

Users of the Assembler under OS-65D may have noticed that after printing sixty lines, the Assembler prints six blank lines. This is the paging feature designed for use with line printers. Users may wish to change these to suit their own uses. Location Ø34C contains the number of lines on a page [preset to 42(hex) which is 66(dec)]. Location 12EØ contains the number of lines to be printed on a page [preset to 3C(hex) which is 60)]. For example, no lines will be skipped if location 12EØ is set to 42.

Hamurabi for Tiny BASIC and 8K BASIC

The following version of the popular computer game Hamurabi was written on a paper tape system in Tiny BASIC. This is an unusually long program in Tiny BASIC. It demonstrates one feature of Tiny BASIC, namely, memory efficiency. This program will run in a 12K computer only if it is run with Tiny BASIC since the 8K BASIC takes up too much space. If you are using an audio cassette based Tiny BASIC system, be sure to omit the colon (:) from the statements such as line 217. The colon is the escape character for audio cassette in Tiny BASIC.

8K BASIC users must change the RND statements, since 8K BASIC always returns a number between 0 and 1. For example, line 310 should be changed to:

```
LET C=INT(11*RND(1))
```

```
75 PRINT
80 PRINT "TRY GOVERNING ANCIENT SUMERIA"
85 PRINT "SUCCESSFULLY FOR A 10-YEAR TERM. "
86 PRINT
91 LET O=0
92 LET T=0
100 LET Z=0
101 LET P=95
102 LET S=2800
103 LET H=3000
104 LET E=H-S
110 LET Y=3
111 LET A=H/Y
112 LET I=5
113 LET Q=1
210 LET D=0
215 PRINT
216 PRINT
217 PRINT "HAMURABI: I BEG TO REPORT TO YOU,"
218 LET Z=Z+1
220 PRINT "IN YEAR ";Z;", ";D;" PEOPLE STARVED, ";I;" CAME TO THE CITY."
225 LET P=P+I
227 IF Q>0 GOTO 230
228 LET P=(P/2)
229 PRINT "A HORRIBLE PLAGUE STRUCK! HALF THE PEOPLE DIED."
230 PRINT "POPULATION IS NOW ";P
232 PRINT "THE CITY NOW OWNS ";A;" ACRES."
235 PRINT "YOU HARVESTED ";Y;" BUSHELS/ACRE."
250 PRINT "RATS ATE ";E;" BUSHELS."
260 PRINT "YOU NOW HAVE ";S;" BUSHELS IN STORE."
261 PRINT
270 IF Z=11 GOTO 860
310 LET C=RND(11)
311 LET Y=C+17
312 PRINT "LAND IS TRADING @ ";Y;" BUSHELS/ACRE."
320 PRINT "HOW MANY ACRES WILL YOU BUY ";
321 INPUT Q
322 IF Q<0 GOTO 850
323 IF Y*Q<=S GOTO 330
324 GOSUB 710
325 GOTO 320
330 IF Q=0 GOTO 340
331 LET A=A+Q
332 LET S=S-Y*Q
333 LET C=0
334 GOTO 400
340 PRINT "HOW MANY ACRES FOR SALE ";
341 INPUT Q
342 IF Q<0 GOTO 850
343 IF Q<A GOTO 350
344 GOSUB 720
345 GOTO 340
350 LET A=A-Q
351 LET S=S+Y*Q
352 LET C=0
400 PRINT
410 PRINT "HOW MANY BUSHELS FOR YOUR FOOD ";
411 INPUT Q
412 IF Q<0 GOTO 850
420 IF Q<=S GOTO 430
421 GOSUB 710
```

```
422 GOTO 410
430 LET S=S-Q
431 LET C=1
432 PRINT
440 PRINT "HOW MANY ACRES DO YOU WANT TO SOW ";
441 INPUT D
442 IF D=0 GOTO 511
443 IF D<0 GOTO 850
445 IF D<=A GOTO 450
446 GOSUB 720
447 GOTO 440
450 IF (D/2)<S GOTO 455
452 GOSUB 710
453 GOTO 440
455 IF D<10*P GOTO 510
460 PRINT "BUT YOU HAVE ONLY ";P;" FIELDHANDS YOU MORON.  NOW THEN, "
470 GOTO 440
510 LET S=S-(D/2)
511 GOSUB 800
515 LET Y=C
516 LET H=D*Y
517 LET E=0
521 GOSUB 800
525 LET E=(S/C)
530 LET S=S-E+H
531 GOSUB 800
533 LET I=(C*(20*A+S)/P/100+1)
540 LET C=(Q/20)
542 LET Q=(RND(21))-3
550 IF P<C GOTO 210
552 LET D=P-C
553 IF (D*100)>(P*45) GOTO 560
554 LET T=((Z-1)*T+D*100/P)/Z
555 LET P=C
556 LET O=O+D
557 GOTO 215
560 PRINT
561 PRINT "YOU STARVED ";D;" PEOPLE IN ONE YEAR!"
565 PRINT "DUE TO THIS EMBARRASSING BOO-BOO YOUR PEOPLE"
566 PRINT "(IN A SINGLE PATRIOTIC GESTURE)"
567 PRINT "NOW REFER TO YOU AS THE 'LATE' HAMURABI."
568 GOTO 975
710 PRINT "HAMURABI:  THINK AGAIN YOU GRUT-GRUT."
711 PRINT "YOU HAVE ONLY ";S;" BUSHELS OF GRAIN.  NOW THEN..."
712 RETURN
713 GOTO 410
720 PRINT "HAMURABI:  THINK AGAIN.  YOU ONLY OWN ";A;" ACRES.  NOW THEN..."
730 RETURN
800 LET C=RND(6)+1
801 RETURN
840 PRINT "WEREN'T WE JUST A BIT HEAVY-HANDED THIS TIME ?"
850 PRINT
851 PRINT "HAMURABI:  I CAN'T DO IT."
855 PRINT "GET YOURSELF ANOTHER STEWARD!"
857 GOTO 975
860 PRINT "IN YOUR 10-YEAR TERM OF OFFICE ";T;" PERCENT OF THE"
862 PRINT "POPULATION STARVED/YEAR...ON AVERAGE A TOTAL OF "
865 PRINT O;" PEOPLE DIED!"
866 LET L=A/P
870 PRINT "YOU STARTED WITH 10 ACRES/PERSON AND ENDED WITH "
875 PRINT L;" ACRES/PERSON."
876 PRINT
880 IF T>33 GOTO 565
885 IF L<7 GOTO 565
890 IF T>10 GOTO 940
892 IF L<9 GOTO 940
895 IF T>3 GOTO 960
896 IF L<10 GOTO 960
900 PRINT "WELL, WELL, WELL,...  THAT DESERVES A FOUR STAR RATING:"
905 PRINT ">>> * * * * <<<"
906 GOTO 975
940 PRINT "THE PEOPLE (REMAINING) FIND YOU AN UNPLEASANT RULER, AND,"
950 PRINT "FRANKLY, HATE YOUR GUTS!"
955 GOTO 975
```

```
960 PRINT "YOUR PERFORMANCE COULD HAVE BEEN SOMEWHAT BETTER, BUT"
965 PRINT "REALLY WASN'T TOO BAD AT ALL."
970 PRINT "75% OF THE PEOPLE WOULD LIKE TO SEE YOU AS A BLOOD SPLAT!"
975 PRINT "TO ERR IS HUMAN; TO REALLY LOUSE UP..."
990 PRINT "LOSE AT 'HAMURABI'."
999 END
```

Constructing a fool-proof end user system

OS-65D is principally a developmental operating system, that is, it enables a programmer to write software for an end user who may or may not be an experienced computer programmer. The most important consideration in producing an end user system is to make it fool-proof and easy to use. The user should only be required to turn on the computer and the disk drive, press the reset button, type a D to execute the programs which have already been written. It is essential that the user be prevented from getting into immediate mode and listing or changing the program, or corrupting the disk with OS-65D. With certain modifications, OS-65D can be converted into just such a fool-proof system.

A typical end user system may consist of a set of business programs written in BASIC. In order to create an end user system, the programmer must do the following:

1. Prevent the user from typing a Control-C or Control-O to BASIC. Doing so would get the user into BASIC, where he could list the program or change it, or even get BASIC to suppress all of its output. In order to disable the Control-C and Control-O functions, you must change the contents of two locations, one inside BASIC and one inside OS-65D. These changes can be made with BASIC via the POKE command or directly from machine language. The following table illustrates these changes.

Location		Contents enabling it		Contents disabling it	
hex	dec	hex	dec	hex	dec
7E3	2Ø19	4C	76	6Ø	96
315	8981	FF	255	Ø	Ø

For example the following line of BASIC disables Control-C and Control-O:

```
25Ø  POKE 2Ø19,96: POKE 8981,Ø
```

2. Use some form of input device other than the INPUT statement. If you were to input commands or information via the INPUT statement, then it would be possible for the end user to go to immediate mode merely by typing ⟨return⟩. In order to avoid this, use a machine language program such as Listing 1, below, which inputs a character using the I/O distributor of OS-65D. This program will be used as a USR(X) subroutine call from BASIC (see OSI 6502 8K BASIC User's Manual, pp. 9-10). The machine language subroutine passes back the character inputted to the variable X.

To use this means of input, you must set up the address of the subroutine. As the 8K BASIC User's Manual states, this address is at locations 23E and 23F. Place the address of the machine language program of Listing 1 in that location by using the POKE instruction. Then call the USR(X) function. The following code places the ASCII code value of the character typed in the variable I using the machine language subroutine:

```
1Ø POKE 574,236:POKE 575,63
2Ø I=USR(X)
```

By using this code and BASIC's string handling instructions you can do all possible forms of input with the USR(X) function rather than the INPUT state-

Listing 1

```
ASSEM

  10 0000           ;INPUT TO BASIC VIA THE USR(X) FUNCTION SO BASIC
  20 0000           ; DOESN'T TRAP I/O ERRORS.  THUS, YOU MUST DO THAT!
  30 0000           ;
  40 0000           ;INPUT:  X=USR(X)
  50 0000           ;  PLACES THE ASCII VALUE (INTEGER) IN X OF THE
  60 0000           ;  NEXT CHARACTER INPUTTED THROUGH THE I/O
  70 0000           ;  DISTRIBUTOR.
  80 0000           ;
  90 0000           ;
 100 3FEC                    *=$3FEC
 110 3FEC 20F921    INPUT    JSR $21F9   INPUT A CHARACTER TO AC
 120 3FEF A8        PASS     TAY  NOW, PASS THE RESULT BACK
 130 3FF0 A508               LDA 8  SEE PAGES 9-10 OF 8K BASIC MANUAL
 140 3FF2 8DFD3F             STA CALL+1
 150 3FF5 A509               LDA 9
 160 3FF7 8DFE3F             STA CALL+2   CALL SUBROUTINE POINTED TO
 170 3FFA A900               LDA #0       BY 8 AND 9 WITH THE
 180 3FFC 200000    CALL     JSR $0000    HIGH PART IN AC AND LOW IN Y
 190 3FFF 60                 RTS
 200 4000                    .END
```

ment. For example, the following code will ask for and input an integer and place it in X.

```
LIST

 5 PRINT "A NUMBER PLEASE";
 10 POKE 574,236 : POKE 575,63
 20 A$=""
 30 A=USR(A)
 40 IF A=13 THEN GOTO 80
 50 IF A<48 OR A>57 THEN PRINT "?" : GOTO 5
 60 A$="A$+CHR$(A)
 70 GOTO 30
 80 X=VAL(A$)
OK
```

There must be more error-checking than this, of course, but this is enough to illustrate the fundamental principal of using POKE and USR functions to avoid the use of the INPUT statement.

Another technique would be to have BASIC print an error message "REDO FROM START", if you type a return. This will also prevent a user from getting back to the command mode. This change can be made by the following patch:

Location		Old Contents		New Contents	
hex	dec	hex	dec	hex	dec
B12	2834	FF	255	E1	225
B13	2835	Ø7	7	ØA	1Ø

These changes can be made by using the POKE command in BASIC.

3. Make sure that the bootstrap process brings you up in the BASIC program, and not in BASIC's immediate mode. In order to modify OS-65D so that the program is loaded and executed on reset, you must place a patch inside OS-65D. When the user types a D, this tells the software in the PROM to place the contents of track Ø into memory. This track is recorded in a very simple format, so that all the bootstrap software will fit in 256 words of a 1702 PROM. For OS-65D, track Ø contains 256Ø bytes which consists of the BASIC Disk I/O subroutines and a power-up initialization package. After this block of data is loaded, execution is started at 22ØØ (hex). The code here will load the rest of OS-65D and 8K BASIC, initialize all the I/O devices, and then cold start BASIC.

At this point, your patch is inserted. One of the tracks loaded will overlay (that is, over-write) this code, which starts to run on power-up. You will then change this track so that after it is loaded, you can load one of your own tracks instead, which will include your patch. Suppose the patch is located from 3FØØ to 3FFF, and that it is on track 43, sector 1, and it starts at 3FC1. The source code for the patch is shown in Listing 2 below. To load this file off track 43, sector 1, you must make the changes illustrated in Listing 3 below. First load the file of OS-65D on track 5 into 32ØØ. Note that this is not the location where it is intended to execute. We then must enter the Extended Monitor and make several changes. At 34C6, place a 43, indicating that you want to load track 43 instead of track Ø6(track numbers are BCD). Track Ø6 must now be loaded by our patch at 34CB to 34CF.Change the low and high address where your track is to be loaded into, namely, track 43 into 3FØØ, instead of track Ø6 into 2B3Ø. Finally, after the call to the subroutine which loads the file, place a jump to your patch.

```
ASSEM
```

Listing 2, at left

```
   10 0000           ;POWER-UP INITIALIZATION FOR END USER SYSTEM
   20 0000           ; TO LOAD SOFTWARE AND AUTO EXECUTE IT.
   30 0000           ;
   40 0000           ;DEFINITIONS OF GLOBALS:
   50 0000           ;
   60 0000           LOCL=$CB
   70 0000           LOCH=$CC
   80 0000           SECT=$CF
   90 0000           ;
  100 0000           INFLAG=$2203
  110 0000           OTFLAG=$2204
  120 0000           INPNTL=$2E67
  130 0000           INPNTH=$2E68
  140 0000           ;
  150 0000           SEEK=$26D6
  160 0000           CALL=$29AC
  170 0000           ;
  180 3FC1                 *=$3FC1
  190 3FC1           ;
  200 3FC1 A901      PATCH  LDA #1  LOAD TRACK 6 OF OS-65D
  210 3FC3 85CF             STA SECT
  220 3FC5 A906             LDA #6
  230 3FC7 20D626           JSR SEEK
  240 3FCA A930             LDA #$30
  250 3FCC 85CB             STA LOCL
  260 3FCE A92B             LDA #$2B
  270 3FD0 85CC             STA LOCH
  280 3FD2 20AC29           JSR CALL
  290 3FD5 A908             LDA #%1000  SET INPUT TO BE FROM INDIRECT
  300 3FD7 8D0322           STA INFLAG  FILE IN MEMORY AT 3F00
  310 3FDA A980             LDA #%10000000
  320 3FDC 8D0422           STA OTFLAG
  330 3FDF A93F             LDA #$3F
  340 3FE1 8D682E           STA INPNTH
  350 3FE4 A900             LDA #0
  360 3FE6 8D672E           STA INPNTL
  370 3FE9 4C3F1F           JMP $1F3F  COLD START BASIC
  380 3FEC                  .END
```

Listing 3

```
A*VA
A*C3200=05,1
A*RE
:#34C6/06 43
:#34CB/30 00
:#34CF/2B 3F
:#34D5/AD 4C
:#34D6/01 C1
:#34D7/FE 3F
:D
A*S05,1=3200/3
```

Now that you have finished the patches, write the modified program back on to track 5, sector 1.

Now refer to the patch you loaded off of track 43, sector 1 (Listing 2). On lines 200 to 280, load track Ø6 which should have been loaded when you loaded this program. Now set the file at 3FØØ (lines 290 to 360). This will prevent the user from issuing commands from the keyboard and insures that you, the programmer, are in control of the dialog at the start. Listing 4, right, shows what you are to place in memory at 3FØØ.The file gives the answer to "MEMORY SIZE?" which is 16128. This allots to BASIC all of memory up to 3EFF. The last page of your 16K machine is your domain for this software. It then types <return> in response to "TERMINAL WIDTH?" The file then types LOAD as a command to BASIC; this gets you out to OS-65D where the file loads the program on track 45 (See Listing 5 below) into BASIC. Finally, it returns to BASIC and runs the program. This was done without any operator intervention after typing D.

Listing 6, p.18 , shows what you have loaded from 3FØØ to 3FFF from track 43, sector 1.

The following is a breakdown of the contents:

3FØØ-3F14	Indirect file of commands
3F15-3FCØ	Nothing
3FC1-3FEB	Power-Up Patch to OS-65D
3FEC-3FFF	USR(X) Subroutine for Input

Listing 4

```
 10 0000          ;GENERATE AN INDIRECT
 20 0000          ;COMMAND FILE THAT TAKES
 30 0000          ;CONTROL AFTER THE USER
 40 0000          ;TYPES A "D" TO D/M?.
 50 0000          ;
 60 3F00                 *=$3F00
 70 3F00          ;
 80 3F00 31              .BYTE '16128',$D
 80 3F01 36
 80 3F02 31
 80 3F03 32
 80 3F04 38
 80 3F05 0D
 90 3F06 0D              .BYTE $D
100 3F07 4C              .BYTE 'LOAD',$D
100 3F08 4F
100 3F09 41
100 3F0A 44
100 3F0B 0D
110 3F0C 4C              .BYTE 'L45'
110 3F0D 34
110 3F0E 35
120 3F0F 52              .BYTE 'RB'
120 3F10 42
130 3F11 52              .BYTE 'RUN',$D
130 3F12 55
130 3F13 4E
130 3F14 0D
140 3F15                 .END
```

Listing 5

```
LIST

 10 REM FIRST DISABLE CONTROL-C AND CONTROL-O
 20 REM
 30 POKE 2019,96 : POKE 8981,0
 40 REM
 50 REM THEN ASK WHICH PROGRAM TO LOAD
 60 REM
 65 POKE 8707,1 : POKE 8708,1
 70 PRINT : PRINT : PRINT
 80 PRINT "THERE ARE 4 PROGRAMS AVAILABLE:"
 90 PRINT TAB(7);"1.  ACCOUNTS PAYABLE"
100 PRINT TAB(7);"2.  ACCOUNTS RECEIVABLE"
110 PRINT TAB(7);"3.  INVENTORY"
120 PRINT TAB(7);"4.  LEDGER"
130 PRINT
140 PRINT "TYPE THE NUMBER WHICH IS BEFORE THE PROGRAM YOU WANT:";
150 POKE 574,236 : POKE 575,63
170 X=USR(X) : PRINT
180 IF X<49 OR X>52 THEN GOTO 70
190 X=X-48
200 ON X GOTO 1000,2000,3000,4000
1000 REM
1010 REM ACCOUNTS PAYABLE IS ON TRACK 51
1020 REM
1030 TRAK=51
1040 GOSUB 10000
1050 END
2000 REM
2010 REM ACCOUNTS RECEIVABLE IS ON TRACK 52
2020 REM
2030 TRAK=52
2040 GOSUB 10000
2050 END
3000 REM
3010 REM INVENTORY IS ON TRACK 53
3020 REM
3030 TRAK=53
3040 GOSUB 10000
3050 END
4000 REM
4010 REM LEDGER IS ON TRACK 54
```

```
4020 REM                    Listing 5, continued
4030 TRAK=54
4040 GOSUB 10000
4050 END
10000 REM
10010 REM GENERATE THE INDIRECT COMMAND FILE TO LOAD THE PROGRAM
10020 REM
10030 POKE 8708,16
10040 POKE 11860,0 : POKE 11861,63
10050 PRINT "LOAD"
10060 PRINT "L";MID$(STR$(TRAK),2);
10070 PRINT "RB";
10080 PRINT "RUN"
10090 POKE 11879,0 : POKE 11880,63
10100 POKE 8707,8 : POKE 8708,128
10110 RETURN
OK
```

Listing 6

```
U3F00,3FFF
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
3F00  31 36 31 32 38 0D 0D 4C 4F 41 44 0D 4C 34 35 52
3F10  42 52 55 4E 0D 00 00 00 00 00 00 00 00 00 00 00
3F20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3FA0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3FB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3FC0  00 A9 01 85 CF A9 06 20 D6 26 A9 30 85 CB A9 2B
3FD0  85 CC 20 AC 29 A9 08 8D 03 22 A9 80 8D 04 22 A9
3FE0  3F 8D 68 2E A9 00 8D 67 2E 4C 3F 1F 20 F9 21 A8
3FF0  A5 08 8D FD 3F A5 09 8D FE 3F A9 00 20 00 00 60
```

Examine now the program on track 45, sector 1, by referring to Listing 5. Control-C and Control-O are disabled on line 30. The I/O Distributor is set to the ACIA teletype terminal on line 65. The user is then told which programs are available, and is asked for program 1, Since program 1 is on track 51 (see Listing 7, below) you must set the variable TRAK=51. The subroutine at line 10000 generates the same sort of an indirect command file that was used to load this program, to reload the same program on track 51. The subroutine generates the following set of commands:

```
LOAD
L51
RB
RUN
```

The subroutine sets the I/O Distributor to input from the file which it just outputed through the I/O Distributor. Notice that these commands are put also at 3FØØ and upwards in memory. The subroutine returns and an END statement is encountered. Control is now returned to BASIC, which gets a LOAD command, which exits to OS-65D. This receives the command L51, which loads the program of interest over the original program. The RB command then returns to BASIC, which gets a RUN instruction, starting the program execution.

Listing 7 illustrates what the program should do on entry and exit. The I/O Distributor must first be set back to the ACIA input, and the program of interest may then continue (this example happens to show a portion of an Accounts Payable program). When it is time for the program to exit, it must make use of an indirect file to load and start the supervisory program shown in Listing 5, and then set the I/O Distribu-

Listing 7

```
10 REM ACCOUNTS PAYABLE  (TRACK 51)
20 REM
30 POKE 8707,1 : POKE 8708,1
40 PRINT : PRINT : PRINT
50 PRINT "--ACCOUNTS PAYABLE--"
10000 REM
10010 REM GENERATE THE INDIRECT FILE TO LOAD THE DISTRIBUTOR
10015 PRINT " NOW EXITING..."
10020 REM
10030 POKE 8708,16
10040 POKE 11860,0 : POKE 11861,63
10050 PRINT "LOAD"
10060 PRINT "L45";
10070 PRINT "RB";
10080 PRINT "RUN"
10090 POKE 11879,0 : POKE 11880,63
10100 POKE 8707,8 : POKE 8708,128
10110 END
```

tor to input from that indirect file. The program then exits.

When you are developing a similar system yourself, be sure that when a program (BASIC source file in the L/P format) is loaded, it doesn't wipe out the indirect file or the USR(X) machine language subroutine which you have created. Source files in the L/P format start at 3179 and are saved in chunks of B pages 2.75K). Always allot only so much memory to BASIC as is not used for indirect files or machine language subroutines. The L/P format Loads and Puts memory regardless of what is assigned to BASIC. Therefore, when you Put a file and it is certain that your indirect file area will be over-written, you must move your indirect file to a higher memory address so that this doesn't occur. Notice that in the example given here, the BASIC L/P format files were loaded into 3179 to 3C78.

One final change you could make when developing an end user system, would be to change the command to list a program to some word other than LIST. Change it, for example, to a random group of characters, such as TBZA, with which the end user will not be acquainted. This will allow the listing of the program, but not in the usual way, thus it serves as another protective feature. To change the command, place the four new ASCII characters at the locations where the word LIST was stored in BASIC. However, you must use 7-bit ASCII and set the most significant bit high on the fourth character. The following table shows where the word LIST is located, and the characters by which it is replaced to change the command to TBZA.

Location		Old Contents		New Contents	
hex	dec	hex	dec	hex	dec
2DF	735	4C	76	54	84
2E0	736	49	73	42	66
2D1	737	53	83	5A	90
2E2	738	D4	212	C1	193

Odds&Ends

The price list and product offerings listed in our July issue are still current. For more details see the catalog in this issue, beginning on page .

* * * * * * * *

We have access to a high performance FOCAL for the 6502 complete with floating point and transcendental functions. If there is interest in FOCAL we will offer it for approximately $40. Please send a note to the journal if you are interested.

* * * * * * * *

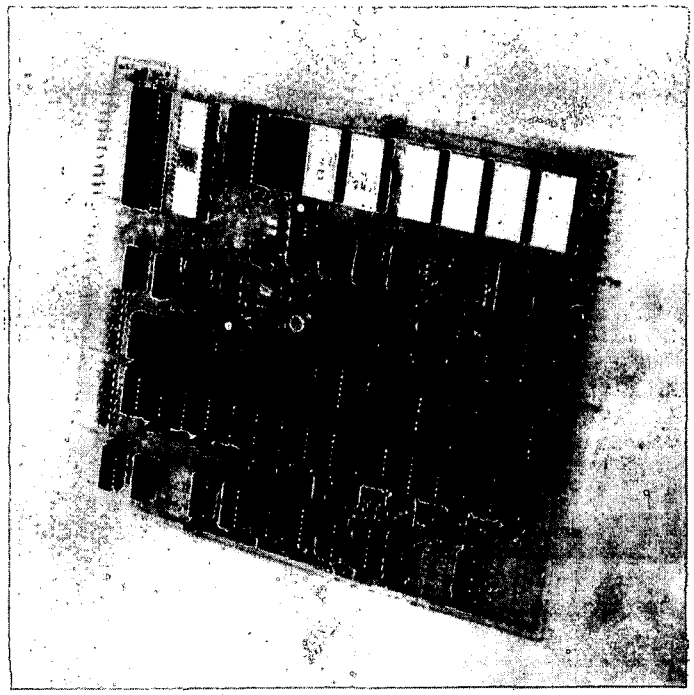
Kit Builder Trouble-Shooting Hints

The main causes of trouble are:

1. Solder bridges. 90% of all malfunctions are due to solder bridges or component failures induced by "powering up" with solder bridges.

2. Junk components. There is a tremendous amount of scrap TTL being peddled by mail order electronics companies and at least one large electronics retailer. As a rule of thumb, "relabeled" TTL will not work in computer circuits. Most of this TTL is from "lot rejects" where a large percentage of the components were bad, indicating a process problem, or, a slightly different unpopular component relabeled as a popular one. Example: 74H20s relabeled as 7420s!

3. Misplaced components and jumpers and PC board shorts. These problems are less prevalent than problems 1 and 2, but are significant. PC boards should be carefully inspected before assembly and after assembly. At least 99.9% of all problems can be corrected or avoided by careful visual inspection and the use of high quality components.



OSI's Model 500 CPU Board

1K Corner

Mini-graphics for the 440 Alpha display

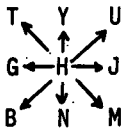
```
U0200,02B1
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0200  D8 A9 00 AA 85 FE A9 D0 85 FF A9 20 85 FD A9 01
0210  20 99 02 A9 D4 C5 FF D0 F5 C6 FF A9 65 85 FE A9
0220  3F 81 FE 20 ED FE 85 FD 81 FE 20 ED FE C9 23 F0
0230  D0 C9 5E F0 60 C9 0D F0 E6 29 7F C9 54 F0 1E C9
0240  59 F0 21 C9 55 F0 24 C9 47 F0 27 C9 4A F0 2A C9
0250  42 F0 2D C9 4E F0 30 C9 4D F0 33 D0 CD A9 DF 20
0260  99 02 10 C6 A9 E0 20 99 02 10 BF A9 E1 20 99 02
0270  10 B8 A9 FF 20 99 02 10 B1 A9 01 20 99 02 10 AA
0280  A9 1F 20 99 02 10 A3 A9 20 20 99 02 10 9C A9 21
0290  20 99 02 10 95 A9 20 10 8D 10 48 65 FE 85 FE 68
02A0  30 07 90 02 E6 FF 4C AD 02 B0 02 C6 FF A5 FD 81
02B0  FE 60 00 85 25 4C 79 11 B1 24 AA A8 68 85 24 68
```

Memory Requirements:

177 (dec) bytes; located from Ø2ØØ (hex) to Ø2B1 (hex) zero page locations FD, FE, and FF. OSI 65V Monitor located at FEØØ (hex); OSI 440 Video Board at DXXX.

Program Control Functions:

When the program is initialized, the screen is cleared and a ? is set at the lower left hand corner of the screen. Typing any ASCII character will replace the ? with that character. The ASCII symbol is the mini-graphic "pen" and may be moved in any direction on the screen by pressing any of the following keys: T,Y,U,G,J,B,N,M. Considering H as the Home Key, the control vectors are best described by this diagram:



In addition to the vector control keys, the program will also recognize the following command keys:

Carriage return: asks for a new pen (?) at the pen's current location. Any ASCII may be used.

(number sign): re-initializes the screen when the program is in the vector control mode.

↑ (up arrow): replaces the pen at its current location with an ASCII space, i.e., blanks the pen.

Program Modification:

The location of the ? on the screen at program initialization may be changed by changing memory location Ø21C (hex) from 65 (hex) to another hex value. If, for example, your screen is under-scanned, 83 (hex) or even A3 (hex) may be more desirable. However, it is not possible to move the initialization location out of the lower part of the screen with this modification.

If this program is to be called as a subroutine from another program, make the following changes:

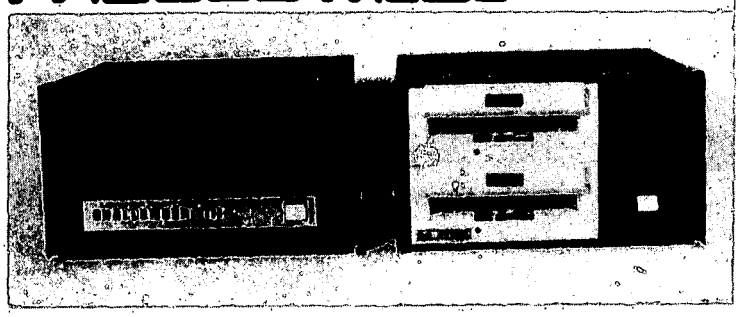
location (hex)	data	
	current	new
Ø295	A9	EA
Ø296	2Ø	EA
Ø297	1Ø	EA
Ø298	8D	6Ø

(up arrow will now cause a return from subroutine. The pen lift function may be replaced with CARRIAGE RTN followed by SPACE. The four locations from Ø295 (hex) to Ø298 (hex) may, of course, be used for any other function that you wish to control with ↑, subroutine calls, etc.

Program Relocation:

As stated in the memory requirements section, the ASCII Mini-Graphics resides at Ø2ØØ (hex to Ø2B1 (hex). However, by changing nine subroutine calls and one Jump Absolute, the program may reside at any location. All affected Subroutine calls are "2Ø 99 Ø2". Their locations are (hex) Ø21Ø, Ø25F, Ø266, Ø26D, Ø274, Ø27B, Ø282, Ø289, and Ø29Ø. The Jump Absolute, "4C AD Ø2" is located at Ø2A6 (hex). For example, if this program were relocated starting at Ø7ØØ (hex), the Subroutine calls would become "2Ø 99 Ø7" and the Jump Absolute would become "4C AD Ø7". Note that regardless of program location, zero page FD, FE, FF will still be used.

PRODUCTnews



The Ohio Scientific Challenger III contains a revolutionary new triple processor CPU board that can run virtually all published software available today for microprocessors at a very small cost increase over comparable single processor computers. Equipped with three microprocessors, Challenger III runs 6800, 6502, 8080 and Z-80 programs.

Challenger III comes standard with the OS-65D Disk Operating System and is ideal for educational applications. Students can study the three microprocessors for programming and engineering analysis.

Small business application is an ideal use for the Challenger III. Businessmen can utilize software packages written for and of the three microprocessors while conducting everyday business functions on the computer.

Industrial development is another area where Challenger III can be utilized for the investigation and comparison of the three processors. A 74 megabyte disk option makes mass data storage a reality for the experienced user.

Still another application of the Challenger III is for personal computing. The Personal Computing enthusiast can experiment with the three processors to no limit with software programs of all three types. Personal finances, strategic games, home and business applications are just a few of the Challenger III applications.

BASIC in ROM Computers by Ohio Scientific

If you're just getting into personal computing and are buying your first machine, you're probably confused by the myriad of companies and products available.

However, there is one simple guideline you should follow when choosing your first computer. Be sure that it is capable of giving you full floating-point BASIC the instant you turn it on. Machines with full 8K BASIC in ROM cost as little as $298.00. Why should you settle for anything less?

Challenger IIP

The Challenger IIP from Ohio Scientific is the ideal personal computer complete with BASIC in ROM and plenty of RAM (4K) for programs in BASIC.

Complete with an audio cassette interface, the Challenger IIP uses a full computer keyboard, not a calculator keyboard.

In addition, the Challenger IIP comes complete with a full 64 character-wide video display, not a 40 character display. The user simply connects a video monitor or home TV set via an RF converter (not supplied) and optionally, a cassette recorder for program storage.

The Challenger IIP comes complete with a 4 slot backplane and case for only **$598.00.** Fully Assembled.

Model 500

The Model 500 is a fully populated 8 x 10 P.C. Board with 8K BASIC in ROM, 4K RAM, serial port and Ohio Scientific Bus compatibility for instant expansion. All you need is a small power supply (+5 at 2 amps and −9 at 500 MA) and an ASCII terminal to be up and running in BASIC. And all for only **$298.00.**

Super Kit

The Super Kit is a 3 board set with a 500 board (like the Model 500) without the serial interface.

The ROMs are configured for use with the included, fully assembled 440 video board to provide a full BASIC computer and terminal.

The Super Kit also includes a fully assembled 8 slot backplane board which gives you 6 open slots for expansion.

To be up and running in BASIC simply plug the boards together, supply power (+5 at 3 amps and −9 at 600 MA), add an ASCII parallel keyboard plus a video monitor or TV set via an RF converter (not supplied).

Total price for the "kit" **$398.00.**

OHIO SCIENTIFIC

11679 Hayden • Hiram, Ohio 44234

Disk Based Co by Ohio

Any serious application of a computer demands a Floppy disk or hard disk because a disk allows the computer to access programs and data almost instantly instead of the seconds or minutes required with cassette systems. In real-world application of computers, such as small business accounting, a cassette based computer simply takes too long to do the job.

Ohio Scientific offers a full line of disk based computers utilizing full size floppy disks with 250,000 bytes of formatted user work space per disk. That's 3 to 4 times the work space of mini-floppies.

Challenger II

Challenger II is available with a single or dual floppy disk and a minimum of 16K of RAM instead of ROM BASIC. The disk BASIC is automatically loaded into the computer so there is no need for ROMs.

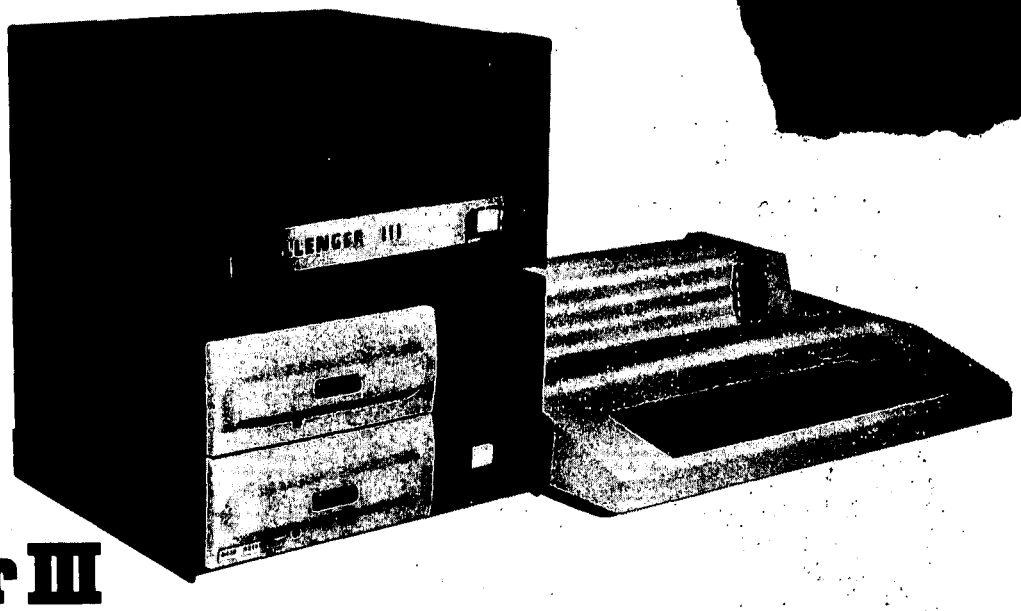
Ohio Scientific's powerful disk operating systems allow the computer to function like a big system with features like random access, sequential, and index sequential files in BASIC and I/O distributors which support multiple terminals and industry-standard line printers.

Challenger II's with disks can have the following optional features:

- 16 to 192K of RAM memory
- Single or dual drive floppys
- Serial and/or video I/O ports
- Up to 4 independent users simultaneously
- Two standard line printer options
- Optional 74 Megabyte Hard disk
- Much more

Challenger II disk systems are very economical. For example a 16K Challenger II computer with serial interface, single drive floppy disk, BASIC and DOS costs only **$1964.00** fully assembled.

mputer System
Scientific



Challenger III

Ohio Scientific proudly announces the ultimate in small computer systems, the Challenger III. This computer has a 3 processor cpu board equipped with a 6502A, 6800, and Z-80.

This system allows you to run virtually all software published in the small computer magazines!

The Challenger III is fully software and hardware compatible with Ohio Scientific products and can run virtually all software for the 6800, 8080 and Z-80 including Mikbug® dependent 6800 programs!

Incredible as this is, Challenger III costs only about 10% more than conventional single processor microcomputers. For example a 32K Challenger III with a serial interface and a dual drive floppy disk (500,000 bytes of storage) costs only **$3481.00.** Fully Assembled, complete with software. Terminal not included.

☐ **Send me the Fall '77 Catalog. I enclose $1.**
☐ I would like to order directly from this advertisement. (Please allow up to 60 days for delivery)

NAME_____

ADDRESS_____

CITY_____ STATE_____ ZIP_____

To order: Payment by:
BAC (Visa)_____ MC_____ Money Order_____
Credit Card Account #_____
Interbank # (Master Charge)_____

_____ Model 500 Boards @ **$298.00** _____

_____ Challenger IIP @ **$598.00** _____

_____ Super Kit @ **$398.00** _____

_____ 16K Challenger II complete with serial interface, single drive floppy disk, BASIC and DOS @ **$1964.00** _____

_____ 32K Challenger III with serial interface, a dual drive floppy disk (500,000 bytes of storage) @ **$3481.00** _____

Ohio Residents add 4% sales tax _____
TOTAL CHARGED OR ENCLOSED _____

Order directly from: Ohio Scientific, 11679 Hayden St., Hiram, Ohio 44234 or your local OSI dealer
All orders shipped insured UPS unless otherwise requested.

OHIO SCIENTIFIC

11679 Hayden • Hiram, Ohio 44234

Ohio Scientific 11679 Hayden Hiram, OH 44234
SMALL SYSTEMS JOURNAL