PEEK (65)

The Unofficial OSI Users Journal

$1.75

MARCH 1983
VOL. 4, NO. 3

P.O. Box 347
Owings Mills, Md. 21117
(301) 363-3268

INSIDE

Column One

OSI, the new Kendata OSI, not to be confused with the old M/A COM OSI or the ancient Charity Chieke OSI, has moved its corporate headquarters to

6515 Main St.
Trunbull, CT 06611
(203) 268-3116

What will go on at this new stand is Order Entry, Marketing and Sales Support. Product development will still be at the old (not ancient or prehistoric) stand, in Bedford, MA. Tech support will go back to the Paleozoic location at Aurora, OH, where they can be reached by the following toll-free(!) number:

Tech Support: (800) 321-5805

The new Masterkey 300 systems are now out in the field in Beta test (this means development is complete, the machines work, but just to be sure, they have several of them actually installed in working locations, undergoing the meanest kind of testing -- real use).

We announced last month that "most" 65U Basic programs will run on the new CP/M networking OS used in the 300 machines. How can you tell if yours will, before investing a wad of money in a new machine? Call the corporate headquarters at the above (203) number, and make an appointment to spend a few days there, trying out and, if necessary, modifying your programs. The Marketing department will have people available to help you.

If this doesn't sound like the old OSI (Beta test? A few days at headquarters with people to help you?), that's because it isn't the old OSI.

One of the things we are doing here is helping our readers to grow. I can think of no better example of this than Fred Schaeffer.

A couple of years ago, Fred started writing letters to PEEK(65). Basically a computer user, as distinguished from a programmer, Fred had a lot to learn. In fact, mostly his first few letters were stories of confusion and frustration (sound familiar?). Now, not too much later, Fred writes us frequently, describing his adventures with increasingly sophisticated modification and enhancement of the OS-DMS programs.

Of course, we can't claim complete credit for the education of Fred Schaeffer -- he is a clever guy, and has been reading about and working with computers a couple of years now, at least. However, he is a good illustration of what an (initially) unsophisticated user can learn by careful reading of PEEK(65), sharing of problems with our readers and hard work. Best of luck to Fred and all other computer enthusiasts!

This brings me to one of my favorite themes, here at PEEK(65) and elsewhere, and to another story.

I was recently invited to speak to a women's club called the 21st Century Club. Seems they had heard from a "futurist" at their last meeting, and the fellow had nothing but Doom and Gloom to predict for the future. At the end of the meeting, one of the members asked him, "isn't there anything cheerful you can predict?" "Sure," quoth the futureguru, "computers. The future for computers is rosy." So, hoping to hear something cheerful, they invited me to speak about computers.

The point is, computers are cheerful. Computers are encouraging, hopeful, powerful and increasingly accesible to us all. The few bad things about computers (depersonlization, inefficiency) are now beginning to be seen as what they were, essentially human problems, software problems, which can be and are being defeated by better programs and more powerful computers. The solution to the automobile pollution problem is not more automobiles; but the solution to the computer depersonalization problem is indeed more computers.

I hear some disagreement among the ranks. "If computers depersonalize us and louse up our accounts, how can we need more computers?" you ask. Easy. The reason we are depersonalized and our accounts are messed up is that the computers available a few years ago, the ones for which the programs now messing up our accounts were written, were limited things, with little capacity, low speed and great inefficiency. They couldn't handle the programs and file space needed to let them treat us as individuals, and the programs hadn't been written anyway. Programmers and users had to save every precious byte of expensive RAM; we had to work for the computers.

Now, things are changing. Every year we see computers with more memory, better software, higher peripheral storage capacity, for less money. Today's (and especially tomorrow's) computers can handle the job -- treating us more as individuals and less like account numbers. Interactive, on-line processing of huge data bases is a reality, and is transforming the way we use and perceive computers.

THE C1P MEMORY MAP

By: Steven P. Hendrix
Route 8, Box 81E
New Braunfels, TX 78130

Sure, you already know all about the memory map of the C1P. After all, OSI included a table in their manual, giving all the locations for RAM, ROM, and input/output devices. What more do you need to know?

If you want to expand the C1P with non-OSI devices, or just understand your machine and some of its unexpected responses, however, you need to know some things OSI left out. There are several large spaces where you can add memory which Basic won't touch (great for hiding your favorite machine language routines from Basic), and all of the I/O devices appear a number of times in the map. There is also a location which deals with the real-time clock which is not even mentioned in any OSI documentation that I've seen.

The hardware of the C1P breaks down something like this:

$0000 - $0FFF	4K RAM
$0000 - $1FFF	8K RAM
$0000 - $3FFF	16K RAM
$0000 - $5FFF	24K RAM
$0000 - $7FFF	32K RAM
$A000 - $BFFF	BASIC ROMS
$C000 - $C020	DISK I/O
$C000 - $C003	PIA
$C010 - $C011	ACIA
$C020	REAL TIME CLOCK
$D000 - $DFFF	VIDEO TERMINAL
$D000 - $D3FF	VIDEO RAM FOR UP TO 32 COLUMNS
$D000 - $D7FF	VIDEO RAM FOR 48 OR 64 COLUMNS
$DF00	KEYBOARD
$FC00 - $FFFF	1K MONITOR ROM (THE STANDARD ROM)
$F800 - $FFFF	2K MONITOR ROM (AFTER-MARKET ADD-ON TYPES)

published monthly

Editor - Al Peabody
Technical Editor - Brian Hartson
Circulation & Advertising Mgr. - Karin Q. Gieske
Production Dept. - A. Fusselbaugh, Ginny Mays

Subscription Rates

US (surface)	$15
Canada & Mexico (1st class)	$23
So. & Cen. America (Air)	$35
Europe (Air)	$35
Other Foreign (Air)	$40

All subscriptions are for 1 year and are payable in advance in US Dollars.

For back issues, subscriptions, change of address or other information, write to:

PEEK (65)
P.O. Box 347
Owings Mills, MD 21117

So far, I haven't shown anything that wasn't in the manual, except for the real-time clock. (Don't get excited about that yet. If you go looking for it without reading the explanation later you'll be disappointed.) It would appear from this memory map that there is quite a bit of space to add memory and other devices, even though it's all chopped up into little pieces. You'd think they could have done the job a little more neatly and left the extra space in one or two big chunks. Actually, this way took a little less hardware (read that $$) for each copy of this model, and it was a low-end computer. When the C1P came out, it was probably the best buy going at the low end of the market, but they took some shortcuts to achieve that.

The biggest unused chunk of the address space is between the end of RAM and the beginning of the Basic ROMs. This is a great spot to add RAM, since you can add a full 8K, and Basic will automatically use it when it tests for the end of memory to set up its pointers. How'd you like to see "40191 BYTES FREE" the next time you boot up Basic? A number of users have added memory like this. There is a slight complication if you do any direct manipulations of memory or do logical operations on 16 bit data (AND,OR). Basic is somewhat inconsistent in the way it handles 16-bit binary numbers. For PEEK and POKE, it allows decimal equivalents to range from 0 to 65535. For most other uses, it uses -32768 thru 32767. Numbers from 0 to 32767 are treated the same way in any case; 32768 thru 65535 correspond to -32768 thru -1. A number of HEXDOS users have written to me saying that the direct memory load and save functions will not work above 32K. The solution is to subtract 65536 from the address, resulting in a number in the correct range.

The next area which appears usable for added RAM lies from $C021 thru $CFFF. This area is not so simple, however. The disk and clock ports appear at more locations than are specified. The entire $C0xx page is taken up with multiple appearances of these devices.

The high-order eight bits of the address are fully decoded and included in the logic for each device in this area, but the lower eight bits are not. Bits 6 and 7 of the address are not considered at all, causing each device to appear four times in the $C0xx page; for instance, the register of the PIA appearing at $C000 also appears at $C040, $C080, AND $C0C0. The ACIA data register appears at $C011, but also at $C051, $C091, and $C0D1. Thus, we can consider the $C0xx page as being made up of four sub-pages, each consisting of 64 locations, and each of which actually access the same 64 hardware locations.

Bit 5 of the address selects whether we are accessing the components related to the disk, or the real-time clock. If this bit is a "0", the disk ports are active; if it is a "1", we may tap the clock. Converting this fact to hex addresses, we see that addresses such as $C00x, $C01x, $C04x, C05x, etc. will be related to disk access.

Next, bit 4 of the address determines whether the processor accesses the PIA or the ACIA: a "0" calls for the PIA, while a "1" accesses the ACIA, so $C00x and $C04x will be for the PIA, while $C01x and $C05x are for the ACIA.

The PIA has four addressable register locations, requiring two address bits to select between them (the contents and use of these registers is beyond the scope of this article). Bits 2 and 3 are ignored, so even within the area defined by the address $C00x, each register is repeated 4 times. The register at $C000 also appears at $C004, $C008, and $C00C, etc. Thus, each register is accessable thru a total of 16 different memory locations (4 times in each of the 4 sub-pages described above). Address bits 0 and 1 are used to select one of the four possible registers on the PIA.

If bit 5 of the address is low and bit 4 is high, we are accessing the ACIA. This causes it to appear at $C01x. Of the low-order 4 bits, only bit 0 is used, causing the ACIA to appear eight times in each sub-page (one for each of the eight possible combinations of the three bits 1, 2, and 3). Bit 0 is used to select one of the two register locations provided by the ACIA.

To understand the relationship of the clock in the address space, it is necessary first to describe the clock and how it is intended to be used. The "real-time clock" must be implemented in software. The hardware support produces an interrupt at some preselected interval, and the software you provide to service that interrupt must carry out the action you desire, such as incrementing a memory location or whatever. There are 18 possible intervals to select from (not 24 as it would first appear from the schematic), ranging from 2 microseconds to 1 second. Ways to use this facility could take up another whole article (and might!), but the reason I include it here has to do with a conflict between the clock and disk.

If interrupts are active in a system, it is possible for them to disrupt time-critical processes like disk access. For instance, in reading from the disk, a character is available every 88 microseconds, with buffering for two characters in the ACIA. Thus, if the processor is busy for more than about 80 microseconds in servicing an interrupt, a character may be lost, and if the interruption takes more than 170 microseconds, it will certainly cause one or more characters to be lost. Since most interrupt service routines will take longer than these critical times, we must ensure that they do not occur during disk access or other similar tasks.

A write to location $C020 is decoded to allow for such processes. Writing to that location triggers the Clear input of all the counters in the chain which produces the interrupt, resetting them to zero and guaranteeing one full interval before another interrupt. If the clock is set for intervals of at least 200 milliseconds, this delays the interrupt for at least long enough to read or write one full track from the disk. Thus, if the clock interrupt is active, we must do a write to $C020 immediately before any disk accesss. This can be a STA, STX, or STY instruction, an INC or DEC, or any other instruction which would modify a location. The data to be stored there does not matter. HEXDOS and, presumably OS65D, do this automatically before accessing the disk.

As with the other two devices in page $C0xx, this clear line appears many times. Any $C0xx address with bit 5 high will reset the timer on any write instruction ($C020, $C021, $C0FF, etc.).

The rest of the $Cxxx area is unused. This would allow space for 3-3/4 K of RAM or ROM. Most practical designs would insert 2K starting at $C800, using either RAM or a custom-programmed EPROM. The beauty of using this area for RAM is that it is automatically protected from Basic - you can load whatever you like into this area and it remains there until you turn off the power (and not even then if you add battery backup). Basic stops at end of contiguous RAM, which in any case will force it to stay below 40K even if you fill in the area at $8000 - $9FFF. Unless you deliberately modify it with POKEs or machine language this area is well protected against runaway programming. You can even cold-start or reboot from disk without affecting it.

Next comes the video RAM. On a stock C1P, there is 1K of RAM from $D000 thru $D3FF. If you have added one of the many video mods which yield a 32 x 64 screen or some subset of it, you will have added memory at $D400 thru $D7FF. Some of the kits simply use the existing memory and re-format it to a 16 x 64 screen, with some subset of that being visible on the screen; with those, your memory will still be in the standard area.

It appears that the address space is clear from $D400 (or $D800) thru $DF00. However, the keyboard port, which supposedly appears only at $DF00 took a bit more than its share of the address space. In fact, only the high-order 6 bits of the address are decoded to select it, causing it to appear to take up 1K of memory space. Any address from $DC00 thru $DFFF will cause the keyboard port to respond, as I learned after a frustrating attempt to debug a peripheral board I designed using some ports at $DC00 thru $DEFF.

If you leave space for possible future video RAM at $D400 thru $D7FF, and consider that the keyboard uses $DC00 thru $DFFF the $Dxxx area has only 1K of free space. While you could use two 2114's to fill that 1K, it seems like a very small block and is probably best used for I/O ports at $D800 thru $DBFF.

Moving on past the keyboard, the entire $Exxx area is unused on both the 600 board and the 610 board. This is a promising area for RAM, since you can add a 4K contiguous block here. However, keep in mind that manufacturers will also be tempted to use this area, so some add-ons on the market may conflict with anything you add here. This is still a good area for adding RAM, since it is large and has all the protection I described above for the $C100 thru $CFFF area.

The cassette ACIA appears in the published memory map at $F000 - $F001. Only the high-order 8 bits of the address are decoded to activate the chip, so it fills the $F0xx page. Bit zero of the address selects either the control register or the data register, so these two registers alternate through the page with the ACIA status appearing at all even locations and the receive -data register at the odd locations, during reads. For write operations, writing to an even location affects the control register, and odd locations output data to be transmitted.

The area from $F100 thru $F7FF is unused, and could hold RAM or ROM, but because of the piece missing at $F0xx, it is perhaps best used for I/O, like $D800 thru $DBFF. 1K would fit at $F4FF, however.

The stock monitor ROM is a 2708, which is only a 1K ROM, filling the space from $FC00 thru $FFFF. However, bit 10 of the address appears at the proper pin to allow use of a 2K ROM, so most of the after - market replacements which are available come in 2716's, which hold 2K bytes. Since the 2708 ignores bit 10, it appears twice. The same data appears at $F800 thru $FBFF as at $FC00 thru $FFFF.

OSI made provisons to allow moving the RAM on the 610 board to different areas of the memory map. I use this feature frequently to substitute some of the RAM for ROMs, so I can test proposed changes without burning a new EPROM every time I discover a bug. This is how I developed the patches to Basic which appeared in my article in the August and September issues of PEEK(65). You must of course arrange a switch to disable the ROMs. You can disable the Basic ROMs by cutting the trace from pin 10 of U23 to pin 6 of U15. Insert a switch to close the break, and you can turn Basic on or off at will. To disable the monitor ROM, you can disconnect pin 1 of U19 from +5, and tie it to

ground through a switch which will then function to disable that ROM.

Now for moving the 610 board memory around, look just to the left of U18 on the 610 board (looking from the end nearest the 600 board's keyboard). You should see a row of eight pads adjacent to U18, and a row of three pads just to the left of that. The row of eight corresponds to each 8K area of the memory space, starting from the front and moving toward the back. That is, the pad nearest the keyboard end of the board is activated when the processor accesses anything at $0000 thru $1FFF. The next pad toward the rear of the board is for $2000 thru $3FFF, and so on to the rearmost pad, which is $E000 thru $FFFF.

The three pads on the left correspond to the three 8K banks of memory on the 610 board. The two rows nearest the keyboard end of the board are activated by the front of the three pads, the middle two rows, by the middle pad, and the back two rows, by the rearmost pad. Upon close examination, you will see that the front bank is tied to the $2000 thru $3FFF select line, the middle bank is tied to $4000 thru $5FFF, and the rear bank is tied to $6000 thru $7FFF. I have cut all three traces between the two rows on my system, and soldered a socket into the row of eight and #22 wires into the row of three. With this arrangement, I can move any given bank by pulling its wire from the socket and attaching it to the select line for the area where I want it to appear.

```
                   -----
         []E000-FFFF|
         []C000-DFFF|
         []A000-BFFF|
         []8000-9FFF| U18
REAR BANK[]==[]6000-7FFF|
MIDDLE BANK[]==[]4000-5FFF|
FRONT BANK[]==[]2000-3FFF|
         []0000-1FFF|
                   -----
```

For a very simple experiment, you can fool Basic into thinking that you have 40K of RAM even if you have only 16K. Since the memory test only checks to see that it can store and retrieve data from the memory, you can fool it by making the same 8K block appear at all locations from $2000 thru $9FFF (I am assuming a full 8K on the 600 board). It does no harm to tie the lines on the right together, so you can simply run a wire from each of the four lines (second thru fifth from the bottom of the diagram) to the select line for the front bank. Now Basic will report 40191 bytes free when you cold start. However, that is not usable memory, since storing anything beyond your real memory will alter what was stored in the lower area, so this is really of no particular use unless you want to be unscrupulous when you sell your machine (the buyer wouldn't figure that one out for months, and by then you'd have skipped the country....).

Onward to more practical matters. You now have a choice about where to have RAM in your system. There are many ways to add the other 8K, but I will describe the method I used. In upgrading my system to 2MHz operation, I discovered that only about half of my 2114s were fast enough. Rather than buy more of those power-hungry beasts, I chose to use the new 61128 16Kx8 RAM, which is cheaper and uses far less power. It is best used as a contiguous 16K block of memory. I pulled all the user RAMs out of the 600 board, and left only two banks populated on the 610 board, replacing the entire lower 32K with two 61128's. Then, I moved one bank on the 610 board to $8000 thru $9FFF, leaving the other bank free to be moved around to replace ROMs. I added the 61128's by wire-wrapping 2 28-pin sockets on an accessory board which contains several gadgets I've already added to the system. The one at $0000 thru $3FFF takes no extra decoding: A13 thru A0 and D7 thru D0 tie to the corresponding lines on the bus, DS1 ties to the phase 2 clock, DS2' ties to A15, and DS3' ties to A14. For the memory at $4000 thru $7FFF, add one inverter to invert A14 so that the part is activated by those addresses.

There is one small additional requirement for any device on the bus, other than those on the 600 board. The data bus buffers on the 600 board and the 610 board are normally set to transmit data outward from the processor to the device. If the device on the bus is sending data back to the processor, a line called DD (data direction) must be driven low. If you have the lower 32K or RAM on another board, you can accomplish this by tying A15' and R/W to the inputs of a NAND gate (7400), and the output to DD. With only the first 16K on another board, you must also tie A14 to an input, requiring at least three inputs, as on a 7410.

One last problem that I ran into in substituting RAM for the Basic ROMs: with 40K of RAM installed, the Basic memory test runs off the end of the RAM at $9FFF and starts "testing" Basic, destroying Basic in the process. This continues until the memory test tests itself, which destroys it and sends the whole system off into the never-never land so familiar to those of us who program in machine language. To prevent that, I added another feature: a write-protect switch for a portion of the RAM on the 610 board. To accomplish this, I first found the R/W line on the underside of the 610 board where it passes between banks near U31. It ultimately goes to pin 10 of all the 2114's on the 610 board. I cut the line, separating the front bank from the middle and rear banks. This leaves the front bank working normally. Then, I added a switch to tie the rear sections either to the front section or to +5. With the switch set to tie to the front bank, everything works normally. With it set to tie to +5, the rear and middle banks become "ROM" (literally). This way, I could set the switch to "normal" and address the RAM at $8000 thru $9FFF to copy the Basic ROMs to RAM and make any desired changes. I would then switch to the write-protected switch position, disable Basic, and move the RAMs to $A000 thru $BFFF. This leaves me with a "RAM Basic", which I can modify any time I want by switching to the unprotected position, making changes, and then switching back to the protected position. Note that you must leave the power on during the whole operation; you won't hurt anything as long as you're careful not to short anything other than the lines involved in the change. The whole process is not nearly as involved as describing it is; it takes about ten seconds to switch over after copying the ROMs to RAM.

The new 6116 CMOS RAMs were my choice to fill in the areas at $C800 thru $CFFF and $Exxx. A 74138 handles all the address decoding to place three of them in those areas, tying OE' low and CS' to the appropriate line like this:

LISTING on page 6

```
                     74LS138
                   ---------
                  |         Y0|O------
PHASE 2' --------O|G2B'     Y1|O------ CHXX'
A15 --------------|G1       Y2|O------
A14' ------------O|G2A'     Y3|O------
A13 --------------|A        Y4|O------ ELXX'
A12 --------------|B        Y5|O------ EHXX'
A12 --------------|C        Y6|O------
                  |         Y7|O------
                   ---------
```

If you add these on an external board, you will have to drive DD as discussed above. One easy way to accomplish this is:

```
               ------
CHXX' -----|        \            -----
ELXX' -----| 7410  |O-----------|      \
EHXX' -----|        /           | 7400 |O------ DD
               ------    R/W -----|      /
                                 ------
```

Whether your interests lie in the hardware, the software, or a mixture of both, you need to understand the architecture of your system, especially with a processor such as the 6502 which relies so heavily on memory mapped I/O. By fully understanding the memory map, you can explore the potential of your system effectively in both hardware and software. I hope I have provided enough material to spark your imagination and assist you in your next project. If you write me with specific questions, please enclose a stamped, self-addressed envelope and I'll try to answer them as well as I can.

★ ★ ★

HEX LOADER

By: Weyburn Wakeford
2645 Dolly Brook Lane
Birmingham, AL 35243

When using a machine language routine with a BASIC program, it is a common practice to load each byte of the routine with a POKE statement. These bytes are written in BASIC using DATA statements with integers from 0 through 255.

The problem is that machine language or object code is most conveniently expressed in hexadecimal notation. The task of converting hexadecimal object code to DATA statements is a significant burden for the programmer. It also creates a proofreading problem.

The following example shows a superior approach. Just leave the DATA in hexadecimal. Your computer does the conversion!

```
05 REM HEXLDR
10 NR=7:AT=24064
15 FORI=0TONR-1:READ(H$)
20 :FORJ=1TO2:A=ASC(H$)
25 : IFA>57THENA=A-7
30 : P(J)=A-48:H$=RIGHT$(H$,1)
35 :NEXTJ:BYTE=P(1)*16+P(2)
40 :POKEAT+I,BYTE
45 NEXTI
50 DATA A9,24,8D,30
55 DATA D4,EA,60
60 REM SET USR(X) POINTERS
65 FORI=0TO255
70 IFAT<256THENLO=AT:HI=I:I=
   255
75 AT=AT-256:NEXTI
80 POKE8955,LO:POKE8956,HI
85 REM YOUR MAIN PGM HERE
90 X=USR(X)
95 REM END OF YOUR MAIN PGM
```

The programmer sets "nr" to the number of bytes to be loaded and sets "at" to the decimal value of his routine's origin.

Of course, your routine would appear in the DATA statements, using object code directly from your assembly listing. This example routine displays a dollar sign at video mapped memory location 54320.

Lines 25 and 30 assume that hex digits range between 48 and 57 and valid hex characters range between 65 and 70 in the ASCII 7-Bit Code Table. Edit checks for H$ or A were omitted since the programmer must proofread his DATA anyway.

Line 80 should be changed to fit your system's convention for linkage to USR(X) routines.

Line 25 can also be written as: 25 A=A+7*A>57 or combined with line 30 in the form A-48+7*A>57 with the A>57 having a -1 value when TRUE and a 0 value when FALSE. In this way, you can compress HEXLDR into 4 lines of BASIC code, plus your DATA statements. Put at the end of your BASIC logic and executed with a GOSUB, it is a modest overhead for such a productivity advantage.

★

HOW TO USE THE REPORT TITLE TO YOUR BEST ADVANTAGE

by: F. S. Schaeffer
84-55 Daniels St., #4F
Jamaica, NY 11435

I have one job which demands a highly customized printout. Yes, even with OS-DMS (which I've come to love despite my nasty notes about it in the past) this can be done but you have to put your "thinking cap" on! The report title in this specific case sets up an inclusive block of numbers and a location relative to those numbers (we're dealing with birdbands that were used at a specific location). Besides being a "midnight hacker", I also band birds for the fish and wildlife service in my spare time.

Suppose I used a location to which (in the program) I've referred to as location "A" and suppose the bands to be reported are from 1234-56789 thru 1234-56790, then, in the report title in stat 03, I enter this data like this:

A123456789 thru 56790

The variable assigned by stat 03 = H$.

Now we get to stat 3A where H$ is programmed to do several chores.

A) First we need the title (of the report) to read: 1234-56789 thru 56790 and this is accomplished by calling it HC$ where HC$=MID$(H$,2,4)+"-"+MID$(H$,6,26)

B) We also need to set the location in the header, so we do: LOC$=LEFT$(H$,1) AND THEN: IF LOC $="A" THEN LOC$=" WHATEVER IT IS".

C) We need to place the prefix (1234) in a specific spot in the report header by means of: LEFT$(HC$,4)+"<- - - - -"

D) We need to space to the row where the first number starts if it is not 01 (or 51) because 50 numbers are reported

NOTE:

In a recent letter from Roger Clegg of Data Products Maintenance Corp. in El Monte, CA, Roger gave us a fairly detailed list of Useful Memory located in OS-65U, and also a modified version of the Directory Program wich we would like to share with you.

```
:                   USEFUL MEMORY LOCATIONS IN OS-65U
:                   ---------------------------------
:
: 21         NULL count (usually 0)
: 22         POS(X) counter
: 27-97      71-character input buffer
: 120,121    Address of start of Basic program
: 122,123    Address of start of variable table
: 124,125    Address of start of array tables
: 126,127    Address of bottom of string space
: 130,131    Address of highest unused byte of string space
: 132,133    Memory size (First byte not available to Basic)
: 1390       Line delete character (usually @), if EDITOR not
             on line
: 1394       Rubout character (usually _), if EDITOR not on line
: 1398       Maximum length of input string (usually 71, maximum)
: 1797       Poke 44 to remove line numbers from listing, 32 to
             restore
: 2073       Poke 96 to kill Control-C, 76 to restore
: 2676       Poke 0 to kill carriage returns (usually 13)
: 2683       Poke 0 to kill line feeds (usually 10)
: 2720       Width of Basic PRINT fields using commas (usually 14)
: 2797       Input prompt character (usually 63 = ASC(?))
: 2888       Poke 0 to enable null input
: 2972       Poke 13 to allow ":" in inputs (usually 58 = ASC(:))
: 2976       Poke 13 to allow "," in inputs (usually 44 = ASC(,))
: 3015       Poke 47 to input D/M/Y as three numbers (usually 44)
: 8495-6     OS-65U Version Number = PEEK(8495)+PEEK(8496)/100
: 8620-1     Version Date (M/Y)
: 8704       Start of Basic dispatch address table
: 8738-9     Address of NULL routine -1 (for replacement by
             SWAP etc.)
: 8778-9     Address of USR(X) routine (usually points to
             "FC ERROR")
: 8960       Start of reserved word list
: 9025-8     "NULL". Replace by RSEQ, SWAP, KILL, PNTR, etc.
: 9057-60    "LIST". POKE 9057,1 to prevent listing.
             (65U uses 9058)
: 9712       Field width of PRINT $R,X  (usually 12)
: 9832       Current disk drive. 0=A,1=B,2=C,3=D,128=E.
             See TI 1006.
: 9889-97    Disk I/O Control Block. See TI 1017.
: 9906-69    Eight file control blocks. See TI 1000 & 1002.
: 9970       Start of 256-byte disk directory buffer
: 10226      Disk error number
: 10287      Lowest character printable to files (usually 13)
: 11193-5    To disable password checking: POKE with 169, 0,
             and 96
: 11657-8    Memory input pointer (device #4. See TI 1013)
: 11661-2    Memory output pointer
: 11664-5    Console I/O device numbers (serial console = 1,
             video = 2)
: 11666-7    Indirect file pointer. See Basic Manual p. 32
: 11668      Lowest "on" bit gives default INPUT device
             (console = 1)
: 11686      Each "on" bit gives default PRINT device (console = 1,
:            printer = 16 (bit #5), console + printer = 17, etc.)
: 11774-5    Line number of error = PEEK(11774)+256*PEEK(11775)
: 12019      51 at 1 Mhz, 102 at 2 Mhz
: 12098      Padding character used by INP$ (usually 32 = space)
: 13314-5    Hard disk cylinder number
: 14358      Lines on page not yet printed (for teletype,
             device #1)
: 14387      Lines per page, device #5 (usually 66)
: 14394      Spooling indicator. 0 = spooling off.
: 14457      Lines per page to be printed, device #5 (usually 60)
:            Poke 66 (or = PEEK(14387)) to kill automatic paging.
: 14646      Poke 91 to move program to indirect file. |
             ( See Basic
: 14721      Poke 24 to get program from indirect file.|
             Manual p. 32)
: 15006      Control-C flag: 0 when control-C not entered
: 15100      Lines per page to be printed (for teletype, device #1)
: 15141      Lines per page (for teletype, device #1)
```

continued

```
: 15886      Delay time before "PRINTER STALLED" message
             (usually 12)
: 15896      Poke 0 to kill "PRINTER STALLED" (waits forever)
: 15908      Lines on page not yet printed (device #5)
: 16317      OS-65U level.  1 = single user, 3 = time sharing,etc.
: 18959      Transient enabled:  1=EDITOR, 2=RSEQ, 3=INP$,
              4=COMKIL
: 19632      Number of seconds to WAIT FOR.  60 or more
              waits forever.
: 19633      Contains 0 if WAIT FOR was unsuccessful.
: 19968      Start of 3584-byte floppy disk buffer
: 23552      Start of RSEQ code if enabled,  otherwise free space.
: 23696      Start of EDITOR code if EDITOR or INP$ enabled.
: 23700      EDITOR's character delete character (usually 95 =
             ASC(_))
: 23701      EDITOR's line delete character (usually 64 = ASC(@))
: 23702      EDITOR's forward space
: 23703      EDITOR's backspace (usually 8)
: 23734-40   EDITOR's forward space echo to terminal
: 23741-47   EDITOR's backspace echo to terminal
: 24527      24527-24564 is free space unless RSEQ is enabled.
: 24565      WP-3 flag.  If not 0, utility programs return to WP-3.
: 24569-71   Day, Month, Year,  in level 1
: 24572-3    Number of bytes of machine code before Basic program
: 24576      Start of workspace for Basic programs (usually 24K)
: 46591      Top of workspace when PATCHS is enabled
: 47871      Top of workspace when COMKIL is enabled
: 49151      Usual top of workspace
: 55381      User Number in Timesharing and Networking
: 55919-24   Second, Minute, Hour; Day, Month, Year, in level 3.
: 56425-30   Devices 3-8, level 3:  User number if locked, 127
             unlocked
: 57199      Network node number.  0=K, 1=L, ..., 15=Z
: 57272      Partition number (0-15) in networking
: 57368      Start of 3584-byte hard disk buffer
: 64513      Last key pressed.  Useful for input without INPUT.
```

★ ★ ★

The Directory Program listed below has the following added features.

1. If you answer "1P" or "5P" to the "Port?" question, it prints the passwords.

2. If you answer "1W" or "5W", it wipes all the passwords off the disk (except for DIREC*) and changes all the access rights to R/W.

3. It displays deleted files as "(DEL)", enabling you to recover them by (5).

4. At the end of each page it pauses. A "D" brings a question about which file to delete.

5. A "C" (for change) enables you to change any file's name, password (if the passwords are displayed), file type, and access rights.

6. Any other response gets the next page of the Directory.

```
1 REM *************************   D I R   ***********************
2 :
4 :
10 Q=256: TR=3584: CLR$=CHR$(27)+CHR$(28): U$=CHR$(27)+CHR$(12)
20 X=PEEK(9832): DV$=CHR$(65+X): IF X=128 THEN DV$="E"
30 PRINT CLR$: INPUT"Unit";D$: IF D$="" THEN D$=DV$
40 FLAG 9: DEV D$: CLOSE: OPEN"DIREC*","PASS",1
50 FLAG 10: PRINT: INPUT"Port";R$
60 D=VAL(R$): R$=RIGHT$(R$,1): IF R$="P" OR R$="W" THEN P=8
70 :
80 GOSUB 3000: DP=1: OF=16
90 :
100 POKE CB+2,DP+97
110 ER=USR(0): IF ER THEN PRINT"READ ERROR"ER: GOTO 6000
120 IF R$<>"W" THEN 160
130 FOR I=9976-32*(DP=1) TO 10216 STEP 16
140 POKE I,0: POKE I+1,0: POKE I+2, PEEK(I+2) AND 252 OR 3: NEXT
150 ER=USR(1): CLOSE: IF ER THEN PRINT"WRITE ERROR"ER: GOTO 6000
160 IF DP>1 AND PEEK(9970)=0 THEN 5000
```

continued page 13

LETTERS

ED:

I guess the ominous break key is on most commercial terminals but not on most keyboards that are associated with monitors. I accidentally pressed mine once and much to my surprise everything still worked afterwards. Well, maybe the break key was broken if it didn't break anything else. I had to investigate that possibility. Before I did however, I had to find out how the little dickens worked and this is what I've learned about it.

First, I'll discuss its effects by the RS232 signals. We all know that in a quiescent state, that is when no keys are pressed, that the RS232 signal line is at a negative level of at least -3v. When a key is pressed the signal voltage then goes to at least +3v. This first positive excursion is the start pulse. If a delete key has been pressed, 00 hex will be transmitted. Which means that the signal will remain positive for the duration of that character and then return to a negative voltage. A stop pulse has been transmitted also, but that wasn't obvious unless the repeat key was pressed simultaneously with the delete key. Then the voltage signal would appear positive most of the time and the negative stop signal would become apparent

The break key transmits much the same way, however, the negative stop signal is not there. That is, the voltage goes positive for the duration of the break key closure. The only difference therefore, between a delete key and a break key is the stop bit that creates the timing for each character sent. Professional equipment uses the break key much as other keys are used. OSI however, found it unnecessary to implement even though the ACIA has the capability.

Here is how the break key can be implemented. The programs that you may use within your own system may not need it, so implementing the break key function would be important to only those that need to communicate break data to other equipment. That means that you'll probably be using more than one port on the terminal based systems. On the keyboard/monitor systems a break key will have to be installed, in which case it would probably be as easy to simultaneously install electronic logic to directly force the break output signal. I will describe how to implement the break function on a secondary port.

The 6850 ACIA will, under normal circumstances, not pass the break key data but then the break key data is not normal data. It lacks the timing characteristics of the other keys. When the serial input to the ACIA is exposed to a break key closure the input register and its buffer are overrun. And framing and overrun error flags are set in the status register. When the break key is released the timing may be such that the register isn't completely filled for that particular character. Then the overrun flag may not be set but the framing error would still exist. Since the break data is to be passed out of the secondary port the routine that handles the output of the secondary port must be altered. This, of course, can be dealt with in a large variety of ways. The way I chose was: when attempting to output a character through port two I would sense bit 4 or (10 hex) of the status register of port one. If there was a framing error in port one I would save the pending character and exit to a break routine. The break routine need not reset the ACIA of port two. Loading bits 5 and 6 into control register of port two will raise the serial output (RS232 again) to a positive voltage. This will remain this way until the ACIA is reset or a new value is set into the control register of the ACIA. Again the ACIA need not be reset. The ACIA will assume the setting that is written to the control register. My routine continues to sense port one to maintain the setting of port two as long as the break key is set. It is important to remember at this point that just reading the status register of port one does not reset flags and therefore, would not indicate a break key release. The data register must also be read to reset the status flags. Once key release is sensed then the original value that port two ACIA was initialized with, must be set back into its control register. Output of port two data register will then return to a negative voltage and its normal operation. Here then in sequential steps is the way my algorithm goes:

1) In port two output routine sense port one for break key closure.

2) If framing error is sensed then JSR to break routine.

3) Set port one to a break setting.

4) READ port one data register to reset status flags.

5) READ port one status register; loop if framing error.

6) Write initialization value of port two into port two control register.

7) READ port two data register to reset its status flags.

8) Return to port two output routine.

It is not wise to implement this function in reverse; that is to sense for a break key from port two. If you transfer data such as machine code there is a chance of losing data which could be catastrophic unless you have a routine to handle that. If you're trying to implement this in OS65U or OS65D, good luck... my blessing and sympathies go with you. However, it's really easy on CP/M to change the BIOS. And your break key will function with all your software.

Arthur Goeres
Portland, OR 97220

* * * * *

ED:

I would like to know why I cannot print graphics characters on my monitor using the PRINT CHR$(X) command. If I run a loop of: FOR X=0 TO 255, the computer only prints those characters that are on the keyboard (i.e. 0-9, A-Z, a-z, & shifted keys). <u>All</u> characters can be POKEd, however. Please let me know if there is a solution.

Paul W. Elder,
Nutley, NJ 07110

PAUL:

The print routine in Basic masks the print CHR$ with 128. This routine gets rid of the 7th bit so that you have the 64 character ASCII set of printable characters.

Brian

* * * * *

ED:

The literature available for OS65U V1.3 (and I assume the same to be true for V1.4x--I have not yet made a move to V1.4) contains the following code which may be used as a model for a CRT cursor positioning subroutine (see reference manual discussions on terminal independence):

```
100 POKE 22,X:ON AR GOTO
    101,102,103,103
101 PRINT AD$;CHR$(X+XF);
    DL$;CHR$(Y+YF);DE$;:RETURN
102 PRINT AD$;CHR$(Y+YF);
    DL$;CHR$(X+XF);DE$;:RETURN
etc.
```

I will not go into a discussion of the variables as they are adequately defined in the reference manual.

I have been using a similar subroutine in my programs to permit flexibility in choice of terminals (for backup and others using my programs). There is one small change I have made to this subroutine which may be of interest to PEEK(65) readers:

```
100 ON AR GOTO 101,102,103,
    103
101 PRINT AD$;CHR$(X+XF);
    DL$;CHR$(Y+YF);DE$;:
    POKE22,X:RETURN
102 PRINT AD$;CHR$(Y+YF);
    DL$;CHR$(X+XF);DE$;:
    POKE22,X:RETURN
etc.
```

Memory location 22 is used to contain a count of the number of "printable" (i.e., non-control) characters output by a PRINT command. If a PRINT to the CRT specifies that no carriage return is to be performed after the message is displayed (e.g., PRINT MSG$;), the contents of location 22 will be the sum of what was in location 22 when the PRINT was issued plus the count of the number of non-control characters encountered during the output sequence. The contents of location 22 can then be used to determine where the cursor is on the X-axis following a PRINT statement. (The value in location 22 will continue to increment as PRINT commands are issued to the CRT until a carriage return is issued, location 22 is POKEd with a value or a count of 255 is reached, in which case the count rolls over.) For example:

```
500 Y=3 : X=10 : GOSUB 100 :
    REM POSITION CURSOR
520 PRINT MSG$; : REM DISPLAY
    MESSAGE
540 X=POS(X) : REM GET
    CURRENT X-COORDINATE
```

Note: POS(X) is essentially a PEEK(22).

This can be particularly useful in screen formatting routines, etc. when messages to be displayed are of variable length (or you may change a message sometime) and it is necessary to retain the X-coordinate for use in some subsequent program process such as INPUT from the operator. Location 22 must be interrogated before any PRINT commands to any other device (line printer, disk, etc.) as the character count will be reset when PRINT#5, PRINT%1, etc. is issued.

The reason for the change I made to the cursor positioning subroutine is that the character count maintained in location 22 may be greater than the number of characters being PRINTed. The character count is incremented each time a non-control character (ASCII value greater than 31) is encountered in the data being sent to the terminal. If the control sequence used to address the cursor contains such characters, the count will be incremented.

To illustrate this, the con-

continued on page 14

```
170 IF DP>1 AND D>1 THEN 300
180 :
200 PRINT#D,CLR$ TAB(15+P/2)"OS-65U FILE DIRECTORY FOR DEVICE"D$;
210 IF D<2 THEN PRINT#D,TAB(58+P)"PAGE"DP"OF"TDP;
220 PRINT#D: PRINT#D
230 PRINT#D,"    NAME      ";: IF P THEN PRINT#D,"PASSWORD";
240 PRINT#D,"    TYPE    ACCESS    ADDRESS     LENGTH     SEC BND
    SEC LEN"
250 GOSUB 4000: N=0
260 :
300 IX=9970+OF: IF PEEK(IX)=0 THEN GOSUB 4000: GOTO 1000
310 :
400 N=N+1: EC=EC+1: N$="": FOR I=0 TO 5: N$=N$+CHR$(PEEK(IX+I)):
    NEXT
410 :
500 PW$="": FOR I=1 TO 2: L=PEEK(IX+5+I): PW$=PW$+CHR$(65+INT
    (L/16))
510 M=L AND 15: M=M+28*(M=15): PW$=PW$+CHR$(78+M): NEXT
520 IF PW$="PAPS" THEN PW$="PASS"
530 IF PW$="ANAN" THEN PW$=" ."
540 :
600 TM=PEEK(IX+8) AND 12: TY$="OTHER": IF TM=0 THEN TY$="DATA"
610 IF TM=4 THEN TY$="BASIC"
620 :
630 TM=PEEK(IX+8) AND 3: AR$="NONE": IF TM=1 THEN AR$="READ"
640 IF TM=2 THEN AR$="WRITE"
650 IF TM=3 THEN AR$="R/W": PW$=""
660 :
700 DA=Q*(PEEK(IX+09)+Q*(PEEK(IX+10)+Q*PEEK(IX+11)))
710 SZ=Q*(PEEK(IX+12)+Q*(PEEK(IX+13)+Q*PEEK(IX+14)))
720 :
730 IF PEEK(IX)=1 THEN RE=RE+SZ: N$=" (DEL)": PW$="": TY$="":
    AR$=""
740 :
800 PRINT#D,MID$(STR$(N),2) TAB(4)N$;: IF P THEN PRINT#D,
    TAB(16)PW$;
820 PRINT#D,TAB(16+P)TY$ TAB(24+P)AR$ TAB(32+P)DA;TAB(43+P)SZ;
830 SB$="NO": IF DA/TR=INT(DA/TR) THEN SB$="YES"
840 SL$="NO": IF SZ/TR=INT(SZ/TR) THEN SL$="YES"
850 PRINT#D,TAB(55+P)SB$ TAB(64+P)SL$
860 :
900 IF DA+SZ>HA THEN HA=DA+SZ
910 OF=OF+16: IF OF<Q THEN 300
920 :
1000 IF D>1 THEN 2000
1010 PRINT: INPUT R1$: IF R1$<>"D" THEN 1100
1020 INPUT"Delete file #";FL: IF FL<1 OR FL>N THEN 2000
1030 POKE 9970+16*(FL-1-(DP=1)),1: GOTO 1600
1040 :
1100 IF R1$<>"C" THEN 2000
1110 PRINT U$;U$;: IF PEEK(IX) THEN GOSUB 4000
1120 PRINT"    1";: IF P THEN PRINTTAB(16)"2";
1130 PRINTTAB(15+P)2+P/8 TAB(23+P)3+P/8
1140 PRINT"Enter first the file number, then the column number,
     ";
1150 PRINT"then the correction": INPUT FL: IF FL<1 OR FL>N
     THEN 2000
1160 PRINTTAB(10)U$;: INPUT CO: PRINTTAB(20)U$;: INPUT CO$
1170 :
1200 IF CO<>1 THEN 1300
1210 CO$=LEFT$(CO$+"      ",6)
1220 FOR I=1 TO 6: POKE 9969+16*(FL-1-(DP=1))+I,ASC(MID$
     (CO$,I,1)): NEXT
1230 :
1300 IF CO<>2 OR P=0 THEN 1400
1310 IF CO$="." THEN CO$="ANAN"
1320 CO$=LEFT$(CO$+"    ",4)
1330 FOR I=1 TO 2: L=ASC(MID$(CO$,I*2-1,1)): IF L<65 OR L>80
     THEN L=80
1340 M=ASC(MID$(CO$,I*2,1)): IF M<78 OR M>93 THEN M=93
1350 POKE 9975+16*(FL-1-(DP=1))+I, 16*(L-65)+M-78: NEXT I
1360 :
1400 T=9978+16*(FL-1-(DP=1)): IF CO<>2+P/8 THEN 1500
1410 TY=2: IF CO$="DATA" THEN TY=0
1420 IF CO$="BASIC" THEN TY=1
1430 POKE T, PEEK(T) AND 243 OR TY*4
1440 :
1500 IF CO<>3+P/8 THEN 1600
1510 TY=PEEK(T) AND 3: IF CO$="NONE" THEN TY=0
1520 IF CO$="READ" THEN TY=1
1530 IF CO$="WRITE"THEN TY=2
1540 IF CO$="R/W"  THEN TY=3
```

continued

```
1550 POKE T, PEEK(T) AND 252 OR TY: IF TY=3 THEN CO=0: CO$=".": GOTO 1310
1560 :
1600 ER=USR(1): CLOSE: IF ER THEN PRINT"WRITE ERROR"ER: GOTO 6000
1610 CH=-1: OF=-16*(DP=1)
1620 GOTO 200
1630 :
2000 IF EA>=EN OR PEEK(IX)=0 THEN 5000
2010 EA=EA+Q: OF=0: DP=DP+1: GOTO 100
2020 :
3000 POKE 8778,192: POKE 8779,36
3010 POKE 9432,243: POKE 9433,40: POKE 9435,232: POKE 9436,40
3020 CB=9889: FOR I=1 TO 5: POKE CB+I,0: NEXT
3030 POKE CB+6,1: POKE CB+7,242: POKE CB+8,38
3040 TDP=PEEK(9902): EN=25088+Q*TDP: HA=EN: RETURN
3050 :
4000 FOR I=1 TO 70+P: PRINT#D,"-";: NEXT: PRINT#D: RETURN
4010 :
5000 PRINT#D: IF D$<"E" THEN PRINT#D,275968-HA"BYTES FREE"
5010 IF CH THEN 6000
5020 IF RE THEN PRINT#D,RE"BYTES RECOVERABLE"
5030 PRINT#D,EC"FILES DEFINED OF"TDP*16-1"POSSIBLE"
5040 :
6000 POKE 8778,208: POKE 8779,16: DEV DV$: END
6010 :
50000 GOTO 50
```

continued from page 12

trol sequence my terminal expects to receive to signal cursor addressing always contains three (3) alphanumeric characters. If the X-coordinate (X) is POKEd into location 22 immediately upon entry to the positioning subroutine, then location 22 will have a value of X+3 following the execution of the PRINT to position the cursor. When the message is PRINTed and X=POS(X) is issued, the program will receive a value indicating the cursor is three positions farther along the X-axis than it really is. If the X-coordinate is POKEd into location 22 following cursor positioning, then location 22 will have the true X-coordinate upon exit from the subroutine.

There are times when I want to do forward space and/or backspace cursor in a program. The codes for these commands are placed in memory when either the EDITOR or Extended Input process the CRT parameter file "CRT 0". Both the GETCRT program and the reference manual contain a model program which can be used to pick up CRT control codes from the operating system. The following code can be added to the routine to make forward/back space cursor control codes available to the application program (line numbers given follow the sequence in the reference manual):

```
63940 Z=23734 : FS$="" : REM
      FORWARD SPACE CURSOR
63945 Z1=PEEK(Z):Z=Z+1:IFZ1<>0
      THENFS$=FS$+CHR$(Z1):
      GOTO63945
63950 Z=23741 : BS$="" : REM
      BACKSPACE CURSOR
63955 Z1=PEEK(Z):Z=Z+1:IFZ1<>0
      THENBS$=BS$+CHR$(Z1):
      GOTO63955
```

David Weigle
Morton, IL 61550

* * * * *

ED:

Perhaps the following information will help Gary Levine and Frank Nelson with their problem in sending a 'break' signal to a host system.

In order to recognize a 'break' condition the host systems that I have occasion to dial up, require that the transmitting modem send a space level for not less than 250 msec. or more than 450 msec. The following routine will accomplish this for my C1P.

First the control register on the 6850 ACIA is set such that a 'break' level is transmitted. Next the X register is loaded with 255 and the subroutine at FC91 is executed. This routine causes a delay for X msec. It is part of the disk boot. Finally the ACIA is reset and reprogrammed for the configuration required by the host system. In my C1P system I store the ACIA configuration in 00F8.

Most systems including CompuServe, the Dow Jones News Service, and local bulletin board services require a configuration code of HEX 09 (7 bits, even parity, one stop bit). The C1P's normal default is HEX 17 (8 bits, no parity, two stop bits).

As you can see from the following listing, the code occupies part of a custom support ROM I developed for the C1P. This ROM includes code for a semi-smart terminal emulator, a screen editor, corrected keyboard, BASIC keyword shorthand entry, and a routine to dump machine code to tape in a form reloadable by the OSI monitor.

```
FBEB A9 60     :64491 LDA #96
FBED 8D 00 F0  :64493 STA 61440
FBF0 A2 FF     :64496 LDX #255
FBF2 20 91 FC  :64498 JSR 64657
FBF5 A9 03     :64501 LDA #3
FBF7 8D 00 F0  :64503 STA 61440
FBFA A5 F8     :64506 LDA 248
FBFC 8D 00 F0  :64508 STA 61440
FBFF 60        :64511 RTS
```

Jim Hays
Seattle, WA 98116

* * * * *

ED:

In reference to Mr. Guy Vanderwaeren's Project #1, you stated only 4K was left to use. Mittendorf Engineering's High Resolution Graphics Kit (unfortunately, no longer manufactured) when added to the 600/610 boards, adds the final 8K of RAM at address $8000 to $9FFF to allow a total of 40K useable RAM. Therefore, Mr. Vanderwaeren's project is possible.

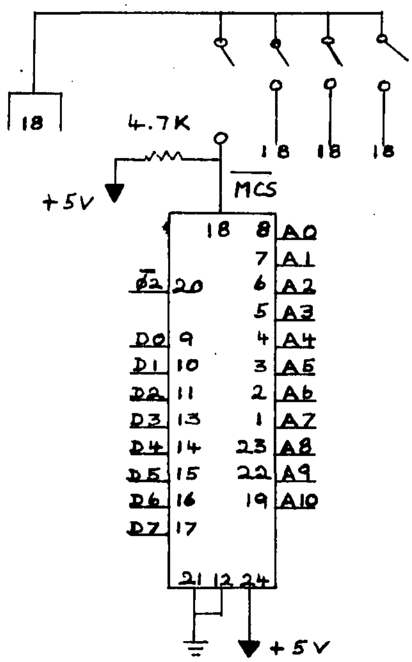
My question concerns a switchable Monitor ROM card for the 600 board. I am using Micro-Interface's ROM-Term monitor ROM (a fine, versatile product), but occasionally would like to use other monitor ROMs without having to take everything apart to change ROMs. I ordered a Gemini ROM card from Orion Software, but was told it was no longer in production due to low demand. It was advertised as a 4 socket card, using #1 socket for the original OSI Monitor ROM with 3 other sockets available for 2716 EPROMs. The card plugged into the 600 board's monitor ROM socket, tapping off 5V from the 600 board or power supply for the EPROMs. Are you aware of any similiar ROM card on the market, or what would building one entail? Would you run all socket pins in parallel, except for the 5V pins which would be hooked up to a switch?

Harold B. 'Bud' Boyd
St. Catharines, Ontario

Bud:

It is said that all things are possible, more so with a computer. We try to answer questions in the context of the letter and at a particular skill level. One of the things we try to do is to keep our answers simple, to the point and what is currently available and without re-designing the computer in question. Sometimes this leaves us with egg on face, because someone will write in and say that we are wrong (we knew it could and probably has been done) and that they have done it. Well, if they have, why don't they share their knowledge through PEEK.

As to your question. It has a very simple solution. See drawing:



Wire up five 24 pin sockets in parallel as in drawing. Wire pin 8 to 8, Pin 7 to 7 and so on. Do this for all pins except Pin 18. Wire pin 18 of four sockets to a 4.7K ohm 1/4 watt resistor. Wire the other end of the resistor to +5 volts. Wire pin 18 to one side of a normally open switch. The other side of the normally open switches should be tied together. There are only four switches. Pin 18 of the fifth socket should be wired to the common side of the normally open switches. Remove monitor prom from its socket on the 600 board and insert it into one of the four sockets just wired. Plug in a 24 pin ribbon jumper cable between the old monitor prom socket and the fifth socket on the new board. By closing the switch you select that prom to use as a monitor. The prom selection technique is crude but should work. Sockets are wired to accept a 2716 EPROM or PIN/PLUG equiv.

Brian

* * * * *

ED:

I am wondering if you know of any concern or individual who has done any work on an Apple emulator or making OSI Apple compatible?

Charles F. Merica N4IF
Covington, VA 24426

* * * * *

ED:

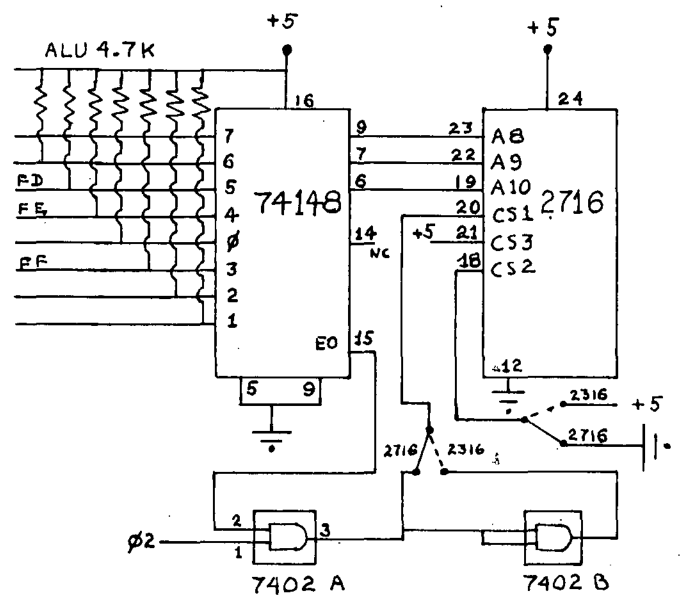
A quick note regarding Roger Miller's article (replacing 3 - 1702 proms with a 2716 EPROM in an OSI C2-4) from the January '83 issue.

I've enclosed a schematic of an adapter I've been using with my 500 board. It requires no cutting of the foil and is the same circuit used on the 502 board, all data, address, FD chip select and +5v lines are picked up from the FDXX socket. 02 is taken from pin 14 on the foil side of the ACIA socket, and FE & FF chip selects are picked up from unused pads for SW-1 (just above the FDXX socket.)

I've been using this adapter for some time now with the monitor written by Mr. A. Penaloza from the August '81 and November '81 issues of PEEK (65).

Elliot Spiro
Wantagh, NY 11793

* * * * *



ED:

Here are answers or corrections to answers to several questions in the January 1983 PEEK.

Your answer to Tim Lowe about his difficulty with CompuServe was unnecessarily complicated, even if your analysis was correct. I believe from his description that the problem occurs because the OSI serial interface is automatically initialized for 8 bits and 2 stop bits upon startup. This causes the interface to miss the start bit of the second character received, convert the remainder of the second and third characters as garbage, and get re-synchronized on the fourth character, starting the process over again. The solution, regardless of which analysis is correct, is to set the interface as specified by CompuServe. I would suggest using the 7 bit - with parity mode to avoid receiving some letters as graphics characters. If the Modem program in use (I'm not familiar with it) does not initialize the interface, the solution is very simple. Before running the modem program, POKE 61440, 9 (7 bits, even parity) or POKE 61440, 21 (8 bits, no parity).

If the modem program does its own initialization (it should not), you must find that initialization and change it. If the program is written in Basic, it will look like:

```
POKE 61440, 3 : POKE 61440,17
```

If so, change the 17 to 9. If it is in machine language, it will look like:

```
A9 03       LDA #3
8D 00 F0    STA $F000
A9 11       LDA #$11
8D 00 F0    STA $F000
```

In this case, change the $11 to $09. I would first try assuming that it does not initialize the modem, and simply initialize it with the POKE I mentioned above and try it.

In answer to Roger Miller's problem with the video display, I believe he needs to adjust the potentiometer which controls the balance of video and sync in the composite video output of his system. If he has a C1P (or Superboard) this is R58, located on the extreme left edge of the circuit board near the back (as viewed from the keyboard).

WEST COAST DISTRIBUTOR

Ohio Scientific LIQUIDATION UP to 60% off

Model	Description	Retail	Cash
	Showroom Demonstrators		
C4PMF	24k w/5" floppy & color	1995	1299
C4PDMF	48k w/dual 5" floppy	2495	1499
CD-3AP	5" disk drv w/pwr sup/cabl	699	350
C8PDF	48k dual 8" floppy/color	3495	1750
C2-OEM	48k 1mhz dual 8" 65U sys	3250	1600
C2-OEM2	48k 2mhz stat mem 65U	3750	1800
C3-OEM	48k 2mhz dual 8" 3 proc	4200	2100
C3-OEM	56k 2/4mhz CP/M compatbl	4400	2200
C3-DTS	56k 10 Mbyte 5 ser ports	9300	4650
C3-D/5	52k 5 Mbyte HD 2mhz	8000	4000
C3-C12	104k 36 MB 2 user 2/1mhz	14900	8950
C3-B33	152k 74 MB 3 user 2mhz	19500	9990
	Brand New Computers		
C3-OEM	56k 2mhz dual 8" 65U 3 proc	4400	2950
C3-DTS	56k 10 M/B HD up to 4 user	9300	5750
C3-C′	52k 36 MB HD 16 slot	13500	8500
C3-D/5	52k 5 Mbyte HD 2mhz - slick	7000	5000
	Accessories and Spare Parts		
CM-2	4k stat ram at D000 L3/CPM	125	49
CM-10	8k stat mem at D000 L3/CPM	200	79
CM-6	48k 1mhz dynam mem board	550	249
CM-9	24k 2mhz stat mem board	450	179
CM-3	16k 2mhz lo pwr stat mem	399	149
CM-11	48k 2mhz stat lo pwr ram	995	499
CA-′	Centr prl ptr intfc w/cabl	235	99
CA-[illegible]	Diablo prl intfc w/ribcabl	200	99
CA-10-1	RS-232 ser intfc w/1st port	200	99
CA-15	Modem/Telephone intfc board	500	299
FD100-8	Siemens 8" disk driv A or B	500	299
PS5-3	5 volt 3 amp power supply	69	29
PS24	24 volt 2.5 amp disk pwr sup	110	49
PS-1	5/12/-9 volt triple pwr supp	270	129
590/5	Hard disk controller pair	600	399
510c	CPU w/3 proc 2/4 mhz	600	299
470b	Floppy disk controller board	175	79

Discounts for quantity purchase/dealers/clubs/schools
Offer limited to qty on hand. Payment by cashiers ck
Demos tested and sold as is. Inspection available.

Toll Free 1-800-854-7165 Call today

SPACE-COM International

22991 La Cadena Drive, Laguna Hills, CA 92653 (714) 951-4648

Paul Chidley asked about running Basic programs under ROM Basic with a real operating system. I ran into this problem myself about three years ago and HEXDOS was the result. It is a full operating system, unlike PICO-DOS, uses ROM Basic so that it is very compact, and can be made to run OS65D programs with some minor adaptations. I sell it, with a 40-page manual and a selection of sample programs, for $49.50. I will mail a "quick reference" card, which gives a feel for some of its features, to anyone who sends a self-addressed, stamped envelope. Readers may also want to see the review which appeared in the February 1982 PEEK.

Ray Audette was having problems caused by the automatic line-feed generated by the C1P with every carriage return (and I KNOW he isn't the first - this problem is common with every combination of computer and printer I've used - to LF or not to LF, that is the question...). The Lineprinter VII should have a switch setting to select whether or not it does a line-feed after every carriage return; check the owner's manual. If not a short program like this will set up the serial port so that it will ignore linefeeds:

```
10 DATA 32, 45, 191   :
   REM  JSR $BF2D
20 DATA 72            :
   REM  PHA
30 DATA 173, 5, 2     :
   REM  LDA $0205
40 DATA 208, 2        :
   REM  BNE *+4
44 DATA 104           :
   REM  PLA
45 DATA 96            :
   REM  RTS
50 DATA 104           :
   REM  PLA
60 DATA 201, 10       :
   REM  CMP #$0A ; LINEFEED
70 DATA 240, 250      :
   REM  BEQ *-4
80 DATA 76,  115, 255:
   REM  JMP $FF73
90 T=PEEK(133)+PEEK(134)*
   256-19
100 POKE 133, 255 AND T:POKE
    134, T/256
110 FOR I=T TO T+18
120 READ T : POKE I,T
130 NEXT I
140 POKE 538, 255 AND T:POKE
    539, T/256
```

Steven P. Hendrix
New Branfels, TX 78130

* * * * *

ED:

Does anyone out there know anything about the High Resolution Graphics Board that is supposed to be available from OSI? Does it exist? What kind of software do they provide? Can they be persuaded into selling just the additional board needed along with the circuit diagrams and driver software so that hardware people can add the board themselves?

Does anyone know if PASCAL and FORTRAN will run on a C8P-DF with 48K of memory? Softec says I needed 64K minimum to create programs.

Can anyone out there do a review of these items? That seems to be one thing I miss since OSI stopped advertising and printing their Small Systems Journal.

Alex J. Kowalski, Jr.
South Bend, IN 46619

Alex:

Are you sure that you really want Hi Res? Although Hi Res machines are available, the mods required to upgrade are extensive (470, CPU, mem boards and restrapping), come with little or no documentation or software and are a large thorn in OSI's side. In short, they do not support the retrofits. If you do find one and get it to work, OSI would almost prefer that you enjoy it but keep it a secret. If others less proficient and persevering than you try, they will keep OSI's lines busy answering questions about an unsupported item. That's why it is unsupported.

A 48K machine running Pascal will have something like 9-11K of work space. There's nothing magic about 64K, its just more than 48K. A number of users have reported that "it doesn't work"! We hope that they are wrong.

Likewise Fortran, though available under CP/M and compiled to P-code, as in Pascal, is difficult to install and rather cumbersome to the end-user.

If you are still interested, after all this, try contacting D. B. Baker, editor of the Osmosus Group, 3128 Silver Lake Rd., Minneapolis, MN 55418. They seem to have had as much experience as anyone with Pascal & Fortran.

ETG

* * * * *

ED:

I believe I have a fix for Tim Lowe's modem problem (Jan '83). OSI's modem program is a basic program that pokes in a machine language program then calls this program using the X=USR(X) function. Make these changes to the basic program MODEM then save it back on your disk.

Change lines:

```
1500 FORI=0+FTO221+F:READX
2000 DATA 32,13,37,173,0,240,
     74,144,6,173,1,240,32,
     251,-1
```

Add line:

```
2140 DATA 169,127,76,67,35
```

Now, here's what is going on. The following code is a partial disassembly of the modem routine that is poked into memory starting at location $5222. This is the area needed to be modified.

```
$5222 JSR $2644  ;Swap 4 bytes
      for the keyboard
$5225 LDA $FC00  ;Check status
      of ACIA (modem)
$5228 LSR A      ;Is the data
      ready?
$5229 BCC $5231  ; N - Then go
      check the keyboard
$522B LDA $FC01  ; Y - Load
      the data into the A reg.
$522E JSR $2343  ;Jump to the
      OS output routine
$5231 JSR $525D  ;Is there a
      key down?
$5234 BEQ $5225  ; N - Then
      check the ACIA again
```

The change made to line 2000 will change the JSR instruction from location $2343 to $52FB. This is where the new code from line 2140 will be poked. This new code will modify the data received from the ACIA before the OS output routine is called. The new code is as follows:

```
$52FB AND  #$7F    ;Strip the
      8th bit from the data
$52FD JMP  $2343   ;Now call
      the OS output routine
```

The change to line 1500 allowed for the 5 extra bytes that were added to the end of the program.

Although I don't have a modem now, I plan to get one in the near future. Does anyone have plans for a good one? Anyway, I'm confident this is a good fix.

Jeff Kalis
Grand Rapids, MI 49506

* * * * *

ED:

Reply to Mr. Ray Audette of Canada about the Radio Shack line printer VII.

A line of BASIC starts with a code 10 for line feed, and ends with a code 13 for return. The LP VII interprets a 10 or a 13 as a line feed, and a 26 as a carriage return. I copied the CASS mini word processor from the June 1981 Aardvark Journal and modified it for the LP VII. Material to be printed is entered as strings and then printed with a line such as this;

```
100 PRINT A$(I);CHR$(26);CHR$
    (13);
```

The semi colons are required to prevent the LP VII from doing a line feed-carriage return. The 'CHR$(26)' causes a carriage return only. The 'CHR$(13)' causes a line feed only. Do not forget the final semi-colon.

I have not been able to modify anything to list a program on the LP VII with single line feeds. Any suggestions??

I have a superboard II series 2 with LP VII. If Mr. Audette will send me a cassette I will send him a copy of the program I use for the printer.

Jack Vaughn
Beaumont, TX 77707

* * * * *

ED:

I just thought I'd write in to express my amazement about Victory Software's 34 program deal for $29.95 advertised in your October issue. I was a little skeptical when I bought it, but boy was I surprised! These programs are really great. Some of the graphics they use are hard to believe I'm watching my OSI. These programs are advertised for the C1, but most of them work just fine on my C2. Some of the utilities and statistical programs are kind of ho-hum filler type junk, but the games are really fun to play. A lot of the graphic games are arcade copies, but most of the strategy games are original and really challenging! Not only that, but they are thoroughly documented. In summary, this is the kind of software package you always wish would be included with the purchase of your computer, but never is.

Stephanie Ondich
Farrell, PA 16121

* * * * *

ED:

I am currently using Steven Hendrix's HEXDOS disk operating system on a C1PMF. HEXDOS is all the operating system I could want and it only occupies 2K of free memory leaving me with 30K for programs and data. It comes with a number of utility and demonstration programs including disk formatting, file creation, and file deletion routines. One utility that is not supplied, however, is a program to create backup copies of disks. This facility is almost a necessity if you have valuable programs and data on your disks. The program shown meets this need. It will copy all the tracks listed in the directory of the disk in the active drive to a formatted disk in the inactive drive. Either the A or B drive may be active. The program leaves the previously inactive drive in active status upon termination. Obviously, the program requires a two drive system. I maintain a copy of this program on every disk so that I can easily create backup copies whenever required.

```
2 REM BACKUP COPY FOR HEXDOS
4 REM 10/31/82
6 PRINTCHR$(26):D=4096:
  M=D+2048:S=1
8 AD=(PEEK(49152)AND64)/64
10 IFAD=1THEND$="B":SD=0:
   CD=128
12 IFAD=0THEND$="A":SD=128:
   CD=0
14 PRINT "Load backup disk in"
16 PRINT"drive "D$" and press
   C"
18 PRINT"three times"
20 FORI=1TO3
22 IFUSR(0)<>67THENEND
24 NEXTI:PRINT:LOAD!:LOAD!
26 LOAD#(1+SD),D:E=D
28 IFPEEK(E+1)>0THENE=PEEK(E)+
   256*PEEK(E+1)+D-2817:
   GOTO28
30 E=PEEK(E-4)-1:IFE>39THEN
   PRINT"Directory error":END
32 FORI=STOE
34 LOAD#(I+SD),M
36 SAVE#(I+CD),M
38 NEXTI
40 PRINT:PRINT"Tracks"S"thru
   "E"copied"
```

Jim Hays
Seattle, WA 98116

* * * * *

ED:

First, a quick answer that might help Ray Audette and others who may have that extra line problem with their printers. In many of the printers there is the capability to reset the default configuration. Listed below is the Centronics Standard that was included with my printer:

(CTRL-I) XN (RETURN) WHERE X=NUMBER OF COLUMNS THE PRINTER WILL USE.
(CTRL-I) (RETURN) WILL TURN ON MONITOR WHILE THE PRINTER IS IN USE.
(CTRL-I) K (RETURN) TOGGLES THE AUTOMATIC LINE FEED OPTION. -->THIS SHOULD SOLVE HIS PROBLEM.
(CTRL-I) (CTRL-X) (RETURN) WHERE X IS ANY CONTROL CHARACTER TO REPLACE THE CTRL-I IF DESIRED.

Several years ago, I devoted the SB II to a portable unit, one that runs off of my car cigarette lighter along with a portable T.V. and a cassette recorder. In order to use the storage and other capabilities of an Apple II, both are hooked on-line. In this mode, I have found that most of my Apple software runs on the OSI with only a few mods made to correct the graphics. This comes about as both use a version of basic by microsoft. As of now, the programs have to be manually converted over for either computer if I have not written them with dual purpose usage in mind.

The project on-hand is naturally a program that will identify and change the few basic tokens that are a problem. I am willing to share the results, or better yet, work along with someone else who is running two computers as I am.

Paul Savard
McAlester, OK 74501

* * * * *

ED:

I have the answer to Ray Audette's letter that appeared in the January 1983 issue. That is because I had the same problem when I bought my Radio Shack LP VII printer about 18 months ago. It was after reading about certain programs changing the output vector and reading Ed Carlson's book, OSI BASIC IN ROM, that I came up with the solution.

Upon reading the disassembly of the support Rom, I noticed that OUTPUT started at $FF67. I came up with a 23 byte machine language program that passes the line feed code (10) to the screen and intercepts and trashes the line feed before it gets to the printer. The printer sees a carriage return and does an automatic carriage return/line feed.

The listing is as follows:

```
202DBF LFFIX JSR 4BF2D ;
       SCREEN PRINTER
48           PHA
AD0502       LDA 40205 ;
       CHECK FOR SAVE FLAG
F008         BEQ EXIT2
68           PLA
C90A         CMP #$0A  ;
       CHECK FOR LINE FEED
F006         BEQ EXIT3
4C73FF EXIT1 JMP $FF73
4C94FF EXIT2 JMP $FF94
4C95FF EXIT3 JMP $FF95
```

Notice that this coding is relocatable any place in memory as long as it isn't interfered with by other programs.

I have a C1P (original, not Series II) and I locate my routine in one of the following places depending on programs in use: $00D8, $0222, or in the stack. I don't have a memory map of the C1P Series II, so I hope that there is similar places to locate the LF fix program. I load this program in the conventional manner of BASIC POKEing numbers into memory using DATA statements. At the end of my BASIC program, I POKE the decimal equivalent of the low byte (start of program) into location 538 and I POKE the decimal equivalent of the high byte into location 539. I'm not sure if this works for the Series II, but if a person had a memory map, it would be a snap to get this program to work.

One minor sidelight is that when this program is in use, not only is the line feed removed from the printer, it is also removed from the tape output. This hasn't posed any problems upon LOADing any BASIC programs SAVEd in this manner. If you don't want this effect on the tape output, then do a <BREAK> <W> before a SAVE. This WARM START resets the output vector at 538 and 539.

Douglas Eichmann
Sioux Falls, SD 57103

* * * * *

ED:

I am working with a Superboard II, and lately I have been busy making improvements on the Monitor Rom, filling all the space $F800-$FFFF with useful routines such as improved screen driver, high speed binary tape save-load and other.

I burned the new monitor into an EPROM, and sent a copy to a friend, who has a Compukit UK 101, the English version of the SB. The Monitor worked fine in his machine, but his Basic consistently gave SN ERROR.

An inspection showed, that UK Basic in at least one place is different from OSI Rom-Basic. The "Fill Input Buffer until CR" routine, located at $A357 makes a JMP to the Monitor. Therefore, to work with UK-Basic, the Monitor must include the following code:

```
FCD5 C9 1C      CMP     1C
FCD7 F0 03      BEQ     FCDC
FCD9 4C 74 A3 JMP     AC74
FCDC 4C 59 A3 JMP     A359
```

Gerdt Vilholm
Greenland, Denmark

* * * * *

ED:

With reference to the Letters Column in Vol. 4, No. 1, I have been using (and modifying) a home-brew terminal simulator program in my (originally) CII-8P for the last year. I would like to try to answer several questions.

Mr. Frank Nelson asked about the BREAK function and the 6850 ACIA. As you answered, the ACIA programming works, but it will not produce a long enough signal to be recognized by many host processors. After setting the ACIA's control register to (decimal) 96, program a delay of at least two or three character-times, increase the delay until the host recognizes your BREAK. In some cases it may take up to 250 milliseconds.

Mr. Tim Lowe also asked about terminal software. My terminal emulator does not mask the high-order bit, and I use CompuServe almost daily. From the information provided in his letter, another possibility is that his software program cannot execute fast enough to read every character (in fact, from the examples, it appears to be catching only every third or fourth character). Check the basic clock rate of the C1P (probably it should be at least one megahertz) and then check that there are no clock slow-down states enabled when accessing the ACIA or other system hardware used by the terminal software. My one megahertz system works fine at 300 baud, but exhibits similar failures when trying to work at 1200 baud.

Mr. Gary Levine asked about sending an interrupt to a host, most systems that I am familiar with do not accept an ASCII 'null'as an interrupt or BREAK. While the 6850 ACIA

(Motorola) documents define a BREAK as a space (logic 0), this is not the same as an ASCII 'null'. A 'null' will still have start, stop, and parity bits while a true BREAK will be a steady logic 0 for several characters duration. Your hardware answer to Mr. Nelson may be the solution here.

I hope this helps.

Alan G. Albright
Escondido, CA 92027

* * * * *

ED:

This is a short subroutine that will allow an output of a null (00) very simply as well as any other of the many ASCII characters that are not accessible from the OSI keyboard. In the illustrated program, I used control letter G which is ASCII 7. The routine simply substracts seven from the value of C. This results in the null output. Also, for instance, if you wanted to output rubout (127) you would just add one twenty (120) to the control letter G (7) and obtain an ASCII value for CH of 127.

This is suggested as possible use by Gary Levine, Denver, January '83 issue, page 22.

```
10 INPUTA$
20 C=ASC(A$):IFC=7THENGOSUB
   1000
30 GOTO10
1000 A=64512:B=A+1:POKE517,1
     :CH=C-7:WAITA,2:POKEB,
     CH
1005 POKE517,0
1010 RETURN
```

M. Bernstein
Asbury Park, NJ 07712

* * * * *

ED:

The ability to change the name of a disk file is a facility that most users of disk operating systems occasionally need. Those of us who use Steve Hendrix's HEXDOS operating system have found that this is often not as easy as it appears. The problem is that although the HEXDOS directory located on track 1 is a listable BASIC file, it cannot be modified by merely re-entering a line with a new file name if the file name contains any BASIC keyword. The reason for this is that since BASIC thinks one is entering a program statement it converts any keyword present to its appropriate token. To test this, create a file named 'TEST'. Load and list the directory with 'LOAD/:LIST'. Change the directory by entering 'n WORDPROCESSOR' where 'n' is the track number of the 'TEST' file you created. Save the directory by entering 'SAVE#1,2817'. List the directory by again entering the command 'LOAD/:LIST'. Everything looks OK! Now try to load the file by entering 'LOAD"WORDPROCESSOR"'. You will find that you get an Fd ERROR. The problem is that BASIC tokenized the two OR's in your new file name.

The following RENAME program solves this problem for the HEXDOS user by providing a procedure for changing file names. It will work for disks in either drive A or B. It requires that the user enter the old file name and then the new file name. If the old file name is not found then the question is repeated. Entering a null string for the old file name terminates the program without modifing the directory. Entering a null string for the new file name returns the program to the old fle name question. CTRL-C is disabled during program execution.

```
2 REM RENAME FOR HEXDOS-
  12/15/82
4 PRINTCHR$(26):AD=(PEEK
  (49152)AND64)/64:IFAD=
  1THEND$="A":D=0
6 IFAD=0THEND$="B":D=128
8 PRINT"Drive "D$" active.
  Press C":PRINT"three times
  to continue"
10 FORI=1TO3;IFUSR(0)<>67
   THENEND
12 NEXTI:PRINT:POKE530,1:
   B=4096:B2=B+2048:LOAD#
   (1+D),B:E=B
14 IFPEEK(E+1)>0THENE=PEEK(E)
   +256*PEEK(E+1)+B-2817:
   GOTO14
16 E=E+1
18 A=B:PRINT:INPUT"Old
   filename";O$:IFO$=""
   THEN44
20 A=PEEK(A)+256*PEEK(A+1)
   +B-2817:IFPEEK(A+1)=
   0THEN18
22 FORI=A+4TOLEN(O$)+A+3:
   IFPEEK(I)<>ASC(MID$(O$,
   I-A-3,1))THEN20
24 NEXTI:IFPEEK(I)>0THEN20
26 INPUT"New filename";N$:
   IFN$=""THEN18
28 CS=LEN(N$)-LEN(O$):IFCS<0
   THENFORJ=ITOE:P=PEEK(J):
   POKEJ+CS,P:NEXTJ
30 IFCS>0THENFORJ=ETOISTEP-1:
   P=PEEK(J):POKEJ+CS,P:
   NEXTJ
32 FORK=1TOLEN(N$):POKEI-LEN
   (N$)+K-1+CS,ASC(MID$
   (N$,K,1)):NEXTK:P=B
34 IFPEEK(P+4)=165THEN40
36 FORI=P+4TOB2:IFPEEK(I)>0
   THENNEXTI
38 I=I+1:T1=I-B+2817:T=INT
   (T1/256):POKEP,T1-256*T:
   POKEP+1,T:P=I:GOTO34
40 I=P+6:T1=I-B+2817:T=INT
   (T1/256):POKEP,T1-256*T:
   POKEP+1,T
42 POKEP+5,0:POKEP+6,0:
   POKEP+7,0:SAVE#(1+D),B
44 POKE530,0:END
```

Jim Hays
Seattle, WA 98116

* * * * *

ED:

In reply to Guy Vanderwaeren's article in the Feb '83 issue (p.12), let me say it is a well thought out plan. His idea of upper and lower address boundaries is farsighted. However, there are a couple of changes I would make.

First, I see no need to invert lines A10-15 going to U18-21. Eliminating U17 would save board space and wiring time. Second, I would change the 1K ohm resistor between U25-11 and ground to 330 ohm. This would insure that pin 11 is pulled below .8v in the absence of an output from DD1 or DD2.

Cheapskate that I am, I'd use 74LS42s in place of the 74LS138s for U23-24 (they're usually 3 to 4 cents less).

I'd like to hear from anyone that has used the newer 2K x 8 RAMs and the 74C series logic ICs. This combination would certainly require less power than previous designs.

Bruce Showalter
Abilene, TX 79601

* * * * *

ED:

The reason I'm writing is twofold. One, do you have any recommendations for "FORTH" support, preferably not too expensive. I presently have a copy of "FORTH" with tiny "PASCAL" from Progressive Computing and frankly, the documentation stinks. I hope there is something better. The other reason is I was intrigued by Jeff Easton's comments in the FEB issue about putting the 6809 on the OSI bus. I think that is a super idea and hope you can convince him to continue and then publish an article on "how to". I can't think of a better combination than OSI's video for games and "FLEX" with "SS-50" bus software in the business area where OSI is lacking. That would be a really super setup.

Neil Dennis
Bliss, NY 14024

* * * * *

ED:

In reference to the letter from Roger Miller, January '83, page 22.

I cannot see any connection at all if the TV mod was done properly. I have done dozens and had no problems, except overscan which is entirely different. I never use an AC-DC set, because hum can be fed to the computer from the TV (besides the safety factor.) The frequencies are similar, but not the same.

Gene Baldwin
Longmont, CO 80501

* * * * *

USER GROUP NOTES:

On at least two occasions in the past, you have been very kind in mentioning us to your readers. Unfortunately, both times the address given has been wrong! The mailing address for the OSMOSUS NEWS is: 3128 Silver Lake Road, Minneapolis, MN 55418, Attn: Donn Burke Baker.

* * * * *

We would like to spread the word that an OSI users Group Net has been started on Sunday afternoons at 2000z on the frequency of 7.229MHz. We wish to invite all amateur radio operators interested in or having OSI equipment to join in with us. The net control station is WBSWRQ, John in Bellaire, Ohio.

Charles F. Merica N4IF
Covington, VA 24426

* * * * *

A number of members have asked us to publish annotated listings of various standard OSI software, notably the 9-digit BASIC used by OS-65D and 65U, with notes on the routines, in much the same way as we covered ROM BASIC in our early issues. It seems to us that the best way of doing this would be to produce supplements on such software, to be offered to members as a separate publication from the Newsletter itself (to do the job properly would take up a good deal of space). To this end, we would like to hear from members who have some form of annotated disassembly of, to begin with, OSI's 9-digit BASIC or ROM BASIC, as the two are very similar. We also need to construct a zero-page map. We are particularly interested in the mathematical routines, as this is where our knowledge of the system is a touch hazy. If you feel you can help in these areas, please get in touch.

OSI/UK User Group
12 Bennerley Road
London SW11 6DS, England

* * * * * * * * * * * * * * *

NEW PRODUCTS ANNOUNCEMENT

It's been a bit late in development, but Generic Computer Products would like to make a few product announcements. The first is a high resolution color graphics board for the OSI. COLOR+ plugs into the 16-pin bus (found on the A-15 board in the back of disk-based C4 and C8 systems, as well as on the CA-20 board). This board features 256 by 192 resolution in 15 colors, optional joystick controller (handles up to 4 with 256 bit resolution), and RF modulator. Any or all of the 15 colors can be displayed in the high-resolution mode, with the minor restriction that only two colors can be displayed in every group of eight horizontal dots. Up to 32 programmable "sprites" are available, which are object-oriented patterns that can be moved smoothly across the screen without disturbing the background. A low resolution mode of 64 by 48 is available, as well as a 20 by 40 text mode. Software will be provided to enhance OS65-D BASIC, providing a superset of APPLE] [graphics commands.

The next product is one that has been in development for some time now, and which Generic is committed to continually extend and improve. It is called Generos (GENERic Operating System) and is initially to be sold as an assembly language development system. The operating system is itself very powerful, and in the future should support several high level languages including BASIC and C. Generos features device independence through the use of device handlers. Devices are accessible by name (DSK1:, LST:, etc.), with custom handlers easily user-installed. GENEROS generates understandable error messages, too. It is designed to be easily user-extensible. Disk usage is also optimized in that files take up a quantity of 256-byte blocks (instead of full tracks), and never more blocks than they actually need.

In its current form, Generos features a powerful disk-based assembler that generates relocatable code. The assembly source resides on the disk, and the object code and listing file can be sent to the disk. Thus files of great size can be assembled. The TECO text editor is also included. TECO is a powerful text-processing language with such features as variables, conditional execu- tion, labels, subroutines (macros), and iterations. Probably the most widely used editor, this implementation contains 95% of the features found in the current release for the DEC PDP-11 series, and is likely the most complete version available for 8-bit computers. In the future, TECO will become a complete implementation.

Generos also sports DDT, a powerful machine code debugger with features such as single instruction execution, trace, byte and word search, hex/ASCII dump, and more.

Generic plans to make its popular MEM+ available as a bare board.

Pricing should be established by the time this announcement appears. For more information, contact: Fial Computer, 11266 SE 21 Avenue, Milwaukie, OR 97222, (503)654-9574

AD$

PEEK (65)
The Unofficial OSI Users Journal

P.O. Box 347
Owings Mills, Md. 21117

BULK RATE
U.S. POSTAGE
PAID
Owings Mills, MD
PERMIT NO. 18

DELIVER TO:

GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

P.O. Box 347 • Owings Mills, Md. 21117 • (301) 363-3268

( ) **C1P Sams Photo-Facts Manual.** Complete schematics, scope waveforms and board photos. All you need to be a C1P or SII Wizard, just — $7.95 $ ______

( ) **C4P Sams Photo-Facts Manual.** Includes pinouts, photos, schematics for the 502, 505, 527, 540 and 542 boards. A bargain at — $15.00 $ ______

( ) **C2/C3 Sams Photo-Facts Manual.** The facts you need to repair the larger OSI computers. Fat with useful information, but just — $30.00 $ ______

( ) **OSI's Small Systems Journals.** The complete set, July 1977 through April 1978, bound and reproduced by PEEK (65). Full set only — $15.00 $ ______

( ) **Terminal Extensions Package** - lets you program like the mini-users do, with direct cursor positioning, mnemonics and a number formatting function much more powerful than a mere "print using." Requires 65U. — $50.00 $ ______

( ) **RESEQ** - BASIC program resequencer plus much more. Global changes, tables of bad references, **GOSUBs** & GOTOs, variables by line number, resequences parts of programs or entire programs, handles line 50000 trap. Best debug tool I've seen. MACHINE LANGUAGE - VERY FAST! Requires 65U. Manual & samples only, $5.00 Everything for — $50.00 $ ______

( ) **Sanders Machine Language Sort/Merge** for OS-65U. Complete disk sort and merge, documentation shows you how to call from any BASIC program on any disk and return it or any other BASIC program on any disk, floppy or hard. Most versatile disk sort yet. Will run under LEVEL I, II, or III. It should cost more but Sanders says, "...sell it for just..." — $89.00 $ ______

( ) **KYUTIL** - The ultimate OS-DMS keyfile utility package. This implementation of Sander's SORT/MERGE creates, loads and sorts multiple-field, conditionally loaded keyfiles. KYUTIL will load and sort a keyfile of over 15000 ZIP codes in under three hours. Never sort another Master File. — $100.00 $ ______

BOOKS AND MANUALS (while quantities last)

( ) **65V Primer.** Introduces machine language programming. — $4.95 $ ______

( ) **C4P Introductory Manual** — $5.95 $ ______

( ) **Basic Reference Manual** — (ROM, 65D and 65U) — $5.95 $ ______

( ) **C1P, C4P, C8P Users Manuals** — ($7.95 each, please specify) — $7.95 $ ______

( ) **How to program Microcomputers.** The C-3 Series — $7.95 $ ______

( ) **Professional Computers Set Up & Operations Manual** — C2-OEM/C2-D/C3-OEM/C3-D/C3-A/C3-B/C3-C/C3-C' — $8.95 $ ______

( ) Cash enclosed ( ) Master Charge ( ) VISA

Account No. ______ Expiration Date ______

Signature ______

Name ______

Street ______

City ______ State ______ Zip ______

TOTAL	$______
MD Residents add 5% Tax	$______
C.O.D. orders add $1.50	$______
Postage & Handling	$ 3.00
TOTAL DUE	$______

POSTAGE MAY VARY FOR OVERSEAS