PEEK (65)

The Unofficial OSI Users Journal

1819 Bay Ridge Ave., Suite 220
Annapolis, MD 21403

Editor: Al Peabody
Tech Editor: Dick McGuire
Contributing Editor: Corky Kirk
OSIO Editor: Wallace Kendall

#3

Column One

Al Peabody

PEEK(65) continues to grow. I am amazed at the response we have had to the first two issues. Amazed and encouraged and delighted. Someone complained in a letter that there had been very little for C1P´ers except promises in the first two issues. This issue is a veritable C1P feast. Please do not write letters complaining about a lack of features for C3 users. Write an article or a letter instead.

Letters. Last issue, a reader asked what could be done about the "string handling bug." I could have thought of something terribly wise to say about how it was a Garbage Collection problem and let it go at that. Instead, we threw the question open to our readers, and have a very scholarly article with a solution, plus a letter with another solution, and I even had a phone call from a C4P user who says his machine, purchased just this February, doesn´t have the problem. I learned my lesson. From now on, PEEK(65) will not attempt to answer "problem" letters. Turns out, you guys out there know far better than we how to solve each other´s problems. All that is needed is a forum, and that we can provide.

PEEK(65) is moving. Beginning April First, perhaps symbolically, our address will be 1819 Bay Ridge Avenue, Suite 220, Annapolis, MD 21401. We have simply outgrown our present quarters. But please do note the date of the move. Mail sent to us at the new address before April First may be lost or returned. Mail sent to the old address wil, of course, be forwarded, if not by the Postal Disservice then by the new tenants.

And do keep those cards and letters coming. I cannot emphasize enough how necessary that is. PEEK(65) will be useless if I have to write it all!

One more thing. If what you want to send in is a program or memory dump, do us all a favor: send it in typed or printed with a nice clean black ribbon. Third-generation photocopies of printouts which were done with last year's ribbon print just awful, and if old fatfingers the editor tries to retype your 300 lines of machine code, why, there's just bound to be at least three fatal typos before I'm through.

Tech Notes

Dick McGuire

BASIC - DOS Interface Subroutine The OS65-U manual gives, as an example of machine code - BASIC interfacing, a description of how to use the DISK I/O under programmer control instead of the usual PRINT %CH instruction. One might well ask, "Why?". There are several reasons. The main reason I have used it in the past is to reduce or eliminate contention for the buffer in OS65-U.

BASIC can handle up to seven channels at once; however there is only one buffer. A program which reads from one file and writes sequentially to another will suffer from this buffer contention unless the programmer uses the DOS subroutine. A very good way to use it is to establish a buffer of appropriate length at the top of the workspace BEFORE any string variables have been declared. This is done by POKEing 127, 129 and 131 with a page number which represents the new top of memory. In a 48K system the normal top of memory is at page 188. One page is equivalent to 256 bytes; therefore if one wanted a buffer 1024 bytes long the new values for the three POKEs would be 184.

The programmer could then load this buffer by POKEing stuff up there or printing to memory (PRINT#4). When he wanted to write the buffer out to the disk he could invoke the subroutine below.

Before doing that however; a few things need to be done. The programmer must find the location on the disk where he wants to write the buffer. This location (called DA in the subroutine above) is the INDEX into the file plus the Disk Address of the start of the file plus 16 bytes. The following subroutine will find the DA of the begining of file KN$ as DA. Both of these subroutines use a buffer which starts at 45056 specified as RA (RAM Address) and will read or write NB (Number of Bytes) bytes. The variable RW determines whether a Read or a Write is to take place. 0=read; 1=write.

There are several things to watch out for. The first is to be sure that you are reading from or writing to the correct place on the correct disk. This subroutine doesn't bother with such trivial things as OPENing or CLOSEing files. It read from or writes to the location it is told to, no matter what. Another problem is that the subroutine hiccups a little when it crosses track boundaries and may lose a little data. Does anyone know why or how to fix that?

```
34200 REM SUBROUTINE TO FIND BEGINING OF FILE KN$
34300 DEV DV$(1):OPEN"DIREC*","PASS",3
34400 FIND KN$,3
34500 A=INDEX(3)+25
34600 DA=25088:NB=512:AR=45312:RW=0:GOSUB 35100
34700 CLOSE 3
34800 Q=45312+A:DA=PEEK(Q)*256+PEEK(Q+1)*65536+PEEK(Q+2)*16777216
34900 NB=PEEK(Q+3)*256+PEEK(Q+4)*256+PEEK(Q+5)*256
35000 RETURN
35200 POKE 8778,192: POKE 8779,36
35300 POKE 9432,243:POKE 9433,40
35400 POKE 9435,232:POKE 9436,40
35500 CB=9889
35600 DH=INT(DA/16777216):RM=DA-DH*16777216
35700 DM=INT(RM/65536)
35800 RM=RM-DM*65536
35900 DL=INT(RM/256):RM=RM-DL*256
36000 POKE CB+1,RM:POKE CB+2,DL
36100 POKE CB+3,DM: POKE CB+4,DH
36200 Q=256
36300 POKE CB+5,NB-INT(NB/Q)*Q:POKE CB+6,INT(NB/Q)
36400 POKE CB+7,AR-INT(AR/Q)*Q
36500 POKE CB+8,INT(AR/Q)
36600 ER=USR(RW)
36700 IF ER=0 THEN RETURN
36800 IF ER<>0THENPRINT "DISK ERROR # ";ER
```

Classy ADs

FOR SALE OR TRADE - Honeywell Keytape System #715, consists of #901 Keytape, #112 High speed Printer and Communications Adaptor with RS232/DP25.
Interface - Best offer over $800 or will trade for ???
I also have an Ohio CD2P Disk Drive with interface, retail $1599, but will sacrifice for $1200. A "Byte 8" power supply and ten slot S-100 Mother Board in small cabinet for $250 and a Z-80 CPU kit/S-100 board for $150 (both for $350). All of the above including manuals. Write Bill Reid, 7259-B Nestle Ave., Reseda, CA 91335. No phone calls please.

SOFTWARE FOR OSI C1P or Superboard on cassette: Tank attack 4K, Tic-Tac-Toe 4K, Bingo 8K $5.00 ea. Disassembler 4K $7.00. Inst. and source listing incl. Write: J.L. Smith, 11201 Crestfield Dr., Huntsville, AL 35803.

COLOR GRAPHICS for the OSI challenger 1P or superboard II
Details on modifications that will enable you to operate in several color graphic modes (8 colors plus black). Easy installation. Includes color-video game for demonstration and checkout that will run on systems with 4K RAM. Low-cost hardware not included. Send $8.65 to S. Hoffman, 873 Dorset Dr., Knoxville, TN 37919.

MULTIPLE KEY FUNCTION -This small utility (206 bytes; 11 lines) permits certain keys to be used as words in programming. You no longer have to type out FOR, THEN, POKE, GOTO ect.; Simply type one key. It is like using a "?" for the word PRINT. It can easily be modified for any number of different words. This program listing consists of eleven lines written in OSI BASIC. Operating instructions for all systems are included with this utility program. Price: $1.50. Send to: James Beneke, 12 Kent Drive, Orchard Park, NY 14127.

THE STRING HANDLING BUG REPAIRED!

By Stan Murphy
Seattle, WA

THE PROBLEM:

Strings that are not defined between quotes in a BASIC statement are stored in the "string space" at the top of memory. For example, A$="A" is stored in the BASIC statement. However, INPUT B$ puts B$ in the string space with pointers to B$ stored in the BASIC variable table.

Suppose the following progran is run.

```
20 C$="":A$="A"
30 INPUT K
40 FORI=1TOK:  C$=C$+A$:NEXT
```

C$ is now K bytes long and takes K bytes to store. However, we have "used" K(K+1)/2 bytes of memory in generating and storing C$. Thus, in the above program, if K=255 (the maximum string length), we need 255 bytes to store the final C$ but we have tried to use 32,640 bytes! Available memory vanishes very quickly if a program contains repetetive operations of this kind.

A memory rearrangement is needed if memory is to be properly utilized. This is accomplished through the Garbage Collection (GC) routine. This routine is called by BASIC when the string space is full. The GC relocates the valid strings back to the top of memory and defines new pointers in BASIC variable space. Using the above numeric example, after GC is executed, 32,385 bytes of memory have been recovered for further use.

OSI´s BASIC in ROM GC routine works fine for programs containing numeric variables, string variables, and numeric arrays. The above program can be run on the smallest memory machines without a problem. However, if the program also contains a string array, then the internal GC will not work properly, If, for example, we add the following line to the above program,

```
10 DIM I$(6)
```

and K=255, the internal GC will cause the screen to "pulse" several times at a 1.6 second period as the GC routine walks through memory. This pulsing is characteristic of GC failure along with a "dead" keyboard. Extraneous charactacters may show up on the screen and the BASIC program may be altered. The execution time without line 10 is under 2 seconds and exceeds 12 seconds with line 10. If the program is entered exactly as written, the pulsing may continue until the computer is reset. Even if the program finally executes, C$ is not placed properly at the top of memory.

A more general program to demonstrate the GC problem is:

```
10 INPUT Q,K
20 DIM L$(Q)
30 FORI=1TO Q
40 FORJ=1TOK: L$(I)=L$(I)+CHR$(64+J)
50 NEXTJ
70 PRINTL$ (I),I:NEXTI
```

Here a first order array with dimension Q is established. Each element is formed from the ASCII code starting at A and contains K symbols. For K=26, each element of the array contains the upper case alphabet in order. At the conclusion of filling each of the array´s Q elements with K symbols the element is printed followed by the element number.

For an OSI BASIC in ROM machine with 8K of memory, and K=26, the program will run for Q<=18. If Q exceeds 18 the GC routine fails. Similarly, if K=62, failure occurs for Q>3.

CIRCUMVENTING THE PROBLEM:

Ideally, one would like to correct the errors in the ROM program. One could reprogram a 2716 EPROM with the correct code and substitute it, after some wiring changes, for the incorrect ROM. This is not a simple solution to execute. There is, fortunately, a simpler approach that is useful.

The enclosed listing is a BASIC program that, when run, places a corrected GC program at the top of memory. It protects the program from being written over by other BASIC programs. It also sets USR function pointers so that the program can be called by X=USR(X). Finally it displays on the screen two useful pieces of information. It provides a POKE statement that may be needed to reset the USR pointers if they are changed by another program. It also provides, in decimal, the LOCATION of the GC program called by USR(X).

The steps to use this approach are as follows:

1. Cold start
2. Load the program
3. Run the program ONCE. (each time the program is run after cold start "memory available" is reduced in increments of the program length) Run time is about 15 seconds.
4. Record the POKE data and LOCATION data for future use
5. Type NEW and LOAD the program to be run that contains string arrays
6. Insert X=USR(X) in the program after each major concatenation to call the corrected GC. Place the POKE statement before this call if USR(X) is used elsewhere in the program to be run.

Take the general program listed above. Add the following line:

60 X=USR(X)

This cleans up the garbage left after the completion of each string array element, L$(I). With this addition, an 8K machine with J=26 will now operate for Q=75 instead of 3 as before. For J=26 the program will operate for Q over 200. Unfortunately, after 50 or 60 elements are generated, the program slows down noticeably since the GC is moving a large number of strings. This is the penalty paid for being forced to call the GC more frequently than is necessary. It is better to err on the side of conservatism because if the internal GC is triggered the program Bombs.

LIST

```
10  X=PEEK (133):Y=PEEK (134)
20  L=256*Y+X:L=L-262
30  Y=INT (L/256) :X=L-256*Y
40  POKE133,X:POKE134,Y
50  POKE11,X:POKE12,Y
55  PRINT"POKE11,";X;":";POKE12,";Y
60  PRINTL;:A=45383:B=45644
70  K=L:FORI=ATOB
80  IFI<>A+34THEN100
90  M=K+146:GOTO230
100  IFI<>A+159THEN120     IFI<>A+57
110  M=K+140:GOTO230
120  IFI=A+67THENPOKEL,4:GOTO220
130  IFI<>A+84THEN150
140  M=K+209:GOTO230
150  IFI<>A+137THEN170
160  M=K+146:GOTO230
170  IFI=A+216THENPOKEL,2:GOTO220
180  IFI=A+217THENPOKEL,24:GOTO220
190  IFI<>A+261THEN210
200  M=K+4:GOTO230
210  X=PEEK(I):POKEL,X
220  L=L+1:NEXT:PRINT"LOCATION":END
230  Y=INT(M/256):X=M-256*Y
240  POKEL,Y:POKEL-1,X
250  GOTO220
```

C2-C3 TIME SHARING

Jeff Beamsley
Tek-Aids Industries Inc.
44 E. University Dr.
Arlington Heights, Ill. 60004

Did you know you can time share on any C2 or C3 serial computer under OS-65U LEVEL 1? You can by using the program MULTI on your OS65-U disk, with C1P's or C4P's as intelligent terminals. Let's look closer.

The Ohio Scientific Level 1 Multi-Terminal Operating System is a very simple implementation of a time shared system. It allows up to 16 serial devices with some level of local intelligence to share the resources of a disk-based system. In the version that is delivered by Ohio Scientific, "MULTI" on the Level 1 OS-65U diskettes, the common shared resources are the floppy disk drive, through the L and S commands, and a printer, through the P command. You are not limited to those commands, however. The "MULTI" program is written in Basic and the structure of the program is very straightforward. The addition of new commands as well as the modification of existing ones is very easy as a result. The same cannot be said, however, for the hardware construction of the system. The documentation in this area is at best ambiguous. Beyond that, if you attempt to construct the system as the Level 1 documentation implies there is a fairly good chance that it won't work. In the following paragraphs I will describe the theory of operation of the Level 1 system, one possible hardware implementation of that system, and an example of the installation of an additonal command.

The theory of operation of the Level 1 system as well as the recommended hardware implementation is covered in considerable detail in the Ohio Scientific Multiple User Computer System Manual. This article is not intended as a substitute for that manual, but rather an addition to it.

The Level 1 system is based on the Device #8 I/O driver, the CTS function of the ACIA, and the REM statement in Basic. The Device #8 I/O driver for the CA-10 multiple serial I/O port board has several unique features. When an "INPUT #8" occurs, the input driver will scan all of the possible 16 serial ports on the CA-10 board until it receives an input from one of them. It will then POKE the appropriate input and output locations to communicate with that active port. This feature relieves the programmer/operator from worry over which port may be active at any one time. The next active port in the order of the scan will always be the next one accessed. This would seem then to be a great way to run a multi-user system. A problem arises, however, when two ports are active simultaneously. The software can service only one port at a time and will continue to service that port until it is done communicating. There has to be some way of preventing transmission from all of the other potentially active ports until the currently active port has concluded its conversation. This is accomplished in hardware by toggling the CTS line (pin 24) on the ACIA's of each of the intelligent terminals communicating with the CA-10 board in the host system. The CTS line must be low (0 volts) for the intelligent terminal to put a character into its transmit buffer for transmission. If we tie that line in each of the intelligent terminals to a line that we can control from the host machine, then it is possible to communicate with one active port at a time and not lose any data from the other potentially active ports. The Level 1 system uses the RTS line (pin 5) of the ACIA's on the CA-10 board to do this toggling.

The final problem arises at the intelligent terminal end. Assuming that we are using "BASIC-IN-ROM" machines populated with a serial port, it is no problem to turn that serial port on or off through software. It is treated the same as a cassette port via the LOAD and SAVE commands. The method of passing commands back and forth through that port to the host system once that port has been "SAVEd" on is not clear given the limited capabilities of the BASIC-IN-ROM machine. The command should be executable from the immediate mode, but there are only a few commands that can be executed in the immediate mode that won't result in a syntax error. All but one of them cause some action that might not be appropriate for terminal-host communication. REM is the perfect candidate for a command-passer. It requires no action on the part of the terminal and it places no restrictions on the format of the command used to pass information to the host. We will take advantage of this flexibility when we add additional commands.

So there you have it. The Device #8 driver scanning the ports for input. The Level 1 software simultaneously toggling the CTS lines of the intelligent terminals with the RTS lines of the CA-10's ACIAs. The instant the Level 1 software receives the "R" of REM it locks on to that port and looks for a space and the appropriate command (L,S, or P). It then looks for another space and the name of the file to be handled. From there it is all downhill.

I have made several choices in this hardware implementation. For the sake of brevity let's discuss only the 502 board based intelligent terminal. There are several unique problems with implementation of this type of system on earlier BASIC-IN-ROM systems, but none are insurmountable. Also I recommend that the clock used for the Level 1 transmissions come from the CA-10 board rather than attempting to generate it internally in each of the intelligent terminals. For BASIC-IN-ROM machines this method has the distinct advantage of maintaining cassette compatibility for "off-line" use.

Let's follow the convention of the Level 1 manual and configure the CA-10 board first. The RS-232 transmit and receive data are already at the connector. All we need to concern ourselves with are the RTS and clock signals. Ohio Scientific recommends the use of the CA-10's spare 7417 output drivers to buffer the RTS lines from the ACIAs. It has been my experience that these drivers are very sensitive to line length and environment. Unless you are going to populate all 16 ports on the CA-10 board, I would advise use of the 1488 drivers to buffer addtional lines coming off the board. If you run out of 1488 drivers, it is not very difficult to install them in the unused 7417 sockets and rewire the board appropriately. One thing we do need to keep in mind with this approach is the inversion that the 1488 driver introduces in the RTS signal. Depending on the number of intelligent terminals in the system, you might also have to be concerned about the fan-out of the clock signal line. I usually limit the number of terminals tied to one 1488 clock driver to 4.

As the number of intelligent terminals connnected to the CA-10 board increases, so do the problems of cabling. I recommend that the RTS signals be run to the spare grounds seperating each ACIA's I/O (pins 4,7,10,13, etc.). The free grounds can be bussed externally to pin 1. I also suggest that the clock be run off the J2 extension connector on the outside edge of the board (pins 50-61). This keeps those signals out of the inevitable jumble along the top of the board and makes bussing those signals much easier. As a final note on cabling, I would also suggest that you use an intermediate cable connector (Molex works very well) between the board and the individual DB25S connectors. This allows you to construct the cables separately and makes it much easier to remove the CA-10 board from the system.

Modification of the 502 board is similarly straightforward. There are only three general functional blocks that require attention. First the RS232 transmiter and receiver must be populated. Then we need to buffer the incoming RTS and clock signals. Finally some attention must be paid to how the system can be best cabled to allow easy transition from Level 1 I/O to cassette and back.

The construction of the RS232 transmitter and receiver is covered in some detail under procedure II of the "Modifications to 502 Boards" section of the Level 1 manual. There are also two foil cuts and a jumper that must be made per procedure V. The first cut prevents the output of the synchronous output driver, U41 pin 2, from affecting the output of the RS232 output driver. The second cut and jumper wires the -9 volt side of the RS232 output driver to ground. This is necessary because the BASIC-IN-ROM machines don't have a -9V supply. If you are contemplating using this serial interface in applications other than the Level 1 system where the -9 volt swing on the RS232 output would be important (e.g. a local printer), don't make this modification. Wire the -9 to a spare pin on the DB25S connector instead and use the same method we will describe later to connect it to ground when in the Level 1 mode. This leaves you free to "steal" -9 volts and get it into the machine for some other application. U20 should also be populated if it isn't already.

There is foil on the board for buffers for both the RTS and clock signals. The component values for both of these receivers are the same as those for the RS232 receive data buffer Q4. Procedure IV of the Level 1 manual instructs you to bypass an inverter on the ACIA SP.I buffer. Because of the inversion we received on the CA-10 board from the 1488 driver, this is unnecessary.

At this point we have all the proper signals coming into the board. Now we need to determine how to route these signals to allow us to use the machine on-line in a Level 1 system and off-line as a cassette machine. There are three signals that determine what state the BASIC-IN-ROM is in. Receive Data is coming either from the cassette interface or the Level 1 RS232 input buffer. The clock is being generated either internally or on the Level 1 CA-10 board. Lastly, the CTS signal is connected either to ground or the RTS line from the CA-10 board. One approach that is certainly valid is to mount a 3PDT switch on each BASIC-IN-ROM machine with each of the three signals as a pole and one throw the Level 1 mode and the other throw cassete. Switches of this type are not cheap, however, and you still have the problem of accidental switching and proper switch polarity for each mode of operation. A somewhat less expensive method that insures the proper signals are in place for each mode of operation is available using the unused pins of the DB25S connector to route the signals. One possible pin-out of a DB25S socket using this idea is the following.

Pin Number	Signal Name
2	RS232 Receive Data
3	RS232 Transmit Data
5	RTS
7	Ground
19	Level 1 clock
20	Cassette clock
21	System clock
22	Level 1 clock (TTL level)
23	RS232 Receive Data (TTL)
24	System Receive Data
25	Cassette Receive Data

Now if we wire pin 21 to 22 and pin 23 to 24 in the DB25P cable that connects the BASIC-IN-ROM machine to the CA-10 board, all the proper connections for a Level 1 system will be made when the cable is plugged in. Further, if we construct a dummy DB25P plug with pin 20 wired to pin 21 and pin 25 connected to pin 24 and plug it in, the BASIC-IN-ROM machine will be able to read and write cassettes.(Note: the RTS signal floats high when not connected and effectively grounds CTS permitting transmission.)

The cabling of this end of the system requires some care. I recommend using the J3 connector for all signals coming into or out of the 502 board. J3 pin 8 through pin 12 are available for DB25S pin 21 (System clock) through DB25S pin 25 (Cassette Clock). The spare ground of J3 pin 6 can be used for the remaining signal on DB25S pin 20 (Cassette Clock). The following cable diagram summarizes this configuration.

Procedure I of the Level 1 manual describes the modifications required to connect RTS to CTS (refer to 502 schematic page 7 of 12). The board also provides a convenient area to wire the clock switch (W2 on the 502 schematic page 7 of 12). The System Receive Data switch is not documented in the schematics, though there is also an optimum spot to wire it. That spot is located at a feed-through directly above U12. The foil going from that feed-through to ACIA pin 2 represents System Receive Data. The foil leading from that feed-through on the parts side to U20 pin 2 represents RS232 Receive Data (TTL). The foil leading from that feed-through on the foil side to U23 pin 5 represents Cassette Receive Data. Some foil cuts and jumpers will easily install this switch. This completes the modifications necessary to install a Level 1 system.

J3 Pin		CA-10 Port 1	
1	RS232 Receive Data --------	2	RS232 Transmit
2	Ground --------------------	1	Ground Buss
3	Level 1 Clock -------------	50	Level 1 Clock
4	N.C.		
5	CTS -----------------------	4	RTS
7	RS232 Transmit data -------	3	RS232 Receive
6	Cassette Clock ------------	i	
8	System Clock --------------	1	
9	Level 1 Clock (TTL) -------	'	
10	RS232 Receive Data (TTL) --	i	
11	System Receive Data -------	¶	
12	Cassette Receieve Data ----	'	

The operation of this system is well described in the "Multi-Terminal Timesharing Executive" portion of the Level 1 manual. The Executive program, MULTI, is the key to its operation. This program is written in Basic so it is easily modifiable. The current commands implemented are L(oad), S(ave), and P(rint). As an example of the flexibility of this system let's add an additional command. The current system allows printing of program listings only. Let's add a command to print data called D(ata). The MULTI program is roughly structured into three parts. The first two set up the variables for the system and look for the REM statement that initiates communication. The third decodes the commands. This is the only area of the program that requires any change to add an additional command. The listing below of the D(ata) command is a simple modification of the existing P(rint) command. The only unusual restriction placed on the transfer of data to be printed by this program is that the string "OK" will terminate data transmission.

```
731 IF C$<>"D" GOTO 740
732 PRINT "PORT";N;"PRINTING DATA"
733 PRINT #DV: PRINT #DV,"FROM TERMINAL";N:

734 INPUT#8,S$: M=PEEK(TN)/2
735 IF N<>M GOTO 120
736 IF S$<>"OK" THEN PRINT#DV,S$
737 GOTO110
```

This command differs from the implemented commands because it can be initiated under program control from the BASIC-IN-ROM machine. An example of how this might be done is included below.

```
 5 FLAG=517
10 POKE FLAG,1          |(Turn on SAVE mode)
20 PRINT"REM D"         |(Output command)
30 FOR X=1 TO 10
40 PRINT X
50 NEXT X
60 PRINT "OK"            (End transmission)
70 POKE FLAG,0           (Turn off SAVE mode)
```

This program will print out numbers 1 to 10 on the Level 1 printer. This is a simple example and there is no provision to insure that a BASIC-IN-ROM machine in a Level 1 system using this command teminates its transmission with the proper string. There is also no provision to prohibit the BASIC-IN-ROM machine from keeping control of the Level 1 machine through this command for an inordinate amount of time. Both of these problems are fairly easily solved with tools present in the standard MULTI program. Careful study of the Level 1 listing will reveal those tools.

Several other useful commands that would greatly increase the power of the Level 1 system are F(ile), a command to dump data into the BASIC-IN-ROM machine; R(un), a command that will both load and execute a program file; C(reate), a command to allow creation of program and data files from the BASIC-IN-ROM machine; and E(rase), a command to erase program and data files. Another useful utility would be some sort of "mailbox" system to allow one BASIC-IN-ROM machine in the Level 1 system to talk to another using a file or series of files on the host's disk. Each of these commands can be implemented in the MULTI program. Some are obviously more complicated than others, but they are all possible.

The Ohio Scientific Level 1 system has great application in the educational market where this type of BASIC-IN-ROM machine network can greatly speed machine throughput. Educators who are adventurous enough to delve into some of the command structures discussed will also find the Level 1 system an excellent tutorial in the construction of operating systems. This type of application is really only the begining for this network. Depending on the BASIC-IN-ROM machine that is used, the "per-user" cost for Level 1 systems can easily drop below $1K. With some work, in fully populated systems, the cost per user can approach $.5K. These days it is very difficult to purchase just a CRT terminal for less than $.75K, much less an intelligent terminal in a time-shared network. Some very obvious applications come to mind quickly. These Level 1 networks can be used as data collection terminals in a production environment. One terminal at each workstation, for example, recording parts used or, in conjunction with the real-time clock, production time per piece. In hospitals and nursing care homes, these networks can be employed to update patient information and disseminate instructions. In those types of institutions, these networks have the added advantage of logging all inquiries and information transfers and can optionally be used, in conjunction with the real-time clock, to log employee activity. These systems can even be used in more traditional business oriented multi-user systems. The BASIC-IN-ROM machines can collect and buffer data for transfer to some larger machine's disk based files all at a very low per-station cost. If the concept of memory buffering is exploited, the BASIC-IN-ROM machines can store a considerable amount of information before they need to make a transfer. This sort of programming will cut down on the total number of transmissions from BASIC-IN-ROM machines to the Level 1 host machine and thereby lower the incidence of simulataneous or queried requests and improve the individual terminal response time.

The Ohio Scientific Level 1 multi-user system is a very interesting and versatile implementation of some very basic concepts. It works suprisingly well and, due to the independent nature of each of its terminals, is relatively difficult to crash. The most amazing thing of all is that there is a real industrial problem solver lurking in the Ohio Scientific Level 1 system in the garb of an educational toy. Because of my interest in the diverse applications for this type of system, I would welcome any questions or dialogue concerning unique applications.

LETTERS

ED:

Been in love with my CIP Superboard for a year now, and have at times gone on computer-mania binges that were halted only by threat of divorce. One of these sessions led to the near completion of a match-up of the Superboard to a KIMSI S-100 adaptor. Would very much like to know if someone else has worked out the last few glitches remaining; even to any other buss adaptor.

About the slow 300 Baud rate that Paul Bowen complained about, I happened to purchase a how-to-do-it sheet from Aardvark and have had zero error rate at 600 Baud. It really is quite simple for all that is needed is to cut a few foil traces and wire in several jumpers, add one small common capacitor, hook it up to a DPDT switch so you can have 300 and 600 Baud. Sit back and enjoy the speed.

Paul Savard
Kalkaska, MI

ED:

As an answer to Paul Bowens´ letter, you can speed up the cassette part by moving pin 2 of V57 from (C4) pin 14 of V59 to (C0) - (C1) - (C2) or (C3) of V30. You will have to reset R57 the 10K pot on V69 of the receiver. What I did on my C4 was to install a switch to change the clock speed of the trans., and a new second pot in the receiver so I could speed from 300 Baud to 600 or 900 Baud. In the C4 the timing cap on the NE555 must be changed to change the speed of the clock. Also, 600 and 900 Baud takes a good tape recorder.

I like PEEK (65) very much. Keep up the good work.

Andrew Hubbell
Adrian, Michigan

ED:

Please excuse my being a "dumb" computer person but....Have a program written for TRS-80 that I would like to use on my OSI C3-S1....however, some things in the TRS program confound me somewhat, for instance:
"CLS" for a TRS means what for a OSI?
"ELSE" for TRS valid for OSI?
"OUT255,4" for TRS means what for OSI?
"OUT255,0" for TRS means what for OSI?

Does "BREAK" in programmer jargon mean the "space bar" in typist jargon??

Appreciate your help!!!!

Larry Kiner
Redmond, Washington

ED:

I am glad to hear that PEEK (65) is going to thrive. I´ll be looking forward to future articles, especially on hardware.

I´d appreciate it if you´d mention to your readers that I have copies of a hex dump of the S.II´s monitor ROM (all 2048 bytes) for $1.00 per copy. I´m also looking for a machine code programer to find the video RAM scroll routine in the aforementioned firmware.

Here are some questions about the S.II/C1 that I´d like to have answered: How and when do the the REPEAT, RUBOUT and ESCAPE keys work? How can one copy a machine code program onto cassette?

If any one wants to compare notes on how to double the line length of the S.II or C1 video display, my address is:

Bruce Showalter
857 Cedar
Abilene, TX 79601

ED:

This is in response to the plea from Arthur Fink in your second issue of PEEK (65). Fink says he can't find anybody who can tell him how to copy track zero from one OS-65D disk to another. Maybe there's more to the question than I'm seeing, but what I (and any other user of OS-65D I've ever talked with) do is use OSI's handy-dandy little copy utility stored on track 1 sector 2 of 8"disks, track 13, sector 1 for 5" units. Here's the procedure:

```
 For 8" disks type:  DISK!"CA 0200=01,2
 For 5" disks type:  DISK!"CA 0200=13,1
```

You will see the following displayed on the screen:

```
    --DISKETTE UTILITIES--
      SELECT ONE:
      1) COPIER
      2) TRACK 0 READ/WRITE
      ?
```

Select number 2 from the menu and push RETURN. Next this will be displayed on the screen:

```
    --TRACK ZERO READ/WRITE UTILITY--
      COMMANDS:
 Rnnnn--READ INTO LOCATION nnnn.
 Wnnnn/gggg,p--WRITE FROM nnnn FOR p PAGES
               WITH gggg AS THE LOAD VECTOR
 E--EXIT TO OS-65D
               COMMAND?
```

This time type: R400 and push RETURN.

The same TRACK ZERO/WRITE UTILITY menu will be displayed again, indicating the operation is complete. At this point remove your master disk and replace it with the disk onto which you want to copy track zero.

Then type:

```
      W4000/2200,B (For an 8" disk) or
      W4000/2200,8 (For a 5" disk)
```

Next replace your original disk, choose E from the menu and push RETURN. Type GO 0200 and push RETURN. The original DISKETTE UTILITIES menu will be displayed and you may select 1 (the COPIER) and copy tracks 1-76 (8" disks) or 1-39 (5" disks) as usual.

All of this information is included in OSI's OS-65D users' manual, but, admittedly, the LOAD VECTOR address (2200) is pretty obscure. You have to really dig to figure that one out.

Hope I've helped Mr. Fink and others who have had trouble with this routine.

Tom Badgett
Bluefield, West Virginia

ED:

I would like to recommend Aardvark's ROM BASIC Data Sheet ($9.95) for anyone wishing to get a look at how Microsoft and the 6502 work together to implement OSI BASIC. It's only a rough framework, but it provides a good basis from which to dig out more information. Aardvark's DISASSEMBLED ROM LISTING ($8.95) is indeed, as advertised, 16 packed pages, five columns per page, 59 lines per column. This results in a format that is not as easy to read as it should be (the leftmost column is illegible in places), but still represents an advantage over trying to copy it down by hand from the TV. One warning: their disassembler does not correctly distinguish between indexed and indexed-indirect addressing modes when the X register is involved (eg. B7A1-B7A4). This can be extremely misleading to anyone trying to decipher the code, and one should check the mnemonics against the machine code (if legible) to prevent wasted effort. I have advised Aardvark of this shortcoming, and they do not appear able to distinguish between the unavoidable loss of sync that occurs when a disassembler encounters data and the failure of a disassembler to produce correct mnemonics. The product is good but could easily be improved.

Philip Hooper
Northfield, VT

ED:

Where do I send my money for the replacement BASIC ROM? You started the rumor so provide the rest of the information!! If not, I am going ahead with my plan to burn the corrections into a 2716 EPROM.

I would like to correspond with others who are using the OSI 440 video board with dot graphics option.

Earl Morris
Midland, MI

Earl:

It was just a rumor I heard at an OSIO meeting. Burn away!

Ed

ED:

I would like to submit for the PEEK (65) readers a routine they might find helpful in developing programs for card games. This routine is a fast, memory-efficient way of getting a truly random "shuffle" of a deck of 52 cards.

```
100 FOR Q=53248 TO 53299:
    POKE Q,0:NEXT

110 POKE 57088,254
120 FOR Q=1 TO 13
130 FOR R=1 TO 4
140 S =INT(RNDC5)*52:
    IF PEEK(S3248+5)>0
    THEN 140
150 POKE 53248+5,Q*10+R
160 IF PEEK (57088)=190
    THEN S=RND(S):GOTO 160
170 NEXT R
180 NEXT Q
```

The result of this routine is the storage of 52 random cards in bytes 53248 to 53299 of the Video RAM. The cards are stored as integers ranging from 11 (ace of hearts) to 134 (king of diamonds). The rank of the card (from 1-13) is obtained by using the formula R=INT(X/10) where R is the rank and X is the value stored in the byte. The suit of the card (from 1-4) is obtained by using the formula S=X-INT(X/10)*10 where S is the suit and X is the value of the byte.

The use of single bytes rather than subscripted variables for storing values of cards will save about 200 bytes of memory. The occasional pressing of the CTRL key during the few seconds it takes for the routine to execute will break up the pattern of pseudo-random numbers generated by the machine and will result in a unique shuffle each time. Note also that a "1" will have to be POKE'd into 530 to disable CONTROL C for the keyboard polling routine.

It is possible for any card game to be programmed, with the computer playing from 1 to 6 hands simultaneously, if the programer uses some ingenuity and has 8K of memory to work with.

David Hille
San Antonio, TX

ED:

Let me thank all of you on your first two issues and contribute the following:

1) The "string bug" on the OSI C1P - the garbage collector is the culprit of course (located at $A2B6). Try:

```
10 DIM A$(3)
20 PRINT FRE(0)
```

The hangup occurs. However, change 10 to

```
10 DIM A$(2)
```

and no problem occurs. In fact, no problem occurs whenever working with strings dimensioned of size N, where N=3*(any integer)+2.

2) Need to detect a key without having to turn on specific rows, then decode columns? This routine will detect a key - any key -without the Input statement. First poke the following anywhere in memory:
32, 186, 255, 133, 255, 96;
then poke the start address into 11 and 12 so it can be called as USR(X).

In the program use the following code:

```
110 POKE 57988,0
120 IF PEEK(57088)=254 THEN 140
130 X=USR(X):CHARACTER=PEEK(255)
140 (PROGRAM CONTINUES)
```

If there is no key pressed, the program continues. If there is, USR(X) is called, which puts the character code in 255.

3) There's a non-machine language screen clear that's really fast.

```
5000 LO=PEEK(129):HI=PEEK(130)
5010 POKE 129,5:POKE130,212
5020 A$="(48 SPACES)"
5030 FORI=1 TO 20:A$=A$+" ":NEXT
5040 POKE 129,LO:POKE130,HI
```

Mark Minasi
Stony Brook, NY

OK

ED:

Maybe you can help me with a mystery. Quite often lately I have been getting the following error code - B -. It is not anywhere in OSI's documentation. It is really baffling me as to what this error could mean.

Well, the most frustrating thing I have found in the OSI documentation is their SCATTERING of Pokes throughout the manual. So, I took it upon myself to list all the Pokes I could find, most of which were not shown in your last issue. If this helps anyone please pass it along.

Valerie Winer
Chestnut Hill, MA

23	Terminal Width	
24	No. of characters in fields (14)	
12Ø	Lo byte address of beginning of BASIC work-space	
121	Hi byte address of beginning of BASIC work-space	
132	Lo byte address of end of BASIC work-space	
133	Hi byte address of end of BASIC work-space	
222	Real Time MONiter (1-enables Ø-disables)	
223	Start Countdown timer (1-start Ø-stop)	
224	Hours to Countdown	
225	Minutes to Countdown	
226	Seconds to Countdown	
23Ø	Mask Register for Port 1A	DATA - corresponding Register
231	" " " " "	CONTROL
232	" " " " 1B	DATA
233	" " " " "	CONTROL
234	" " " " 2A	DATA
235	" " " " "	CONTROL
236	" " " " 2B	DATA
237	" " " " "	CONTROL
238	" " " " 3A	DATA
239	" " " " "	CONTROL
24Ø	" " " " 3B	DATA
241	" " " " "	CONTROL
249	Contains latest value of 56832 & can be PEEKed.	
548	Hi byte address for A C driver	
549	Lo byte address for A C driver	
741	"LIST" (76 - enables 1Ø - disables)	
75Ø	"NEW" (78 - enables 1Ø - disables)	
1797	Control line # list of BASIC (32-enables 44-defeat)	
2Ø73	"CONTROL C" (173-enable 96-disable)	
22ØØ	ROM direction of mask Ø to load at 222ØØ	
2888	Null input jumps out of program (Ø-disable 27-enables)	
2893	(28-"REDO FROM START" enable)	
2894	(11-"REDO FROM START" enable)	
2888,8722	Both Ø & null input to "INPUT" yields empty string or a Ø If Both-27 then input statement functions are normal.	
2972	(58-comma is a string input termination 13-disable)	
2976	(44-colon is a string input termination 13-disable)	
87Ø8	Output flag for peripheral devices (44)	
89Ø2	Determines which register (less 1) RTMON scans	
89Ø9	Hi byte address of PIA for RTMON scanning	
891Ø	Lo byte address of PIA for RTMON scanning	

8917	USR(X) operation code	
8944	Output flag	
8954	Location of JSR to disk USR(X) routine	
8955	Lo byte address of USR(X) pointer	
8956	Hi byte address of USR(X) pointer	
896Ø	Memory (RAM) page count minus 1	
8993	I/O distributor input flag	
8994	I/O distributor output flag	
8995	Index to current ACIA on 55Ø board	
8996	Location of random seed for RND function	
8998	Lo byte address of pointer to disk buffer 1	
8999	Hi byte address of pointer to disk buffer 1	
9ØØØ	Lo byte address of the end (plus 1) of disk buffer area	
9ØØ1	Hi byte address of the end (plus 1) of disk buffer area	
9ØØ6-13	Memory buffered disk I/O bit 6 device parameters	
9Ø98	Lo byte address for memory input	
9Ø99	Hi byte address for memory input	
91Ø5	Lo byte address for memory output	
91Ø6	Hi byte address for memory output	
9132	Lo byte address for memory buffered disk input	
9133	Hi byte address for memory buffered disk input	
9155	Lo byte address for memory buffered disk output	
9156	Hi byte address for memory buffered disk output	
9213	Lo byte address for memory buffered disk input	
9214	Hi byte address for memory buffered disk input	
9238	Lo byte address for memory buffered disk output	
9239	Hi byte address for memory buffered disk output	
9368	Hi byte address for indirect file input (low-ØØ)	
9554	Hi byte address for indirect file output (low-ØØ)	
9392	Active State Register for Port 1A	Data Register
9393	" " " " " "	Control "
9394	" " " " " 1B	Data "
9395	" " " " " "	Control "
9396	" " " " " 2A	Data "
9397	" " " " " "	Control "
9398	" " " " " 2B	Data "
9399	" " " " " "	Control "
94ØØ	" " " " " 3A	Data "
94Ø1	" " " " " "	Control "
94Ø2	" " " " " 3B	Data "
94Ø3	" " " " " "	Control "
9666	Move left margin to the right	
9667	Moves scroll up	
968Ø	This location contains the cursor character designation	

9822	Sector # for USR(X) Disk Operation
9823	Page count for USR(X) disk write
9824	Lo byte address of memory block for USR(X) Disk Operation
9825	Hi byte address of memory block for USR(X) Disk Operation
9826	Track # for USR(X) Disk Operation
12Ø42	Location of 24 used by random access file calc. routines.
53248-55295	Video Memory
56832	Tone Generator,character/line & Color on/off (Ø - 7)
57Ø88	Keyboard - Keypad - Joystick
57Ø89	Frequency for Tone Generator
58348	Specifies Color (Ø - 15)
63488	Telephone Interface
63489	" " Control Register for 63488
6349Ø	Telephone Interface
63491	" " Control Register for 6349Ø
63492	1PA4 MODEM self test 1PA5 MODEM squelch 1PA6 MODEM originate 1PA7 MODEM answer mode 1CA1 DTMF decodes strobe
63493	Control Register for 63492
63494	1PBØ - 1PB7 Dialer Data (1CB1, 1CB2 not used)
63495	Control Register for 63494
63496	ACIA
63497	Control Register for 63496
64256	43Ø board's A/D converter output
64512	BAUD RATE (1-12ØØ 2-3ØØ)
64513	ACIA Data Register (for printer or modem)

5Ø948	Data Register for Port 1A
5Ø949	Control " " " "
5Ø95Ø	Data " " " 1B
5Ø951	Control " " " "
5Ø952	Data " " " 2A
5Ø953	Control " " " "
5Ø954	Data " " " 2B
5Ø955	Control " " " "
5Ø956	Data " " " 3A
5Ø957	Control " " " "
5Ø958	Data " " " 3B
5Ø959	Control " " " "

OSI OSI

SOFTWARE FOR OHIO SCIENTIFIC

Over 50 programs for C1, C2, C4 & Superboard, on tape and disk. All come with listings and compete documentation.

GAMES - 4K - Tape

CHESS FOR OSI - specify system $19.95

STARFIGHTER 5.95
Real time space war.

SEAWOLFE 5.95
Floating mines, three target ships, etc.

LUNAR LANDER 5.95
With full graphics

TEN TANK BLITZ 9.95
A sophisticated real time tank game.

8K GAMES

BACKGAMMON 9.95

BLACKJACK 6.95
Plays all Vegas rules

Add $1.00 each for Color/Sound

UTILITIES

C1P CURSOR CONTROL $9.95
gives real backspace, one key screen clear, and **midline editing**

RENUMBERER 5.95

SUPERUTILITY 12.95
Has Renumberer, Variable table maker and Search

BUSINESS

SMALL BUSINESS ANALYSIS 15.95
Does profit and loss, quick ratio, breakeven analysis and more. 13 pages of documentation.

STOCK PORTFOLIO 6.95
Keeps track of your investments

Alien Invaders $6.95

Our $1.00 catalog has free game and utility listings, programming hints and a lot of PEEKs and POKEs and other stuff that OSI forgot to mention - and a **lot** more programs for sale.

DISKS 5" COLOR/SOUND $29.95
DISK 1. STARFIGHTER, ROBOTANK, SEA WOLFE, BOMBER, TEN TANK BLITZ
DISK 2 BREAKTHROUGH, LUNAR LANDER, ALIEN INVADER, KILLERROBOTS, SLASHBALL

AARDVARK
1690 Bolton, Walled Lake, Michigan 48088 • (313) 624-6316

Have you noticed that your 430 board won't work under OS65-U Ver 1.2? Here is a patch from D & N Microproducts which should restore it.

```
RUN"CHANGE","PASS

DISK CHANGE UTILITY

MODE: HEX(H), DEC(D) ? H
UNIT ? A
ADDRESS OFFSET ? C00
ADDRESS ? 3CCB
00003CCB    A2 ? AD
00003CCC    1E ? 06
00003CCD    A9 ? FB
00003CCE    03 ? A9
00003CCF    9D ? FF
00003CD0    00 ? 8D
00003CD1    FB ? 05
00003CD2    A9 ? FB
00003CD3    11 ? 4C
00003CD4    9D ? DB
00003CD5    00 ? 3C
00003CD6    FB ? .
ADDRESS ? 3CE9
00003CE9    AE ? AD
00003CEA    FA ? 05
00003CEB <  3C ? FB
00003CEC    BD ? 4A
00003CED    00 ? 90
00003CEE    FB ? FA
00003CEF J  4A ? AD
00003CF0    90 ? 03
00003CF1    FA ? FB
00003CF2    BD ? 8D
00003CF3    01 ? 07
00003CF4    FB ? .
ADDRESS ? 3D5E
00003D5E    AE ? AD
00003D5F    FA ? 05
00003D60 <  3C ? FB
00003D61    BD ? 10
00003D62    00 ? FB
00003D63    FB ? AD
00003D64 J  4A ? B6
00003D65 J  4A ? 38
00003D66    90 ? 8D
00003D67    F9 ? 04
00003D68    AD ? FB
00003D69    B6 ? 4C
00003D6A    CE ? 6E
00003D6B    9D ? 3D
00003D6C    01 ? X

OK
```

The patch is permanent to the disk.

THE GRAPHICS CHIP

✓

E.H. Carlson
3872 Raleigh Drive
Okemos, MI 48864

What do you make of (or with) the 128 non-ASCII characters available from the graphics chip? Pull out your Graphics Reference Manual and puzzle along with me. Or perhaps better, use the little program below to put pairs of graphics characters on the screen, because many of them do not look the same on the graph paper as on the screen.

```
10 Q=53248              40 POKE D,A
15 POKE 56900,0         42 POKE D+1,B
20 D=Q+1024+16          50 PRINT
30 INPUT A,B            99 GOTO10
```

There are some very good characters in the set. For example, the eight orientations of the tank of the arrow probably can't be beat by anyone. But..., there are also some real wonderments!

Characters 0 and 1 are two race cars, always going straight ahead. The road will have to be curvable to make an action display with them. I think 5 and 6, 7 and 8 are submarines going right and left. But what are 9-12? Perhaps 9 and 10 form the spaceship ENTERPRISE. On the other hand, 9 and 12 may be a biplane awaiting the Red Barron's bullets.

Now chess: 4 is the queen and perhaps 24 is the knight and 30 the rook. But here my imagination peters out. Where are the king and bishops? The pawns can be fudged in many ways, for example, by the up arrow 16. What are 25 and 28 supposed to be? Part of the chess set somehow?

The line segments 128-135 allow adjustments of a horizontal line by 1 pixel step. Likewise, 136-143 for vertical lines. The set 135, 145, 151, 155, 162-164, 158-161, 154, 150, 144 and 128 allow vertical bar graphs whose heights and base are adjustable to 1 pixel. The corresponding horizontal set is quite incomplete.

With 175-182 we are back to war, small gunboats. Larger ships can be made by tacking on the tails of the submarines, 5 or 8.