PEEK (65)

The Unofficial OSI Users Journal

$1.75
January 1983
Vol. 4, No. 1

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

INSIDE:

Column One

It was Karin's familiar voice.

"Could you POSSIBLY come up tomorrow morning and finish proofing PEEK(65)? The printer has just called and told us he has to pick up everything to make the plates tomorrow afternoon!"

As it happened, I could not come tomorrow morning, nor this afternoon. In fact, you will be getting your first look at this issue at the same time I do. Of course, I was in on the selection and editing of the articles and letters in the issue, but I haven't seen the whole issue together yet.

So how did I manage to get Column One written and inserted? In the old days, I would have written it, then telephoned PEEK(65) and dictated it over the phone.

Today, we don't do things that way. Today, I sit down at my computer and write my column with WordStar, then transmit it to PEEK(65) with Ascom. No wonder people are excited by computers. They can be so liberating!.

The PEEK(65) CBBS has finally died. Doomed perhaps by the easy availability of national databases with local numbers to call from every city in the US, the CBBS had faded to just a very few faithful users. We will miss it, but it is not economical to tie up a $10,000 machine (even at night) for 6 of us to use as our private mailbox.

See you on MicroNet!

A couple of months ago, I asked readers to tell me of any good software they know of which provided true record-level locking for multiuser systems. Responses to date have been sparse:

Dick McGuire reports that he has implemented a Master File Editor under OS-65U which used 65U's WAIT FOR and WAIT CLEAR instructions to achieve record locking, but as yet no full "turnkey" applications;

OSI's new 350 announced at COMDEX, running KeyOperator 1, which runs most standard CP/M applications programs, boasts of "multiuser environment, with multiple directories and record/file locking";

#OASIS and UNIX both feature built-in record locking, but I have seen very little applications software to run under either system. The DataPro report on microcomputer software lists 22 Accounts Payable packages which run under CP/M, 2 for MP/M, one of which mentions record locking, and 2 for UNIX -- none for OASIS;

Several operating systems which are "derivatives" of CP/M and offer each user a CP/M look-alike interface have the ability to perform record locking, but only if the program running at the time asks for it;

Don't talk to me about Ethernet, powerful business micros, etc., etc., -- until you can show me a micro OS with record level locking and a wide selection of software, not an "application development system"!!

This month's content, it seems to me, reflects our philosophy of trying to serve all our readers. From the hacker (SUB$ + UP$ and Hendrix' continuing excellent series on Basic) to the business user (Ref manual and other bugs in 65U V 1.42 level III).

Will next month's PEEK(65) be as good, or better? It's up to you. As of now, we don't know what will fill next month's pages. YOU have to send it to us. Tell us what you are doing with your computer, or how you solved a past problem. Send us an article, a letter, or at least a nice card!

Next month, if all goes well, we will review the new OSI multiprocessing computer which can run CP/M and converted 65U concurrently. I am not sure how they do it, but the concept is exciting.

al

WHAT NEXT, OSI?

From Cheiky entrepreneuralship to MA/COM Fortune 500 to Kendata venture and back to "Ohio Scientific, Inc."

Once again, publishing constraints work against PEEK. We must go to press and sit here frustrated in the knowledge that Kendata will be making announcements within days. Regardless, after many months of silence on OSI's future, you, our readers, are entitled to the best information we can give and we will tell you what we do know and what we have heard.

In midsummer, MA/COM's desire to sell its interest in OSI became common knowledge, courtesy of the Wall Street Journal and others. Then, on November 23, Kendata, Inc. announced its purchase of OSI, which it intends to operate as a wholly-owned subsidiary.

The official release said, "Mr. Kenneth Wortz, President and Chief Operating Officer of Ohio Scientific, Inc. and Kendata, Inc., said that "we're going to change the name back to Ohio Scientific, Inc. to take full advantage of the Company's excellent reputation among dealers and the market, plus the awareness it has earned over the years. We remain committed to expand Ohio Scientific's leadership position in microcomputer technology. Our plans include strengthening distribution channels, streamlining headquarters and manufacturing operations and introducing at COMDEX in Las Vegas new advanced KEYFAMILY multiprocessing and integrated office systems hardware and software."

Speaking of the new machine announced at COMDEX, it's called the 350 series and from the outside looks just like a 250. Inside, it has a System Processor Board that contains a Z80, system monitor port, serial printer port, clock boot ROM, 56K RAM and IBM format floppy disk drive controller. Next to the SPB there are up to eight Application Processor Boards, each Z80 based with 64K RAM, serial console port and serial printer port. The SPB handles all disk I/O, print spooling and management and all data communications. A 1meg./sec. bus makes for swift data transfers when coupled with a four channel DMA system. The combination of the KeyOperator 1 operating system and KEYBASIC mean that the system can run many programs written for CP/M and can convert most existing OS-U programs as well. Sounds like a winning combination! The elves have been busy in spite of the long silence.

Who is Kendata? Kendata, Inc. is a new company (August '82), located in Stratford, Conn., which, Chief Executive Officer, Joe Sorrentino, describes as being in the business of marketing smart terminals and peripherals to commercial and end-using customers. This is expected to make a very good and complimentary marriage with OSI's computer lines.

Ken Wortz is in the process of bringing on lots of new management with the explicit job of concentrating their efforts on more products, more support and better service. It would seem natural to expect intelligent terminals and Mr. Wortz says that more and better software is of prime concern. Watch for a new general business package, a new DBMS and enhanced KEY BASIC. Equally important, there are no plans to drop the personal line.

Ken Wortz is an affable and determined man who, with just three weeks under his belt, most certainly has his hands full just assessing the current situation, not to mention detailed plans for the future. With his past experience with M/A-COM Alanthus under Carl English, we expect to have those definitive plans and directions for OSI to report in the February issue of PEEK.

The Editors.

★ ★ ★

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

THE WORKINGS OF BASIC
THE EXECUTIVE ROUTINE

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

This month's routine is the major loop of the Basic Interpreter which executes statements by dispatching the appropriate routines to execute the Basic Verbs contained in the lines. It also handles checking for ctrl-C between lines and skipping over colons which separate statements on the same line. It does not handle the tokenizing of Basic reserved words; that is handled before this routine is called, by the line compression routine mentioned last month.

This routine is called from the immediate mode processor as explained last month. For execution of a program, the RUN, GOTO, and GOSUB verbs set things up so that this routine continues into the body of the program upon return from those routines. This routine is entered at $A5F6. Upon entry, the GETBYTE routine is set so that the next byte it returns will be the first byte of the buffer, and the buffer (as well as all lines in the program) is already tokenized and compressed.

The JSR $00BC at $A5F6 calls GETBYTE, returning the first byte of the line with the Z and C flags set to indicate end of line and ASCII digits, respectively. The JSR $A5FF at $A5F9 and the following JMP are set up so that the routine implementing a particular word can end with an RTS, which will cause a jump to $A5C2 where the interpreter takes care of the housekeeping to get set for the next statement of the same line or the next line, as appropriate.

The routine starting at $A5FF is the verb dispatcher. It looks up the address of the routine implementing the particular verb and calls that routine; it also recognizes the implied "LET" when a line such as A = 0 is executed. Recall that as this routine is entered, the Z and C flags are set according to the first byte of the statement to be executed: if it is a NUL or a colon, the Z flag is set (indicating end-of-statement); if it is an ASCII digit, the C flag is cleared. Thus the BEQ at the $A5FF checks for the end of the statement, branching to an RTS if that is the case.

If the first character of the line is a digit, something is wrong. A program line may not begin with a digit after the line number, and an immediate line starting with a digit would have already have been intercepted as an editing command. Thus we need not consider the C flag here. The next test is essentially a test for a byte greater than or equal to $80, which would be a token for a Basic reserved word. Subtracting $80 and comparing against zero at $A601-$A605 accomplishes this as well as starts to convert the token (if it is one) to an index into the address table. If the result is negative, the BCS at $A604 is not taken, and the JMP $A7B9 jumps to the routine implementing LET, since the only way a statement can begin with something other than a reserved word is to be an implied LET as mentioned above.

If the first byte is a reserved word, we find ourselves at $A609 with the token minus $80 in the A register. Now we do a comparison to see if it is a word which may start a statement. Only the first 28 of the reserved words qualify, so if the token does not fall in the acceptable range, there is a syntax error in the line. The branch is to a JMP to the routine which prints the SN ERROR message.

The ASL at $A60D has the effect of multiplying A by 2. This is because the address table contains two bytes for each word. We move the result to the Y register for use as an index into the table, and then load the two bytes for that word and push them to the stack. The address given in the table points to the byte before the first byte of the routine for the word, which is exactly what should be on the stack for an RTS to cause a jump to the first byte of the routine.

At $A617 we JMP to the GETBYTE routine, apparently giving up entirely since this was a JMP rather than a JSR. Think through what really happens, though: at the completion of GETBYTE, it does an RTS. The last address on the stack is a return address which will cause a jump to the beginning of the routine which executes the verb in question, with the byte appearing after that verb already loaded into the A register and the flags already set. Since nearly every verb must interpret some byte after the verb, this saves some coding in the routine for every verb. A rather roundabout but very efficient way of getting to the code for the verb, indeed!

Oops! We appear to have run out of code to handle executing statements. That is only an appearance, however. Recall back at $A5F9 that we left a return address on the stack which we have not yet used. The routine executing the verb at hand will eventually finish with an RTS, which will return to $A5FC and jump from there to $A5C2, to move along to the next statement to be executed.

As we start to trace the code at $A5C2, you'd better get out a pencil to trace the trail of JSRs and returns as this is going to get a little messy. It will appear to be wastefully complicated, but that was the price of generality in the code.

First we JSR $A629, leaving a return address on the stack. At that location we find a JMP $FFF1, which goes to the monitor ROM. Notice that we have not pushed a return address on the stack with this, so that the next RTS will cause the return to skip past this level directly back to the code at $A5C5.

Jumping into the monitor ROM opens Pandora's box, since there are so many different after-market ROMs around to replace the original. I will explain the routine which originally came with my system, and those of you with other ROMs can just assume that your ROM does something similar, though it may do other things in addition. For instance, ROMs with a trace function intercept the interpreter here to implement that function, since this routine is called after every statement is executed.

Moving to $FFF1, we find an indirect jump through $021C. This is where the address stored in page two comes into play. The address normally stored there is $FF9B, so the JMP ($021C) at $FFF1 may be likened to a JMP $FF9B. Notice again that no return address was left on the stack, so the next RTS will still go back to $A5C5.

Starting at $FF9B, the first thing that happens is a test of $0212 (decimal 530), which is the control for whether or not ctrl-C will be recognized. It is also used by some of the supplemental routines such as the trace used by HEXDOS. If this byte is zero, ctrl-C is recognized normally by continuing through the rest of the routine. If not, the BNE branches to the RT$ at $FFB9, causing us to return finally to the executive routine through the return address left on the stack earlier.

If $0212 is zero, the code from $FFA0 thru $FFA8 checks to see if the ctrl key is pressed. Storing $FE to the keyboard port at $DF00 activates the last row of the matrix, including the ctrl key, all the shift keys, the repeat key, and the esc key. The BIT instruction at $FFA5 tests bits 6 and 7 of the port, effectively setting the N flag if no key is pressed in column 7 of the selected ROM(s) and setting the V flag if no keys are pressed in column 6 of the selected row(s). Since only the last row is selected, the BVS at $FFA8 effectively tests to see if the ctrl key is pressed and branches if it is not, since the only active row at this point is the control row, and the ctrl key appears at the 6th column in that row.

The code from $FFAA thru $FFB3 uses a similar method to determine if the C key is pressed. If the branch at $FFB2 is not taken, then both the ctrl and C keys must have been pressed. The LDA at $FFB4 is to set up an indication that the stop was due to ctrl-C as opposed to a normal end to the program, for the routine which is jumped to by the JMP $A636 at $FFB6. If either the ctrl or C key were not pressed, the routine finishes with an RTS, returning to $A5C5.

The code at $A5C5 takes care of finishing up and skipping over the end-of-statement mark and determining if there is another statement to be executed. The Y and A registers are loaded with the current byte pointer at $A5C5. If Y is now zero, we are working on a line in the immediate mode (the terminal buffer is in page 0) and can skip over saving the pointer. The BEQ at $A5C9 accomplishes this. Notice that it also skips over loading Y with 0, since by the fact that the branch was taken we already know that Y contains 0. This is a very tiny effort at a slight speedup, but there was no reason not to do it. $A5CB-$A5CE saves the location of the current byte at

OSI COMPATIBLE PRODUCTS

56K 2-MHz Ultra Low Power CMOS Static Memory Board **MEM-56K $850**

Partially Populated Boards (Specify address locations required) ... **MEM-48K $750**
MEM Board uses the new 2K-Byte Wide Static RAM chips which are 2716 EPROM compatible. Any 2K byte memory segment can be populated with RAM or EPROM (or left empty for use of Address Space by another board). Fully expandable to any memory size you will ever need. No special addressing requirements, just solder in extra sockets

MEM-32K $550
MEM-24K $450
MEM-16K $350
MEM- 8K $250
MEM- 4K $200

Extra 2K RAM Memory Chip **$24**
Optional Parallel Printer Port **-P $120**
Optional Calendar/Clock Software available in EPROM) **-T $ 25**
Both options (Disk software mods provided for use of 6522 VIA on printer). **-PT $125**

EXAMPLE USES:

C4P & C8P: Expansion to **4K RAM** of Basic workspace.
Parallel Printer Port — Reserve Serial Port for MODEM
Calendar/Clock Displaying on unused portion of screen.
Space for 5.75K of **Enhanced System Monitor EPROMS.**

All of this on 1 Board, using only one of your precious slots. Software for Enhanced System Monitor capabilities is continuously being developed and improved. As new EPROM Monitors are available, you may upgrade to them for any price differential plus a nominal $10 exchange fee. Another possibility is to fill any portion of the memory with Basic Programs in EPROM for **Power-on Instant Action.** This custom EPROM programming service is available at $25 per 2716 (Includes EPROM). Extra copies at $15 for each EPROM.

C4P-MF & C8P-DF:

Memory expansion to **48K.**
Add 6K **Memory** above BASIC for special software requirements.
Parallel Printer Interface and/or **Displaying Calendar/Clock.**
Add 1.75 K **Enhanced System Monitor ROM.**

C3: Up to **56K of Memory** Expansion — can be addressed for **Multiuser.** (Optionally, each user can have his own **Dedicated Printer Port).**

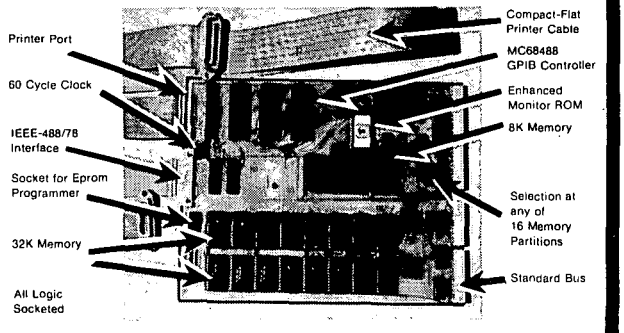
C1P,C4P & C8P FLOPPY DISC CONVERSIONS:
Memory/Floppy Board (Includes MI48P1 ROM) ..... **MEM F-16K $450**
C1P-600 Board Adapter & Cable ..... **A600/48 $ 50**
Additional Memory/Printer/Times ..... **(See MEM Board Prices)**
5¼" Drive/Case/Power Supply & Cable to MEMF Board **FD5 $399**

IEE-488 INTERFACES AND SOFTWARE:
The General Purpose Instrumentation Bus (GPIB Controller interface is available for all OSI Computers. Machine code GPIB Drivers are linked to Basic to provide easy control of IEEE-488 instruments which is equal to the best of Hewlett-Packard Controllers and far superior to most others. Basic Commands for Serial Poll, Parallel Poll, IFC Clear, full Local/Remote Control, Respond to SRQ Interrupts, Send Trigger, do Formatted Input/Output, Direct Memory Input/Output and MORE. Interface includes IEEE-488 Ribbon Cable/Connector.

GPIB Controller Interface for C2, C3, C4 and C8 Systems ..... **GPIB 4-488 $395**
GPIB Software for **OS-65D** (Add -8 for 8" or -5 for 5") ..... **GPIB 488-D $ 70**
GPIB Software for **OS-65U** ..... **GPIB 488-U $100**
GPIB Software on two 2716 EPROMS for **ROM** based systems ... **GPIB 488-R $100**
Add Optional **Parallel Printer Interface** to GPIB 4-488 ..... **-P $120**
Add Optional **Calendar/Clock** to GPIB 4-488 ..... **-T $ 25**
Add **2K RAM** to GPIB 4-488 (Specify location, $4000-$BFFF & $D000-$$EFFF available) **-M $25**
GPIB Controller for C1P, Includes Software, Clock, All Features of ROMTERMS, & space for 6K EPROM ..... **GPIB 6-488R $395**
Add Optional Parallel Printer Interface to GPIB 6-488R ..... **-P $120**

EPROMS:
C1P ROM with 24/48 Col Display for Series II, Smart Terminal, Line Editing, Corrected Keyboard Screen Clear and More ..... **ROM-TERM II $59.95**
C1P ROM with 24 Col Display. Other ROM-TERM II Features, Disk Boot, and ROM/Disc Basic Interchange ..... **ROM-TERM $59.95**
C4P-MF/C8P-DF Disk warm start, changed IRQ Vector and just flip switch for Serial or Video System with Corrected Keyboard ..... **SYNKEY $39.95**
ENHANCED MONITOR ROMS FOR USE ON GPIB 4-488 & MEM BOARDS:
Expanded Support for **C4P & C8P** Featuring Calendar/Clock, Line Edit, Smart Terminal, Memory Files, Parallel Printer Control, Corrected Keyboard, All Features of ROMTERMS, **Disk Support** with Warm Start and More **MI48P1 $59.95**
Expanded C2 Monitor with Calendar/Clock Software, Hard Disk Boot, Warm Start and Multi-User Control for C2 Systems **MIC2-1 $59.95**

IEEE-488 CONTROLLER INTERFACE



THE GPIB 4-448 INTERFACE BOARD CONVERTS ANY OSI COMPUTER INTO AN IEEE-488 INSTRUMENT BUS CONTROLLER!

BENEFITS — Provides a Sophisticated Instrumentation Controller at very low cost (often saving thousands of Dollars). The combination of IEEE-488 Instrumentation Controller and High Capacity Hard Disk file storage available on OSI Computer systems is available at a fraction of the cost required by the nearest competitor. The IEEE-488 Bus, also known as the GPIB,HP-IB or IEC-625 is the most popular International Standard for connecting instrumentation systems. This 16-line bus is designed to interconnect and control up to 15 instruments at a time. Currently, over 2000 different instruments are available to work on this bus. They include: Plotters, Digitizers, Printers, Graphic Displays, Recorders and a multitude of specialized Test/MeasurementControl Equipment.

EPROM-ABLE — Can be used with a C4-P to create a dedicated IEEE-488 controller.

C2-D MULTIPLE USER SYSTEMS

SAVE — 2 and 3 user Time Sharing Systems are available on the C2-D Winchester Disk Computer at a considerable cost savings from C3 Multiple User Systems. The 3 user C2-D System can be expanded to include a word processing printer, 4 other parallel printers and 3 serial printer interfaces.

COMPATABLE — The special C2-D Multi-User Executive Program is 100% compatable with OS-65U V1.4. The Multi-User Real Time Clock, Memory Partition Control and IRQ Interrupt Management are done on the Micro Interface Memory Board. Thus, the CPU board is not modified and remains in factory condition.

CONVERSIONS — The Up-Grade of your existing C2-D Computer to Multiple User Configuration is also available. Call for details.

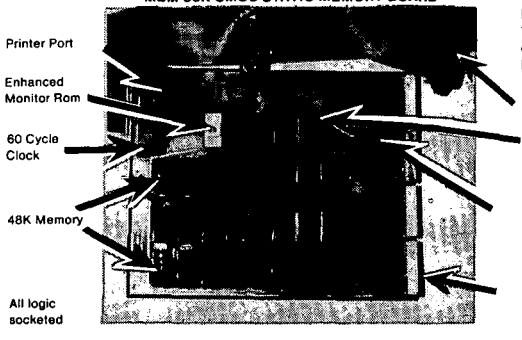
FLOPPY DISK UPGRADES FOR C1P, C4P & C8P

Our Memory/Floppy Board provides easy conversion of 502 and 600 CPU Computers to Floppy Disk Operation. The **MEMF** Board has a floppy disk interface which includes a data separator and the ability to automatically lift the disk drive heads — your floppy disk lifetime will be extended many times. You will retain the cassette interface for your existing software; which can easily be converted to Disk.

This **MEMF-16K** Board is populated with **16K RAM** (50K possible) and has features of the MEM CMOS Static Memory Board with an added floppy interface. The low power memory means extra power supply not required. **ROM Basic** is retained even when Board is populated for 48K Disc Basic. An optional Parallel Printer Port and Real Time Calendar/Clock is on board.

Complete Ready to Run conversion kits with 5¼" or 8" Disk Drives are available.

MEM-56K CMOS STATIC MEMORY BOARD



ULTRA-LOW POWER — By using CMOS Static RAM Memory, the total power consumption is about ½ Amp at 5 Volts when populated for 48K. In fact, most of power is used by the Address Line Buffers and the Data Transceivers.

MULTI-USER — Can be addressed for any of the 16 multi-user memory partitions. The low power and single memory board/partition simplify installation and provide a typical $1400 saving for a 3-user system.

MICRO-INTERFACE
3111 SO. VALLEY VIEW BLVD., SUITE I-101
LAS VEGAS, NEVADA 89102
Telephone: (702) 871-3263

Check with your local Dealer or Order Direct.
Phone orders accepted.
TERMS: Check/Money Order/Master Charge/VISA
Sent POSTPAID ON PREPAID ORDERS.
Foreign Orders: Prepaid only.
Add 5% for handling/shipping.

$008B-$008C. This information is needed for a few of the verbs such as GOSUB, FOR, and INPUT.

The LDA at $A5D1 gets the current byte for testing to see if there is another statement on this line to execute. It would appear that we could accomplish the same thing by a JSR to REGETBYTE, but the flag settings then would not distinguish between a null and a colon, which is critical here. If the byte is a null, the BEQ at $A5D3 branches to the code which handles the end of a line, be it a program line or an immediate line entered from the keyboard. I will take up that portion shortly.

If the branch is not taken, the byte must be a colon or there is a syntax error (a byte which should have been part of the last statement was not accepted). The comparison and BEQ at $A5D5-$A5D8 tests for the colon. If the byte is not a colon, the JMP $AC0C at $A5D9 jumps to the error routine which prints the SN ERROR message, returning to the immediate mode and we are finished. If the byte was a colon, we are back where we started, with the next byte to be returned by GETBYTE being the first byte of a statement, so the BEQ at $A5D7 jumps back to $A5F6, starting the whole process over.

Now back to the case where the byte is a null, indicating the end of the line. The bytes following that null will be the beginning of the next program line, assuming we are executing a program rather than an immediate line. The first two bytes will be a pointer field, whose second byte is non zero if there is a line there to be executed. The end of the program is signalled by two zero bytes appearing immediately after the end of the last line. The code at $A5DC-$A5E1 tests to see if the second byte after the end of the line is a zero, branching to a routine which does little more than return to the immediate mode if it is. In an immediate mode line, a zero was placed two bytes after the end of the line by text compression routine, so that this logic would work equally well on an immediate mode line or on a program line.

If we are executing a program and there is a next line to execute, a little housekeeping is in order before starting in

LISTING 1

```
$A5C2 JSR $A629      ; CHECK FOR CTRL-C
$A5C5 LDA $C3        ; GET THE NEXT-BYTE PTR
$A5C7 LDY $C4
$A5C9 BEQ $A5D1      ; IMMEDIATE MODE
$A5CB STA $8B        ; SAVE FOR SOME VERBS
$A5CD STY $8C
$A5CF LDY #0
$A5D1 LDA ($C3),Y    ; GET NEXT BYTE
$A5D3 BEQ $A5DC      ; END OF LINE
$A5D5 CMP #$3A       ; COLON
$A5D7 BEQ $A5F6      ; END OF STATEMENT
$A5D9 JMP $AC0C      ; SYNTAX ERROR
$A5DC LDY #2         ; CHECK FOR END OF PROGRAM
$A5DE LDA ($C3),Y
$A5E0 CLC            ; FLAG NORMAL FINISH
$A5E1 BEQ $A651      ; END OF PROGRAM
                     ; JUMP BACK TO IMMEDIATE MODE
$A5E3 INY            ; LOAD NEXT LINE #
$A5E4 LDA ($C3),Y    ; AND SAVE AT $87-$88
$A5E6 STA $87
$A5E8 INY
$A5E9 LDA ($C3),Y
$A5EB STA $88
$A5ED TYA            ; MOVE NEXT-BYTE PTR TO
$A5EE ADC $C3        ; BEGINNING OF NEXT LINE
$A5F0 STA $C3
$A5F2 BCC $A5F6
$A5F4 INC $C4
$A5F6 JSR $00BC      ; MAIN ENTRY POINT
$A5F9 JSR $A5FF      ; DISPATCH ROUTINE FOR VERB
$A5FC JMP $A5C2      ; HANDLE END OF STATEMENT

$A5FF BEQ $A579      ; TO AN RTS IF A NULL LINE
$A601 SEC            ; TEST FOR TOKEN AND
$A602 SBC #$80       ; START CONVERTING TO INDEX
$A604 BCS $A609      ; YES - IT IS A TOKEN
$A606 JMP $A79B      ; MUST BE AN IMPLIED "LET"
$A609 CMP #$1C       ; CHECK FOR VALID VERB
$A60B BCS $A5D9      ; NO - ITS AN ERROR
$A60D ASL            ; MULTIPLY BY 2
$A60E TAY            ; FOR USE AS AN INDEX
$A60F LDA $A001,Y    ; GET HIGH BYTE OF ADDRESS - 1
$A612 PHA
$A613 LDA $A000,Y    ; AND LOW BYTE
$A616 PHA
$A617 JMP $00BC      ; GETBYTE, RETURNING TO VERB ROUTINE

$A629 JMP $FFF1

$FFEB JMP ($0218)    ; INPUT
$FFEE JMP ($021A)    ; OUTPUT
$FFF1 JMP ($021C)    ; CTRL-C
$FFF4 JMP ($021E)    ; LOAD
$FFF7 JMP ($0220)    ; SAVE

$FF9B LDA $0212      ; CTRL-C ENABLE
$FF9E BNE $FFB9      ; DISABLED
$FFA0 LDA #$FE       ; MASK TOENABLE JUST CTRL KEY ROW
$FFA2 STA $DF00      ; KEYBOARD PORT
$FFA5 BIT $DF00      ; TEST CTRL KEY
$FFA8 BVS $FFB9      ; NOT DEPRESSED
$FFAA LDA #$FB       ; MASK TO ACTIVE ROW CONTAINING "C"
$FFAC STA $DF00
$FFAF BIT $DF00      ; TEST "C" KEY
$FFB2 BVS $FFB9      ; NOT DEPRESSED
$FFB4 LDA #3         ; CRASH CODE
$FFB6 JMP $A636      ; STOP AND RETURN TO IMMEDIATE MODE
N$FFB9 RTS           ; CONTINUE EXECUTING NORMALLY
```

★ ★ ★

ARTICLE CONTINUED ON PAGE 6.

on the new line. First, the code at $A5E3-$A5EC loads the line number (the two bytes immediately after the pointer field) and stores them at $0087-$0088 for use by the error routine and the BREAK routine in telling what line number caused the program to stop.

This process leaves the Y register loaded with a 4, which is the amount to be added to the current-byte pointer to leave it set correctly for starting the next line. Rather than a LDA with an immediate 4 (which takes 2 bytes) the TYA accomplishes the same thing in one byte. The remainder of the code at $A5EE-$A5F5 adds this 4 to the next-byte pointer. After this, we are back to the beginning of the routine at $A5F6, ready to execute the next line or statement.

Next month: A 59-byte block delete function.

★

65U SUB$ AND UPS$

by: Robert Camner
The Maret School
3000 Cathedral Avenue, N.W.
Washington, D.C. 20008

I have been a subscriber to PEEK(65) for over two years, and have been a very successful moocher. You have saved me countless hours of work, and have enabled me to do things with my machine that I would not have otherwise thought to do. I have always been too busy using my machine to share with others the fruits of my labors. I will try to make partial amends with this article.

Everything I'm about to say refers to 65U. BASIC-in-ROM and 65D readers may disappointedly skip to the next article or letter.

First, a 65U "gotcha." Take any recent version of 65U (say V1.3 or V1.42, or any rather late V1.2) and run the program in figure 1, after creating a data file called FILE2. I'd save the program first, unless you wish to type it in again. Now reboot, load the program, delete line 50, and run it again. Quiz question: this bug has not always been in 65U; when did it first appear?

When we first bought our OSI machine, I had to convert a bunch of programs that had previously run on an HP2000. I discovered to my chagrin that Microsoft BASIC has no substring function (ie INSTR$, SUBSTR$, or equivalent). It can be emulated by the following, of course:

```
100 TA$="THIS IS THE TARGET
     STRING"
110 SE$="IS":REM we will
    search for this string
120 FOR I=1 TO LEN(TA$)
    -LEN(SE$)+1
130 IF MID$(TA$,I,LEN(SE$))
    =SE$ THEN PRINT"EUREKA":
    END
140 NEXT I
150 PRINT"DRAT!!!"
```

Not only is that program rather slow, but it creates garbage in string space at a rapid pace. As a means to teach myself 6502 machine language, I wrote SUB$ and UPS$, the assembly listings of which are given in figs. 2 and 3, respectively (will machine code experts please forgive me and bear in mind that these were my very first machine code programs!). The first portion of each routine was adapted from various OSI Tech notes. Figure 4 gives a BASIC program that will implement these statements by poking the code into the top 2 pages of RAM (you will need to reassemble them if yours is not a 48K system. Figure 5 gives a BASIC program that should explain the usage of these statements. Please bear the following in mind:

(a) SUB$ and UPS$ are not functions, but statements (I couldn't figure out how to make them functions; I kept getting SN ERROR when I branched to the code, which never even began to execute).

(b) All the arguments must be variables; they may not be constants!

(c) UPS$ leaves non-letters and upper case letters unchanged.

(d) The READ and DATA statements are disabled by these commands (I do not use an overlay, but I need the space in the reserved word list).

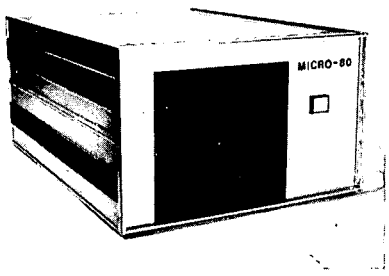
I do not use UPS$ particularly often, but SUB$ has sped up many of my programs written for administrative use here at the school.

We have the following setup at the Maret School: a C20EM that is a C8PDF in one box, and 5 C1Ps that tie into the C2 via LEVEL 1 networking. I have a snazzy network executive that allows each C1 to think it really has access to a time-sharing system (ie true two-way communication and single entry commands unlike that piece of trash, MULTI, that OSI provided to attempt to do the same job!). I know it's common to toot one's own horn, but another local school plus a large local public school system have bought the system and the executive, so I'm not guilty of being the only person in the world who thinks its worthwhile. I don't know why other manufacturers haven't put together a decent network for microcomputers at an affordable price. OSI's hardware (with some reasonable software) outperforms any microcomputer network I've seen and at a significantly lower cost.

Answer to the quiz question: this bug came into 65U through a fix thoughtfully provided by the folks at OSI in Tech Note #29, page 3. The description of the problem said that rarely, an attempt to read past an end of file would hang up the system. Of course, I couldn't create a crash with my pre-tech note V1.2. As soon as I implemented the fix, I could get a consistent system hang by trying to read past the end of file. The bug persisted through V1.3 and now is in V1.42.

LISTINGS START PAGE 8

NEW FROM D & N MICRO PRODUCTS, INC.



MICRO-80 COMPUTER

Z80A CPU with 4MHz clock and CP/M 2.2 operating system. 64K of low power static RAM. Calendar real time clock. Centronics type parallel printer interface. Serial interface for terminal communications, dip switch baud rates of 150 to 9600. 4" cooling fan with air intake on back of computer and discharge through ventilation in the bottom. No holes on computer top or side for entry of foreign object. Two 8" single or double sided floppy disk drives. IBM single density 3740 format for 243K of storage on each drive. Using double density with 1K sectors 608K of storage is available on a single sided drive or 1.2 meg on a double sided drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane for expansion. 48 pin buss is compatible with most OSI boards. Uses all standard IBM format CP/M software.

Model 80-1200	$2995
2 8" single sided drives, 1.2 meg of storage	
Model 80-2400	$3495
2 8" double sided drives, 2.4 meg of storage	
Option 001	$ 95
Serial printer port, dip switch baud rate settings	

Software available in IBM single density 8" format.

Microsoft		**Digital Research**		**Micropro**	
Basic-80	$289	PL/1-80	$459	Wordstar	$299
Basic Compiler	$329	Mac	$ 85	Mail-Merge	$109
Fortran-80	$410	Sid	$ 78	Spellstar	$175
Cobol-80	$574	Z-Sid	$ 95	SuperSort I	$195
Macro-80	$175	C Basic-2	$110	**Pascal**	
Edit-80	$105	Tex	$ 90	Pascal/MT +	$429
Mu Simp/Mu Math	$224	DeSpool	$ 50	Pascal Z	$349
Mu Lisp-80	$174	**Ashton-Tate**		Pascal M	$355
		dBase II	$595		

Convert almost any static memory OSI machine to CP/M® with the D & N-80 CPU Board.

Z80A CPU with 4MHz clock. 2716 EPROM with monitor and bootstrap loader. RS-232 serial interface for terminal communications or use as a serial printer interface in a VIDEO system. Disk controller is an Intel 8272 chip to provide single or double density disk format. 243K single density or 608K double density of disk storage on a single sided 8" drive. A double sided drive provides 1.2 meg of storage. DMA used with disk controller to unload CPU during block transfers from the disk drives. Optional Centronics type parallel printer port complete with 10 ft. cable. Optional Real Time Calendar Clock may be set or read using 'CALL' function in high level languages. Power requirements are only 5 volts at 1.4 amps. Available with WORDSTAR for serial terminal systems.

Includes CPM 2.2

D & N-80	serial	$695
D & N-80	serial w/Wordstar	$870
D & N-80	video	$695
Option 001		$ 80
parallel printer and real time calendar clock		

D & N-80 CPU BOARD

OTHER OSI COMPATIBLE HARDWARE

IO-CA10X Serial Printer Port **$125**
Compatible with OS-65U and OS-65D software

IO-CA9 Parallel Printer Port **$175**
Centronics standard parallel printer interface with 10 ft. flat cable

BP-580 8 Slot Backplane **$ 47**
Assembled 8 slot backplane for OSI 48 pin buss

24MEM-CM9	**$380**	**24MEM-CM9F**	**$530**
16MEM-CM9	**$300**	**16MEM-CM9F**	**$450**
8MEM-CM9	**$210**	**8MEM-CM9F**	**$360**
BMEM-CM9F	**$ 50**	**FL470**	**$180**

24K memory/floppy controller card supports up to 24K of 2114 memory chips and an OSI type floppy disk controller. Available fully assembled and tested with 8, 16, or 24K of memory, with floppy controller (F). Controller supports 2 drives. Needs separated clock and data inputs. Available Bare (BMEM-CM9F) or controller only (FL-470). Ideal way to upgrade cassette based system

C1P-EXP Expansion Interface **$ 65**
Expansion for C1P 600 or 610 board to the OSI 48 pin buss. Requires one slot in backplane. Use with BP-580 backplane

BIO-1600 Bare IO card **$ 50**
Supports 8K of memory, 2 16 bit parallel ports may be used as printer interfaces. 5 RS-232 serial ports, with manual and Molex connectors

DSK-SW Disk Switch **$ 29**
Extends life of drive and media. Shuts off minifloppy spindle motor when system is not accessing the drive. Complete KIT and manual

D & N Micro Products, Inc.
3684 N. Wells St.
Fort Wayne, Ind. 46808
(219) 485-6414

VISA MasterCard

TERMS $2.50 shipping, Foreign orders add 15%. Indiana residents add 4% sales tax.

Disk Drives and Cables

8" Shugart SA801 single sided **$395**

8" Shugart SA851 double sided **$585**

FLC-6 6ft. cable from D & N or OSI controller to 8" disk drive **$ 69**

5 1/4" MPI B51 with cable, power supply and cabinet **$450**

FLC-5 1/4 8 ft. cable for connection to 5 1/4 drive and D & N or OSI controller, with data separator and disk switch **$ 75**

Okidata Microline Printers

ML 82A Dot Matrix Printer **$534**
120 CPS, 80/120 columns, 9.5" paper width, friction or pin feed

ML 83A Same as 82A except **$895**
16" paper width, 132/232 columns with tractor feed

ML 84 Same as 82A except 200 CPS, **$1152**
16" paper width, 132/232 columns, 2K buffer, dot addressable graphics, with tractor feed

by: Robert Camner

Fisure 1

```
10 REM TEST CRASH
20 CLOSE:OPEN"FILE2",1
40 FLAG27:REM for V1.2 make this POKE 2888,0
50 FLAG28:REM for V1.2 make this POKE 2888,27
60 FLAG5:REM index = 1E9 on end-of-file
100 INPUT#1,L$
101 IF INDEX(1)>=1E9 THEN 40010
102 PRINT INDEX(1)
110 GOTO 100
40010 PRINT"40010":END
```

Fisure 2

```
 10 0F2E=              PTRGET=@7456    ; RETURNS PNTR TO VAR
 20 00C6=              CHRGOT=@00306   ; RTS WITH CHAR AT (TXTPTR,+1)
 30 0E13=              COMCHK=@07023   ; CHECK IF (TXTPTR,+1)=","
 40 1B44=              FLOATC=@15504   ; FLATS BIN IN MACHO,MACMHO
 50 0E1E=              SNERR:=@7036    ; SNERR OUTPUT ROUTINE
 60 10D0=              FCERR:=@10320   ; FC ERR OUT PUT ROUTINE
 70 0094=              VARPNT=@00224   ; PNTS TO VAR AFTER PTRGET
 80 0092=              VARNAM=@00222   ; VAR NAME AFTER PTRGET
 90 00AE=              FACEXP=@00256   ; FP EXP
100 00AF=              FACHO =@00257   ; FP MSB
110 00B0=              FACMHO=@00260   ; FP MID MSB
120 00B2=              FACLO:=@00262   ; FP LSB
130                    ;
140 BF00          *=$BF00
200 BF00 F024          BEQ SN          ; EOL IS SNERR
210 BF02 202E0F        JSR PTRGET      ; GET PNTR TO VAR1
220 BF05 20130E        JSR COMCHK      ; NEXT CHAR MUST BE COMMA
230 BF08 F01C          BEQ SN          ; EOL IS SN ERROR
240 BF0A 20DEBF        JSR GETTYP      ; GET VAR TYPE
250 BF0D C903          CMP #03         ; IS STRING?
260 BF0F D07D          BNE FC:ERR      ; IF NOT, ERROR
270 BF11 A000          LDY #$00        ; CLEAR Y
280 BF13 B194          LDA (VARPNT),Y  ; GET V1 LEN
290 BF15 8DF8BF        STA V1LEN       ; STUFF IT
300 BF18 C8            INY
310 BF19 B194          LDA (VARPNT),Y  ; LOW BYTE OF ADDR
320 BF1B 48            PHA             ; PUSH IT ON STACK
330 BF1C C8            INY
340 BF1D B194          LDA (VARPNT),Y  ; HI BYTE OF ADDR
350 BF1F 48            PHA             ; PUSH IT ON STACK
360 BF20 202E0F        JSR PTRGET      ; GET PNTR TO VAR2
370 BF23 20130E        JSR COMCHK      ; NEXT CHAR MUST BE COMMA
380 BF26 F063     SN   BEQ SN:ERR      ; EOL IS SNERR
390 BF28 20DEBF        JSR GETTYP      ; MUST BE STRING
400 BF2B C903          CMP #03         ; IS STRING?
410 BF2D D05F          BNE FC:ERR      ; IF NOT ERROR
420 BF2F A000          LDY #$00        ; CLEAR Y
430 BF31 B194          LDA (VARPNT),Y  ; GET V2 LENGTH
440 BF33 8DF9BF        STA V2LEN       ; STUFF IT
450 BF36 C8            INY
460 BF37 B194          LDA (VARPNT),Y  ; LOW BYTE OF ADDR
470 BF39 48            PHA             ; PUSH IT ON STACK
480 BF3A C8            INY
490 BF3B B194          LDA (VARPNT),Y  ; HI ADDR
500 BF3D 48            PHA             ; PUSH IT ON STACK
510 BF3E 202E0F        JSR PTRGET      ; GET PNTR TO VAR3
520 BF41 A594          LDA VARPNT      ; SAVE ADDR OF VAR3
530 BF43 8DF6BF        STA VAR3
540 BF46 A595          LDA VARPNT+1
550 BF48 8DF7BF        STA VAR3+1
560 BF4B 20DEBF        JSR GETTYP      ; CHECK IS FL PNT
570 BF4E C905          CMP #05         ; IS?
580 BF50 D03C          BNE FC:ERR
590 BF52 20C600        JSR CHRGOT      ; CHECK FOR EOL
600 BF55 D034          BNE SN:ERR      ; IF NOT, SNERR
610 BF57 68            PLA             ; PULL VAR2 ADDR
620 BF58 8593          STA VARNAM+1    ; USE VARNAM FOR VAR2 ADDR
630 BF5A 68            PLA
640 BF5B 8592          STA VARNAM
650 BF5D 68            PLA
660 BF5E 8595          STA VARPNT+1    ; USE VARPNT FOR VAR1 ADDR
670 BF60 68            PLA
680 BF61 8594          STA VARPNT
690                    ;
700                    ; CHECK STRING FOR TEMPLATE
710                    ;
```

CONTINUED ON PAGE 9

by: Robert Camner

FIGURE 2 CONTINUED FROM PAGE 8

```
711 BF63 ADF9BF          LDA V2LEN       ; CHECK IF OBJ STR TOO SHORT
712 BF66 CDF8BF          CMP V1LEN
713 BF69 B01B            BCS NTF
715 BF6B A200            LDX #$00        ; RESET SEARCH START POINTER
716 BF6D ECF8BF          CPX V1LEN
717 BF70 F014            BEQ NTF
720 BF72 A000     NXTPOS LDY #$00        ; CLEAR TEMPLATE POINTER
730 BF74 B192            LDA (VARNAM),Y ; GET FIRST TEMPLATE POINTER
732 BF76 8EFCBF          STX TEMP
735 BF79 ACFCBF          LDY TEMP        ; GET POINTER TO Y
740 BF7C D194            CMP (VARPNT),Y ; MATCH WITH STRING?
750 BF7E F011            BEQ CHECK       ; IF SO, CHECK REST OF TEM
760 BF80 E8       NXTSTR INX
770 BF81 ECF8BF          CPX V1LEN       ; DONE WITH STRING?
780 BF84 D0EC            BNE NXTPOS      ; NO, SO GO ON
790 BF86 A2FF     NTF    LDX #$FF        ; SET (NOT FOUND) INDEX
800 BF88 4CB2BF          JMP FOUND
801 BF8B 4C1E0E   SN:ERR JMP SNERR:
802 BF8E 4CD010   FC:ERR JMP FCERR:
810 BF91 8EFABF    CHECK STX TEMPTR
820 BF94 A900            LDA #$00
830 BF96 8DFBBF          STA CHKPTR
840 BF99 EEFABF   CHKLP  INC TEMPTR      ; INC TEMPORARY POINTER
850 BF9C EEFBBF          INC CHKPTR      ; INCREASE TEMPLATE POIN
860 BF9F ACFBBF          LDY CHKPTR
870 BFA2 CCF9BF          CPY V2LEN       ; DONE WITH SEARCH
880 BFA5 F00B            BEQ FOUND
890 BFA7 B192            LDA (VARNAM),Y ; LOAD TEMPLATE BYTE
900 BFA9 ACFABF          LDY TEMPTR
910 BFAC D194            CMP (VARPNT),Y ; COMPARE WITH STRING
920 BFAE D0D0            BNE NXTSTR      ; IF NOT,CHECK NEXT STRING
930 BFB0 F0E7            BEQ CHKLP       ; IF MATCH CHECK NEXT CHAR
940                      ;
950                      ; X CONTAINS POSITION TO TRANSFER TO VAR3
960                      ;
970 BFB2 ADF6BF   FOUND  LDA VAR3        ; GET VAR3 ADDR TO VARPNT
980 BFB5 8594            STA VARPNT
990 BFB7 ADF7BF          LDA VAR3+1
1000 BFBA 8595            STA VARPNT+1
1010 BFBC A900            LDA #$00
1020 BFBE 85AF            STA FACHO       ; HIGH BYTE
1025 BFC0 E8              INX             ; SET CORRECT LENGTH
1030 BFC1 8A              TXA             ; TRANSFER LENGTH TO ACC
1040 BFC2 85B0            STA FACMHO      ; LOW BYTE
1050 BFC4 A290            LDX #16+@200    ; SET UP EXPONENT
1060 BFC6 38              SEC             ; SO BCS TO NORMAL
1070 BFC7 20441B          JSR FLOATC      ; BIN TO FLT PT
1080 BFCA A5AF            LDA FACHO       ; RESET SIGN BIT
1090 BFCC 297F            AND #%01111111  ; B7 IS SIGN BIT
1100 BFCE 85AF            STA FACHO       ; STORE
1110 BFD0 A000            LDY #$00
1120 BFD2 A205            LDX #$05        ; 5 BYTES TO MOVE
1130 BFD4 B9AE00  MOVNM1  LDA FACEXP,Y    ; GET A BYTE
1140 BFD7 9194            STA (VARPNT),Y  ; STUFF INTO FP DESCRIPTOR
1150 BFD9 C8              INY             ; NEXT
1160 BFDA CA              DEX             ; LOOP COUNTER
1170 BFDB D0F7            BNE MOVNM1      ; LOOP
1180 BFDD 60              RTS             ; RETURN TO BASIC
1190                     ;
1230                     ;
1240 BFDE A905    GETTYP LDA #$05         ; DEFAULT IS FP
1250 BFE0 2492           BIT VARNAM      ;  + = FP OR $
1260 BFE2 1008           BPL TSTSTR      ; CHECK IF $ OR FP
1265 BFE4 2493           BIT VARNAM+1    ; INT HAS TWO BMI'S
1270 BFE6 100B           BPL FCERR       ; MUST BE USER DEFINED VAR
1280 BFE8 A902           LDA #$02        ; INT
1290 BFEA D006           BNE TYPRTS
1300 BFEC 2493    TSTSTR BIT VARNAM+1
1310 BFEE 1002           BPL TYPRTS      ; ITS FP
1320 BFF0 A903           LDA #$03        ; IT IS A STRING
1330 BFF2 60      TYPRTS RTS             ; RETURN FROM GETTYP SUB
1340                     ;
1350 BFF3 4CD010  FCERR  JMP FCERR:
1360 BFF6 0000    VAR3   .DBYTE 00
1370 BFF8 00      V1LEN  .BYTE 00
1380 BFF9 00      V2LEN  .BYTE 00
1390 BFFA 00      TEMPTR .BYTE 00
1400 BFFB 00      CHKPTR .BYTE 00
1410 BFFC 00      TEMP   .BYTE 00
```

continued

Fisure 3

```
 10 0F2E=              PTRGET=@7456     ; RETURNS PNTR TO OF VAR
 30 00C6=              CHRGOT=@00306    ; RTS WITH CHAR AT
                                          (TXTPTR,+1)
 50 0E1E=              SNERR:=@7036     ; SNERR OUTPUT ROUTINE
 60 10D0=              FCERR:=@10320    ; FC ERR OUT PUT ROUTINE
 70 0094=              VARPNT=@00224    ; PNTS TO VAR AFTER
                                          PTRGET
 80 0092=              VARNAM=@00222    ; VAR NAME AFTER PTRGET
130                    ;
140 BE00               *=$BE00
150                    ;
200 BE00 F054    UPS$   BEQ SN:ERR       ; END OF LINE IS SNERR
210 BE02 202E0F         JSR PTRGET       ; GET PNTR TO VAR1
220 BE05 20C600         JSR CHRGOT       ; NEXT CHAR MUST BE EOL
230 BE08 D04C           BNE SN:ERR       ; IF NOT SNERR
240 BE0A 2040BE         JSR GETTYP       ; GET VAR TYPE
250 BE0D C903           CMP #03          ; IS STRING?
260 BE0F D048           BNE FC:ERR       ; IF NOT, ERROR
270 BE11 A000           LDY #$00         ; CLEAR Y
280 BE13 B194           LDA (VARPNT),Y   ; GET V1 LEN
290 BE15 8D55BE         STA V1LEN        ; STUFF IT
300 BE18 C8             INY
310 BE19 B194           LDA (VARPNT),Y   ; LOW BYTE OF ADDR
320 BE1B 48             PHA              ; PUSH IT ON STACK
330 BE1C C8             INY
340 BE1D B194           LDA (VARPNT),Y   ; HI BYTE OF ADDR
350 BE1F 48             PHA              ; PUSH IT ON STACK
360 BE20 68             PLA              ; PULL VAR ADDR
370 BE21 8595           STA VARPNT+1     ; STORE LOW ADDR
380 BE23 68             PLA
390 BE24 8594           STA VARPNT
405 BE26 A000           LDY #$00         ; CLEAR Y
410 BE28 B194    NXTCAR LDA (VARPNT),Y   ; GET FIRST CHAR
420 BE2A C961           CMP #'a          ; CHECKIF BELOW a
430 BE2C 900B           BCC NEXCAR
440 BE2E C97A           CMP #'z          ; CHECK IF ABOVE z
450 BE30 F002           BEQ CHANGE
460 BE32 B005           BCS NEXCAR       ; NO CHANGE IF ABOVE z
470 BE34 18      CHANGE CLC              ; CLEAR CARRY
480 BE35 E91F           SBC #31          ; SHIFT TO UPPER CASE
490 BE37 9194           STA (VARPNT),Y   ; STUFF CHAR
500 BE39 C8      NEXCAR INY
510 BE3A CC55BE         CPY V1LEN
520 BE3D 90E9           BCC NXTCAR
530 BE3F 60             RTS
1240 BE40 A905   GETTYP LDA #$05         ; DEFAULT IS FP
1250 BE42 2492          BIT VARNAM       ; + = FP OR $
1260 BE44 1008          BPL TSTSTR       ; CHECK IF $ OR FP
1265 BE46 2493          BIT VARNAM+1     ; INT HAS TWO BMI'S
1270 BE48 100F          BPL FC:ERR       ; MUST BE USER DEF VAR
1280 BE4A A902          LDA #$02         ; INT
1290 BE4C D006          BNE TYPRTS
1300 BE4E 2493   TSTSTR BIT VARNAM+1
1310 BE50 1002          BPL TYPRTS       ; ITS FP
1320 BE52 A903          LDA #$03         ; IT IS A STRING
1330 BE54 60     TYPRTS RTS              ; RETURN FROM GETTYP SUB
1340 BE55 00     V1LEN  .BYTE 00
1350 BE56 4C1E0E SN:ERR JMP SNERR:
1360 BE59 4CD010 FC:ERR JMP FCERR:
```

Figure 4

```
10 REM To enable SUB$ and UPS$
20 POKE 133,190:REM lower high memory
30 :
32 REM poke in SUB$ code
34 :
40 FOR I=48896 TO 48896+255
42 READ X
44 POKE I,X
46 NEXT I
50 :
52 REM poke in UPS$ code
54 :
60 FOR I=48640 TO 48640+91
62 READ X
64 POKE I,X
66 NEXT I
70 :
100 REM poke in SUB$  into reserved word list
110 POKE 8982,ASC("S")
120 POKE 8983,ASC("U")
```

CONTINUED

SEMAPHORE CHECKING, FILE SIZE & FILE ADDRESS 65U V1.42, LEVEL 3.

by: Colin Law
P.O. Box 3819
Auckland, New Zealand

When will we ever get a manual that we can rely on! I decided to look into semaphore checking and wrote a routine to set a batch of semaphores, check the range 0 to 210, and then clear those that were set. I set the limit at 210 because that's the limit on my Level 3.

Listing 1: I used the code from page 16 of the 65U Time Share Reference Manual V1.42 but got SN error in 120 and found that the parentheses didn't match up: one opening and two closing. After trying three different places for the missing parenthesis AND after discovering that the stray X was meant to be SM, I gave up and found the routine given on page 17-A of the Programmer's Reference Guide...

Listing 2: same problem again: SN error in 110 and AGAIN the parentheses don't match up! That one was solved by deleting one of them and at last my routine began to produce results. BUT after a few runs I began to suspect that there was still a bug! Sure enough, every time I started my set of 16 with 9,17,25,33 (i.e. n*8 + 1) the routine told me that the semaphore one lower had been set. Since I was the only user, I knew that no other semaphores were being set and indeed the extras weren't set because attempts to clear them came up with FC error...

Listing 3: That set me onto really looking at these wonderful (?) routines produced by the M/A-COM OSI wizards and I soon had my routine working properly. The problem is an old one when you get to it: if you have a FOR/NEXT loop which loops 1 to N then you shouldn't enter it when N=0 because it won't know that N=0 until it has been through the loop and come to the NEXT! I thought that everyone, even OSI knew that one. The faulty loop is FOR Z1=... in line 110 and Listing 1 didn't work (even after sorting out parentheses and odd X) because it had INT(SM/8) instead of INT(SM/8)*8. Listing 2 worked to some extent but when SM was an exact multiple of 8 the loop still went round once and Z became 2 instead of remaining at 1. i.e. the sequence

CONTINUED ON PAGE 12

FIGURE 4 CONT.

```
130 POKE 8984,ASC("B")
140 POKE 8985,ASC("$")+128
150 POKE 8716,255:POKE 8717,190:REM SUB$ dispatch address
160 REM poke in UPS$ into reserved word list
170 POKE 8970,ASC("U")
180 POKE 8971,ASC("P")
190 POKE 8972,ASC("S")
200 POKE 8973,ASC("$")+128
210 POKE 8710,255:POKE 8711,189:REM UPS$ dispatch address
220 :
230 NEW
240 :
990 REM machine code for SUB$
991 :
1000 DATA240,36,32,46,15,32,19,14,240,28,32,222,191,201,3,208
1010 DATA125,160,0,177,148,141,248,191,200,177,148,72,200,177,148,72
1020 DATA32,46,15,32,19,14,240,99,32,222,191,201,3,208,95,160
1030 DATA0,177,148,141,249,191,200,177,148,72,200,177,148,72,32,46
1040 DATA15,165,148,141,246,191,165,149,141,247,191,32,222,191,201,5
1050 DATA208,60,32,198,0,208,52,104,133,147,104,133,146,104,133,149
1060 DATA104,133,148,173,249,191,205,248,191,176,27,162,0,236,248
1070 DATA191
1080 DATA240,20,160,0,177,146,142,252,191,172,252,191,209,148,240,17
1090 DATA232,236,248,191,208,236,162,255,76,178,191,76,30,14,76,208
1100 DATA16,142,250,191,169,0,141,251,191,238,250,191,238,251,191
1110 DATA172
1120 DATA251,191,204,249,191,240,11,177,146,172,250,191,209,148,208
1130 DATA208
1140 DATA240,231,173,246,191,133,148,173,247,191,133,149,169,0,133
1150 DATA175
1160 DATA232,138,133,176,162,144,56,32,68,27,165,175,41,127,133,175
1170 DATA160,0,162,5,185,174,0,145,148,200,202,208,247,96,169,5
1180 DATA36,146,16,8,36,147,16,11,169,2,208,6,36,147,16,2
1190 DATA169,3,96,76,208,16,0,0,0,0,0,0,0,252,3
1900 :
1910 REM machine code for UPS$
1920 :
2000 DATA240,84,32,46,15,32,198,0,208,76,32,64,190,201,3,208
2010 DATA72,160,0,177,148,141,85,190,200,177,148,72,200,177,148,72
2020 DATA104,133,149,104,133,148,160,0,177,148,201,97,144,11,201,122
2030 DATA240,2,176,5,24,233,31,145,148,200,204,85,190,144,233,96
2040 DATA169,5,36,146,16,8,36,147,16,15,169,2,208,6,36,147
2050 DATA16,2,169,3,96,0,76,30,14,76,208,16
```

Continued

Fisure 5

```
LISTING FROM TERMINAL 0

10 REM Demonstrates SUB$, UPS$
20 A$="ABCDE"
30 B$="CD"
40 C$="aBcDe12"
100 SUB$ A$,B$,Q
110 PRINT Q:REM Q=3(B$ is substring of A$)
120 SUB$ A$,C$,Q
130 PRINT Q:REM Q=0 (C$ is not a substring of A$)
150 UPS$ C$
160 PRINT C$:REM C$="ABCDE12"
200 REM
210 REM syntax
220 REM SUB$ string1,string2,float-point-var
230 REM checks if string2 is a substring of string1
240 REM if so, returns starting location in float-point-var
250 REM if not, returns 0 in float-point-var
260 REM UPS$ string
270 REM converts all lower case characters in 'string' to upper
                                                        case
```

★ ★ ★

Continued from page 10

of Z was 2,2,4,8,16,32,64,128 instead of 1,2,4,8,16,32,64, 128. The first 2 meant that the location 55333 was being examined for semaphore SM+1 but was reporting the result as status of SM. I wonder whether anyone else has tried to use these recommended routines and had spurious results. Surely it must be a record for a Manual to refer to the same routine in two places and make different errors in each listing!

A further error occurs on page 13-A of the Programmer's Reference Guide. There is a rehashed version of disk address and file size routines that appeared in PEEK(65) during 1980. The trouble is that M/A-COM OSI couldn't copy it properly. The routine shows:

```
OPEN F$,P$,CH : Z=9898+CH*8
ADR=256*(PEEK(Z+1))+256*(PEEK
    (Z+2))+256*(PEEK(Z+3))
LN=256*(PEEK(Z+4))+256*(PEEK
   (Z+5))+256*(PEEK(Z+6))
```

I suspect the problem came up because of the ^ function being unavailable with V1.42 extensions. Again it's a problem of parentheses and you do get sensible answers occasionally with the routine as printed. However, here are two corrected versions, take your choice:

```
OPEN F$,P$,CH : Z=9898+CH*8
     :T= 256
ADR = T * PEEK(Z+1) + T * T *
    PEEK(Z+2) + T * T * T *
    PEEK(Z+3)
LN  = T * PEEK(Z+4) + T * T *
     PEEK(Z+5) + T *
    T * T *PEEK(Z+6)
```

```
OPEN F$,P$,CH : Z=9898+CH*8 :
    T = 256
ADR = T* (  PEEK(Z+1) + T*
    ( PEEK(Z+2) + T*PEEK
    (Z+3) )  )
LN  = T* (  PEEK(Z+4) + T*
    ( PEEK(Z+5) + T*PEEK
    (Z+6) )  )
```

LISTING 1

```
100 FOR SM = 0 TO 210
110 Z = 1 : FOR Z1 = 1 TO
    SM-INT(SM/8) : Z = Z *
    2 : NEXT Z1
120 A = 1 : IF PEEK (55333 +
    X/8) AND Z) THEN A = 0
130 IF A THEN PRINT SM;
140 NEXT SM : PRINT
150 :
```

LISTING 2

```
100 FOR SM = 0 TO 210
110 Z = 1 : FOR Z1 = 1 TO
    SM-INT(SM/8)*8) : Z = Z *
    2 : NEXT Z1
120 A = 1 : IF (PEEK (55333 +
    (SM/8)) AND Z) THEN A = 0
130 IF A THEN PRINT SM;
140 NEXT SM : PRINT
150 :
```

LISTING 3
(This one works)

```
10 REM   SEMAPHORE CHECKING
20 REM   SETS 16 SEMAPHORES
30 REM THEN CHECKS ALL
   SEMAPHORES 0 TO 210
40 REM   THEN CLEARS THE 16
   THAT WERE SET
50 :
60 INPUT "START"; ST : IF
   ST+15 >210 THEN PRINT "!!!"
   : GOTO 60
70 FOR I = ST TO ST + 15 :
   WAIT FOR I
80 PRINT I; : NEXT I : PRINT
90 :
100 FOR SM = 0 TO 210 : T =
    (SM - INT (SM/8) * 8)
110 Z = 1 : IF T THEN FOR Z1 =
    1 TO T : Z = Z * 2 : NEXT
    Z1
120 A = 1 : IF (PEEK (55333 +
    INT(SM/8)) AND Z) THEN
    A = 0
130 IF A THEN PRINT SM;
140 NEXT SM : PRINT
150 :
200 FOR I = ST TO ST + 15 :
    PRINT I; : WAIT CLEAR I :
    NEXT
210 PRINT "CLEARED" : PRINT
220 GOTO 60
```

Colin Law
New Zealand

★

UPGRADE YOUR MONITOR REPLACE THREE 1702 PROMS WITH A 2716 EPROM IN AN OSI C2-4P

by: Roger E. Miller
449 Falstaff Road
Rochester, NY. 14609

A procedure is described whereby the C2-4P monitor is replaced by one of several enchanced versions available for C4P computers.

Three 1702A PROMS are replaced by an Intel 2716 EPROM.

The OSI C2-4P contains a model 500 single board computer with the operating system contained in three 1702A PROMS. The monitor is sufficient to operate a cassette system, but there are numerous enhanced versions on the market. These are generally intended for easy replacement into the newer C4P computers, however, with a little effort the benefits can also be made available to those with older machines.

Before attempting this retrofit you must be sure you are willing to cut a foil on the 500 CPU board and that the factory PROM configuration is the same as that to which this description applies. Figure one is extracted from the OSI model 500 PROM Implementation schematic, which states that there are three available configurations. Figure two is the CPU board component layout and will help in this determination as well as locating devices during the actual retrofit. The described change is for a three PROM decoding scheme where jumpers J3 and J4 are in place, switch SW-1 is missing, and point K1 is a dead end trace. Monitor replacement from the other two configurations is possible, but not discussed here. Last, this

change applies to a new monitor organized the same way as the OSI SYNMON. There are three other possible system monitors in use by OSI. They all contain code sufficient for operating a cassette system, but the page select signals are not necessarily the same and are not given in this description, (cf. ref 3).

This procedure requires building and installing a narrow wirewrap board next to the Model 542 keyboard, transferring signals from the CPU to the patch board, and altering one of the three page decoding signals. First, the wirewrap board contains a 74LS148 priority encoder and 16 pin socket, an Intel 2716 EPROM and 24 pin socket, a 24 pin socket to accept the plug on a 24 wire ribbon cable, and wirewrap pins for three more signal transfer lines. In addition, it has a 0.1 microfarad despiking capacitor and three 4700 ohm 1/4 watt pullup resistors. The encoder will work without the pullups, but they and the capacitor are good practice. The layout of the components is arbitrary depending on available board material and whether it is decided to squeeze it in inside the case next to the keyboard or be neat and locate the board outside the case. Be careful of long unshielded cables; they may be a source of noise problems.

Next, some means to relocate address, data, and control signals must be devised. I chose a 24 wire ribbon cable to transfer all signals from the A4 socket to the protoboard. This cable has a plug/PROM carrier at each end to facilitate future removal of boards. It provides address lines (A0 - A7), data lines (D0-D7), page five select (FDXX), and the five volt power line. Three additional signals are needed. It was surprising to learn that Vbb for a 1702A is negative nine volts rather than d.c. ground. Therefore, it is necessary to locate a place for soldering a wirewrap pin into the ground foil of the CPU board. The ground trace originates at pins B27 and B28 on the backplane and there are several places where the wire may be attached. Likewise, the page four (FEXX) and page three (FFXX) select signals may be traced from pin 14 of sockets A5 and A6 respectively, and wire or pins soldered in. The three wires were routed through the plated-through holes, provided for a PROM select switch (SW-1) as a quasi-strain relief. Again use a connector for ease of future disassembly. The three wires are soldered to the top of wirewrap pins on the protoboard.

Figure three provides pinouts for transferring A0-A7, D0-D7, and power from the cables to the new EPROM.

Figure four gives the wiring details for the priority encoder and control signals required by an Intel 2716 EPROM.

This completes the first try at installing an improved monitor. However, it does not work! After much head-scratching, the 74LS148 encoder was breadboarded and it was discovered that the 02*VMA signal used in 1702A decoding is not used for 2316B/2716 decoding in newer OSI machines. This signal causes the GS output of the encoder to be timed incorrectly with respect to its page select output signals. The chosen correction requires major carpentry on the CPU board. The 02*VMA is replaced with a logic high on pins 11 and 12 of device F6,a 7430 NAND gate. The 02*VMA originates at pin B42 on the backplane and the branch in question is used by F6 and by F5, for RAM buffer enable. Following this trace can be tricky so be careful in locating these three locations. Right in front of B42, on top, is the first through hole. The second is on a topside trace to pins 11 and 12 of F6. The last is also on top and on a short trace to pin 10 of F5. The foil is cut, on the bottom,

FIGURE 1: 5ØØ CPU 17Ø2A PROM DECODING

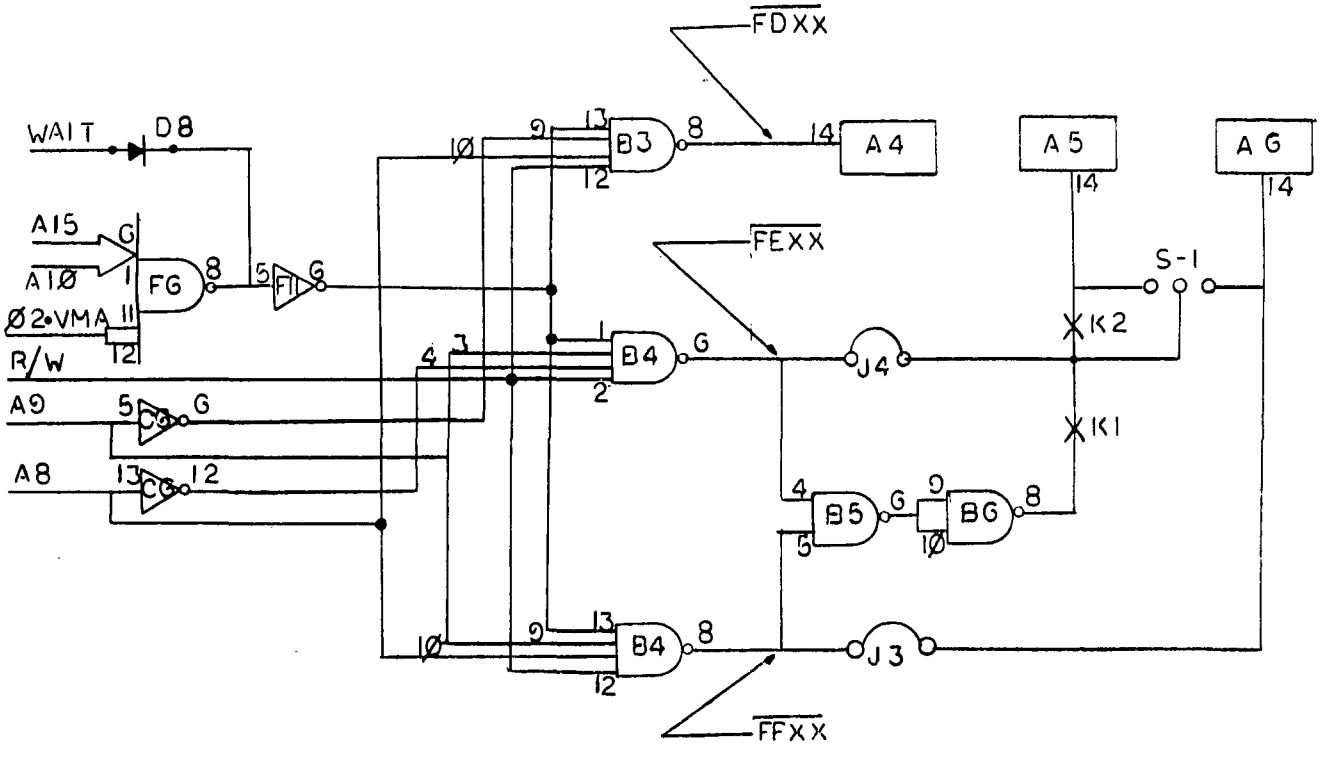


FIGURE 2: C.P.U. COMPONENT LAYOUT

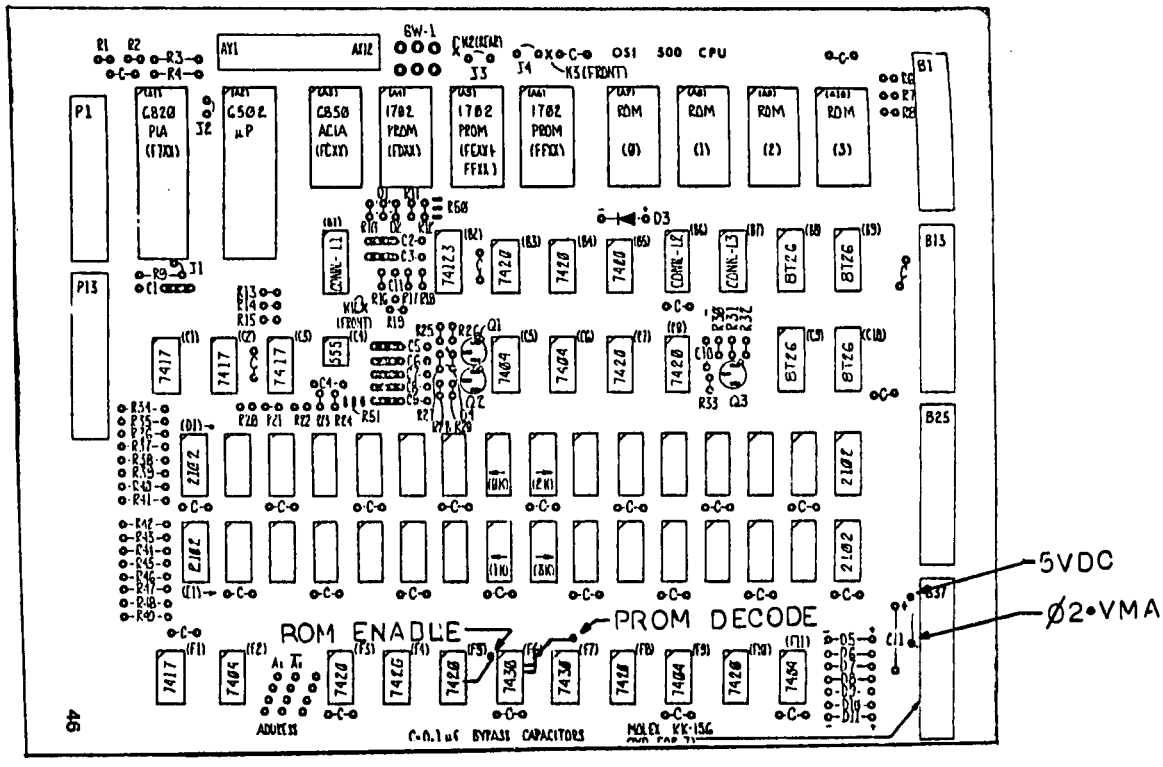
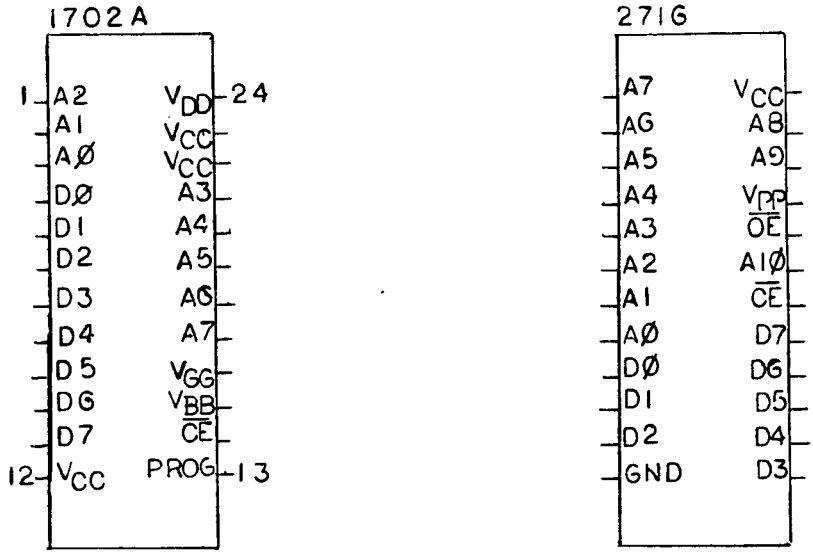


FIGURE 3: PINOUTS - 1702A AND INTEL 2716



just before the second location. A jumper from the first to the third point effectively carries 02*VMA around the PROM page decoding section. The circuit may work if pins 11 and 12 are left to float high. I chose to force them high via the five volt buss, without a limiting resistor. The five volt buss starts at B25 and B26 and by following, on top, there is another through hole in front of B38 where a logic high can be obtained. Another 4700 ohm resistor might be soldered in and wirewrap run from the free end to F6 pins 11 and 12. Now your new monitor will work. Enjoy the new features you have wanted for so long.

This article is intended to encourage those who are nervous about digging into their machines and learning what makes them tick. The retrofit is the result of two years of wishing for one of the great new systems available. Wishing did not make it happen; trying did, and it is a super feeling to decipher a small piece of these wonderful machines we all enjoy so much.

Here, the technical level only requires a willingness to carefully follow foil traces and make the required solder-in or wirewrap connections. An ohm-meter is extremely useful to insure

FIGURE 4: ALTERNATE DECODING FOR 2716 EPROM

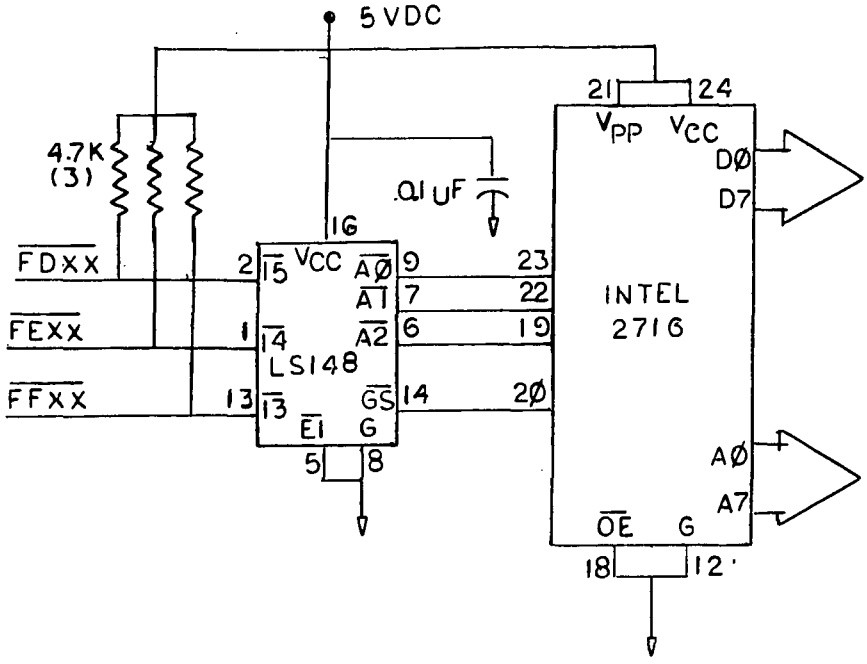
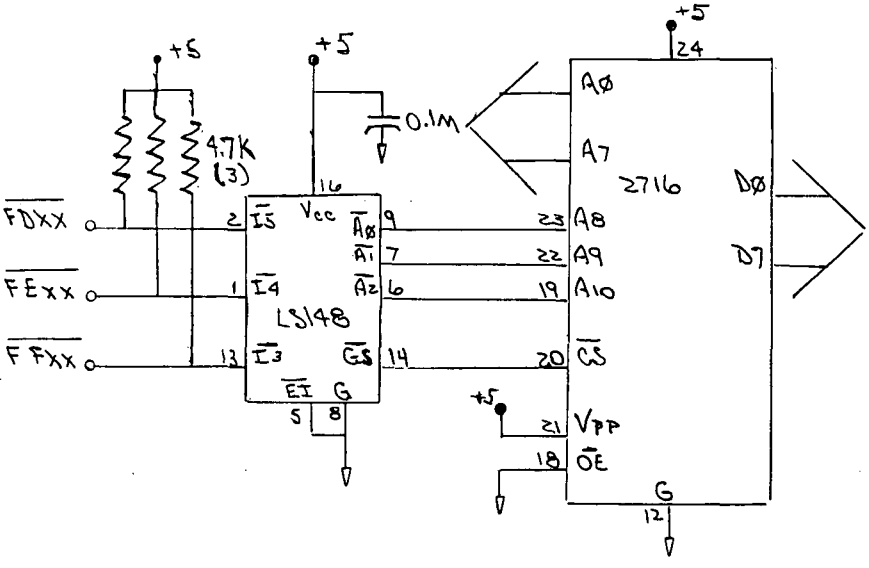


FIGURE 4: ALTERNATE DECODING FOR 2716 EPROM



NOTE: REPLACE (02*VMA) ON U6-11,12 WITH LOGIC HIGH

that the connections do go to the intended I.C. pins. The novice hardware person should be able to complete this improvement in 3 - 5 hours, once all materials are at hand. Two more references are given and may be useful. The S.S.J. contains a discussion of OSI ROM configurations and is quite helpful, once it is realized that OSI page numbers are in the opposite order of those given in TTL data manuals, for the 74LS148 priority encoder. Since many replacement monitors carry the floppy and/or hard disk bootstrap routines in pages one and two, further improvements to older cassette systems suddenly appear. For example, articles have been published which suggest switch selection of page numbers to maintain ROM BASIC when upgrading to a disk system.

Happy tinkering!

REFERENCES

1. OSI Model 500 Data Sheet, August 1977

2. OSI Model 502 Data Sheet, April 1978

3. OSI Small Systems Journal, Micro No. 27, August 1980

FIGURE 4 : ALTERNATE DECODING FOR 2716 EPROM

NOTE: REPLACE (02*VMA) ON U6-11,12 WITH LOGIC HIGH.

SINGLE SWITCH CONTROL

By: Leonard F. Watkins, Jr.
1044 N. Waco
Wichita, KS 67203

So you want a single switch to control and turn on your computer, monitor and other units with your computer. But the thought of having to construct one of those complex and fancy sensing circuits drive you out of your gourd? Do not despair. You can construct a simple unit using a minimum of parts and a simple-circuit that will do this just as well as the fancy circuits and for a lot less money and in a lot less time. However, this is not a training project and you will not learn anything new by constructing it but you will have the single switch control that you want and that is the purpose of this project. This is a tried and fully tested circuit and unit. I am using such a unit to control and turn on my C2-4PMF with my monitor. The unit switches both the computer and disk drive on when the monitor switch is turned on.

Anyone who can read a circuit diagram and is competent enough to go into the power supply of the monitor to obtain the low voltage to use in controlling the relay can complete this project without any difficulty. Note, ALWAYS make sure the power is disconnected from the monitor before starting modification to the power supply. The modifications to the power supply are simple, you merely add a connection to ground to go to the relay and a conection to the plus voltage to also go to the relay. If

MULTI-PROCESSING
with the Denver Board

The Denver Board (Model DB-1) is an assembled and tested terminal expansion circuit board for expanding terminal usage on any Ohio Scientific, Inc. (OSI*) Series C2 and C3 computer system. The DB-1 is designed to reduce terminal speed loss from 80 to 90 percent when two or more terminals are added to the computer. Each terminal is also provided with an additional 16K bytes of memory.

Each DB-1 comes with a full 90-day parts and labor warranty, and a factory repair/exchange program is also available should a DB-1 that is out of warranty ever need servicing.

FEATURES

- 64K Bytes Random Access Memory (RAM)
- One Programmable Read Only Memory (PROM) for BUS arbitration and interprocessor communications.
- Six light emitting diodes (LED's) for power, master BUS indicator, transmit and receive.
- Automatic system boot switch.
- Auxiliary BUS for expansion printer I/O circuit board.
- Four reset modes:
  Power-on reset.
  Master reset (front panel).
  Individual reset from terminal with BREAK key.
  Individual reset from DB-1 with pushbutton switch.
- Memory expansion capability of 4K bytes common memory using standard OSI memory expansion circuit board.

SOFTWARE

95 percent of existing OS-65u* software is compatible with the DB-1. An OSI operating system patch program is supplied on 8-inch floppy disk as required. The patch program is copied to the user disk that contains the OSI operating system; and when the computer is turned on, the patch program will automatically tie-in.

for more information call or write:

DBi, inc.

p.o. box 7276
denver, co 80207
(303) 364-6987

Dealer Inquires Invited

* OSI and OS-65u are trademarks of M/A-COM Office Systems, INC.

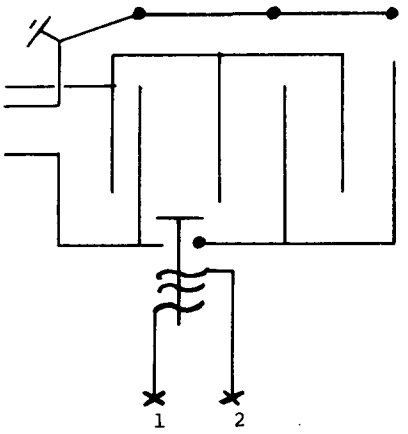
you like, and I did so, you can install a connector in these leads so that the unit can be completely separated from the monitor. I have not included this in the text or diagram because it, like so many other modifications, can be "hand rolled" by the maker to his/her own satisfaction. You will note that a diode has been placed in the positive lead to the relay. This is to stop-back EMF produced by the relay to enter the monitor - it is not absolutely necessary but is included anyway.

Construction is not critical, most parts can be substituted but I would suggest that only a grounded circuit be used using three wires. Any enclosure that you wish to use may be used that is sufficiently large enough to accommodate the number of outlets you wish to use and the relay that you are going to use. But if metal is used, then the enclosure should be grounded to the 3rd wire ground. The relay needs to have a coil voltage rating equal to the low voltage of the monitor and a current capacity great enough to handle all the load that will be placed on the outlets.

I used a plastic 3 switch box in the unit I constructed, which works quite well. I have not included any description on the construction of the unit since it is so simple that anyone who has enough experience to make the pick off in the low voltage section of the monitor will not need to be told how to construct such a simple project. However, one last reminder, always make certain that all power is off and disconnected before working on any of this project.

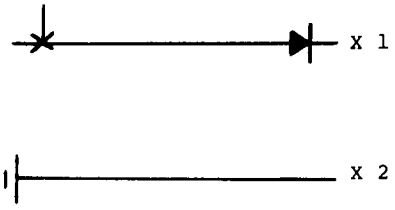
SCHEMATIC

MONITOR SECTION



LOW VOLTAGE

TO MONITOR



PARTS LIST

3 Grounded Sockets 117 V.A.C.
1 Diode See Text
1 Relay See Text
1 Enclosure See Text
1 3 Wire Power Lead w/Plug

CONSTRUCTION NOTE: As constructed in this unit the first socket is live all the time and the remaining two are controlled by the on/off switch of the monitor.

LETTERS

ED:

I recently added the RS-232 port to my SeriesI C1P and added a modem. I have accessed your CBBS without any problems and was excited about getting hooked up to CompuServe. So I got my account number and password and called the phone number. Well that's when the problems started.

The data they send shows up as a mixture of text and graphics characters. As an example, when they say USER ID it comes out on my screen as U R D. However, they seem to be getting my data accurately. Because when I send my user id, I get the next prompt P S R; which I assume is Password. Then when I send my password, I get a screen of characters and mainly graphics, which is impossible to decipher.

I called CompuServe and they checked out the phone line and the node which was operating correctly. So all they could tell me was that it was probably a hardware problem. Well, I figure if it was a hardware problem, I wouldn't be able to access the PEEK CBBS.

I am using a direct connect modem, so it isn't room noise. I am using the Modem program supplied with OSD3.3. CompuServe said my "terminal should be configured either 300 baud, even parity, 1 stop bit, word length 7 OR 300 baud, no parity, 1 stop bit, word length 8." I don't know enough machine language to determine if the OSD3.3 Modem program does this or not.

Could you please help me? Thank you very much.

Tim Lowe
Blacksburg, VA 24060

Tim:

Sounds like your problem is that Compuserve is sending 8-bit bytes of (256 possible values). Many terminals will only print 128 different characters, so if they get, say, a 193, they automatically subtract 128 and come out with 65, an "A". Your machine uses all 256 characters, the top 128 being the graphics characters.

The answer is to find where your program is getting the character from the line, which will be something like:

LDA XXXX

Where XXXX is the address of your input port. Then the next instruction must be:

AND 7F

which masks off the top bit, effectively substracting 128 if it is over 128.

Problem is, to do this, you need the source code to find where that part of your program is (there will be lots of LDA instructions) AND you need an assembler to add the AND 7F!

Can someone else help?

Al.

* * * * *

ED:

I thought my fellow PEEK (65) readers might appreciate the following:

Users of OS65D 3.2 who would like to improve handling of random files without having to patch the OS or upgrade to version 3.3 can do so with a simple BASIC subroutine.

A famous flaw of version 3.2 is that the system reads a track into the buffer with every DISK GET even if the track is already there. People have gone to great lengths to circumvent this, including rewriting portions of the OS or writing their own file handling routines in BASIC. The following short

routines solves the problem with a minimum of change to existing software. It uses the normal disk buffer and all I/O is to device #6 as usual. As a bonus, it allows records of any length.

```
600 TN=INT(R/16):IF TN=TL
    THEN 620
610 TL=TN: IF PEEK(9005) THEN
    DISK PUT
615 DISK GET,R:RETURN
620 RP=12926+(R-TN*16)*128
630 HI=INT(RP/256):LO=RP-HI
    *256
640 POKE 9133,HI:POKE 9132,
    LO:POKE 9156,HI:POKE
    9155,LO:RETURN
```

Line 600 calculates which track of the file the needed record is on. Note that this is not the disk track number, but the track of the file where the first track is number zero. We'll let the OS worry about where it actually is on the disk-- after all, that's its job! If this track number is other than the last one called, line 610 checks the buffer-dirty flag and performs the PUT if needed. In either case, line 615 then does the GET to call the new track in. As long as we're using standard size records the pointers will be set for us, so we're done.

However, if the new track is the same as the last one called, we know it is already in the buffer. All that's needed is to reset the device #6 I/O pointers to the start of the correct record. Line 620 calculates the record pointer, and 630 translates to high/low byte format for the actual pokes.

The numbers shown are for minifloppy systems. For 8" systems, change the buffer start location in line 610 to 12670 (or elsewhere if you have relocated your buffers). Also change from 16 to 24 records per track in lines 600 and 620.

For all systems, changing the records-per-track and the bytes-per-record (128 in line 620) can provide records of any length-- not just the powers of 2 allowed by the OS. If you try this, you must delete the RETURN at the end of line 615 so the custom record pointers will always be calculated. Also change the DISK GET,R to DISK GET,R*-INT(records-per-track/16) ... or /24 for 8" systems.

To prepare your program, first eliminate all DISK PUTS. The one in the subroutine will be called when needed. Then replace all DISK GET's with R=(record number):GOSUB 600.

After the file opening command, add a DISK GET,0 to insure that the buffer is loaded the first time. All INPUT's and PRINT's are to device #6 as usual. Close the file as recommended to force the final track write.

Implementation of this routine in a BASIC file sorting program reduced running time from 35 to 9 minutes, with much less wear to the disk and drive.

Steve Donachie
Miami, FL 33143

GREAT....

AL.

* * * * *

ED:

This note is a follow-up on my phone call concerning the need to determine how to transmit a BREAK via an RS-232C communications link to a remote computer (usually a large, mainframe machine). The problem seems as if it should not be a problem at all for the OSI machine. "Dumb" terminals send the BREAK easily and reliably, "smart" terminals of the OSI variety have troubles (at least for me).

I need the BREAK in working with mainframes. The prin-

ciple use is to interrupt on-going processing that can serve no useful purpose. The BREAK allows the "smart" terminal (or "dumb" terminal) operator to regain control of the mainframe and to redirect its activity. The seriousness of the problem is most apparent during long distance communications when a long listing or a long program execution is started and is found to be wrong or unwanted. Without being able to "BREAK", the terminal operator can only

. wait until completion of the listing or program execution, or

. hang up the telephone.

In the latter case, the operator must re-establish the channel; but in the "hanging-up," he may have lost valuable files and information. The frustration factor is great, resources have been spent with little or no return, and accounting information is possibly lost by the operator. The former method is also frustrating and obviously costly of clock time, operator time, communication resources, and computer resources.

According to the literature for the 6850 ACIA used on the 550 board, bits 5 and 6 of the control register may be set during the port's initialization to allow a BREAK to be transmitted. I haven't been able to make the right initialization; or, if I did it correctly, the BREAK is not possible in my case. This tends to say that a difficulty exists between the 6502 CPU and the 6850 ACIA. The difficulty may be with the software, the hardware, or both. The Operating Manual for the ACT-5A terminal by Micro-Term states that the BREAK causes a logical "0" to be sent for the duration of the time that the key is depressed.

Perhaps a reader of PEEK(65) will know the answer and will inform us all.

Frank M. Nelson
Bethesda, MD 20817

Mr. Nelson:

The answer to your question is to do the following:

Set the Control Register to 03

Set the Control Register to 96
+ normal init. code

Set the Control Register to 03

Set the Control Register to 17

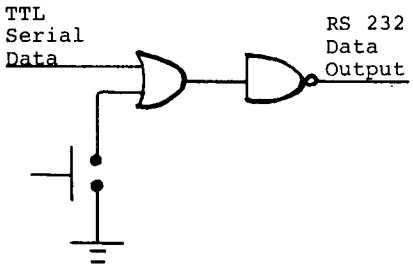
According to the Motorola Manual, setting Control Register bits 5 & 6 to ones will cause RTS to be low, disable transmitting interrupt and transmit a BREAK level on Transmit Data out.

What the above does is:

1. Reset the ACIA
2. Setup and Transmit Break
3. Reset
4. Restore to original condition.

I have used this method myself on another computer system and have found it to work well.

If you for some reason still can't get it to work, try adding some hardware, i.e.:



The Micro-Term terminal uses a similar setup to transmit a BREAK.

Brian

* * * * *

ED:

Today I added another feature to EDMAFL which could be applicable to the business world.

I have to enter into each master file record a band number. Now most of the time when banding a large number of birds in the field, we try to group the small birds with other small birds so that the band numbers are within the same series (or size), knowing that this will help us later with the paperwork. With that in mind, I felt there was absolutely no necessity for me to laboriously pump in the entire 9-digit band number for each and every record. To begin with, in this EDMAFL, and I guess that's very important, I am NOT using the Audit Trail (but even if I were, there's a way around this), so I have deleted the Audit Trail print lines that occur somewhere between 5500 and 5550. However, in order to help us along with this numbering problem, we'll let the software think that the audit trail is switched on, by line 126 which has to be changed from F6=K1 to F6=K2.

In my case, the band number is the very first field (not that this matters) so somewhere above line 1130 (which says: INPUT S$:GOSUB 5500) but AFTER line 1060 we put in some additional information:

```
....IF FDLB$(FPTR)="BAND
    NUMBER" THEN PRINT TAB(40)
    ;"LAST";AFC$(1)
```

Of course the field label depends on your own needs. The variable AFC$(1) will be explained later. Then AFTER the last of the print statements (that place the fields on the screen) we have the following:

```
....IF FDLB$(FPTR)="BAND
    NUMBER" THEN PRINT "NEXT
    NUMBER":GOTO 1420
```

(1420 in my case because I need some additional room which I haven't got anymore in this area because of other customization and this program is now so full (to its upper limit) that resequencing would not be wise; in fact, it would

probably prove destructive).

Then at 1420 I have the following:

```
1420 REM SAME BAND NUMBER + 1
     CODE
1421 PRINT:INPUT"Y/N";YN$
1422 IF YN$="N" THEN GOTO
     1130
1423 IF YN$="Y" THEN BNBR=VAL
     (AFC$(1))+1
1424 S$=RIGHT$(STR$(BNBR),9):
     GOTO 1142
```

Now in the audit trail area (lines 5500-5550) I have only these two lines:

```
5500 IFF6=K1ORS$="/" ORS$=""
     THEN GOTO 5550
5510 AFC$(1)=FC$(1)
5550 RETURN
```

So, what do we have here? When the first field in the data base gets presented for input, a prompt will appear which says NEXT NUMBER Y/N. If you reply N, then you get a question mark (in my case a > since I don't like those questions marks - you do this by placing POKE 2797,62 at the very beginning of the program), and this is now your prompt to fill in the input for that field (line 1130). The GOSUB in line 1130 brings you to the point where FC$(1) is changed to AFC$(1), and when the next record rolls around, at TAB(40) next to the input box for the first field, the contents of the first field in the previous record appear (on the very first record you do after entering DMS, this will obviously be blank). So let's suppose you have done one record and on the second record you answer 'Y' because you do want the next number in sequence. And that's exactly what you'll get!

One small footnote- (see line 1423 and 1424 above), you cannot write S$=VAL(AFC$(1))+1 because then you're mixing apples with oranges and in line 1424 you MUST use RIGHT$(STR$(BNBR),9) because if you don't use RIGHT$ then the entire contents is shifted one space to the right (I am not sure why) and you will get a notice that the length of the input exceeds the length of the field by one character.

For people using order numbers or invoice numbers (in business uses of OS DMS) this technique might prove quite useful.

Fred Schaeffer
Jamaica, NY 11435

Fred:

Nice work! Two comments.

1) Make a backup copy and RESEQ it. It shouldn't hurt, and will make future customization easier.

2) STR$(BNBR) inserts the extra space because BASIC stores numbers as signed values, but does not put in the "+" sign if the number is positive. The extra space is "holding room" for a minus sign in case the number is less than zero!

Al

* * * * *

ED:

Just a quick reply to the letter in the September issue from Fred Schaeffer regarding EDMAFL.

My first attempt at his problem with my EDMAFL (also DMS 9/79) produced exactly the same result... *** Disc error 130 in line 1220. However, when I amended line 126 to read F6=K1 again, the problem didn't go away! The only occurrences of F6 in the whole program are 126 F6=K1, 5500 IF F6=K1 ... That seems harmless enough and your thought that FP() and FPTR may be wrong didn't show any faults. The author used FPTR for current field number being edited and array FP(n) for field offsets. Then I thought carefully about what I had done - in using my original unamended DMS Nucleus disc I had needed a data file to experiment with. I used CITYSO which is one of the supplied demonstration files and I had accessed it with the password <PASS>. In fact the password should have been <ANAN> or <.> Line 1220 in EDMAFL is the first point at which the program does PRINT%1. Up to that time the program had only INPUT%1 from the file. When it tried to PRINT%1, it found that there was an access rights violation.

I suggest that Mr. Schaeffer check whether he has given the correct password. If that is correct then check whether there are any errors in the password verification in the EDMAFL being used.

I also note errors in the audit printer subroutine of my EDMAFL which produces rather nonsensical printout. I had not previously used the audit printer facility so I had

never noticed. The extra " is clearly wrong in line 5510 and it seems more appropriate in line 5520 to print the record number rather than contents of field 1. I don't know whether these errors were specific to my EDMAFL or not, but just in case anyone else wants them I have shown my changes in listing 1.

LISTING 1:

```
5510 PRINT#AD,"DMS EDITOR";"
     TAB(13);DT$;TAB(25);
5520 PRINT#AD,"REC #; ";FC$
     (K1);TAB(53);"FIELD: "
     ;FDLB$(FPTR);
```

Changed to:

```
5510 PRINT#AD,"DMS EDITOR"
     ; TAB(13);DT$;TAB(25);
5520 PRINT#AD,"REC #; ";RPTR;
     TAB(53);"FIELD: ";FDLB$
     (FPTR)
```

Colin Law
New Zealand

* * * * *

ED:

I own a C8P-DF video system with 48K and 8" floppies. I use primarily OS-65U V1.2, but also have 65D V3.2. I work for the time-sharing division of Control Data Corporation, and would like to use my computer to access our network.

Currently, I am using the MODEM driver program that came with my machine, although I have written Phil Lindquist for a copy of his STOS program. The problem I have is this. The break key on my keyboard is disabled in favor of the reset button on the CPU. However, now I cannot send an interrupt command to the Control Data System, since, for the life of me, I can't figure out how to generate a 'null' character (decimal '00') from the keyboard. As a result, when I'm printing a long program listing from the Control Data System, the only way I can break out of it is to hang up and re-dial, very frustrating! Can you help?

Can I rewire the break key to simulate the normal function of a dumb ASCII terminal? (Please be specific about any hardware modifications, since I am NOT a hardware guru!) Any suggestions would be greatly appreciated. I have been receiving your journal for over a year now and find it very informative. Keep up the good work!.

Gary L. Levine
Denver, CO 80231

Readers - who can help re transmitting ASCII 0?

Al.

* * * * *

ED:

I have a cassette system, (BASIC-in-ROM), and finally got around to modifying a TV monitor for direct video, (see BYTE, Feb. '79). When the screen fills with text, everything becomes unreadable!! A fellow OSI'er with a disk system, did the same TV mod and had the same problem. We discovered that reducing the screen memory by 1-3 lines eliminated the problem --- some kind of interference between maintaining video data and vertical retrace?? So far I've been unable to find anyone able to provide the fix for ROM-BASIC.

I believe it is contained in several MICRO 6502 articles (No. 36, pg 75, 46:67, 51:99) on cursor control and I/O, by K. V. Laurash and M. J. Keryan. However, we don't even have a nodding relationship and I am unable to decipher the fix. Bill Thompson at Cleveland Consumer Computers & Components was also unable to help. So any help you or the readers are willing to provide will be greatly appreciated.

Roger E. Miller
Rochester, NY 14609

* * * * *

ED:

"On a C1P-MF, is there any way to have a program stored on an OS65D disk but run under ROM BASIC as PICO-DOS does?"

This question was asked by Richard List in your June 1982 issue. The answer is both simple and advertised in PEEK (65). One of the many features of ROM-TERM by MICRO-INTERFACE, is a CTRL-B command that toggles you between disk and ROM BASIC. This allows me to keep a copy of our club's cassette programs on disk (65D V3.3) and list, copy, verify, edit or run them as needed.

I hope this has been helpful.

Paul Chidley
Shelburne, Ontario

* * * * *

ED:

I've been the owner of a C8P DF with 48K RAM and two 8" floppy disk drives since August 1980.

In early 1981, I heard about OSI's development of a 710 board with both the Z8000 & 68000 microprocessors residing on it. I have been awaiting further information on the 710 board; however, I have not heard any news about the 16-bit expansion board since.

Frank Chew
Hayward, CA 94544

Frank:

Development was apparently dropped about the time M/A COM purchased OSI.

Brian.

* * * * *

ED:

In response to David P. Redlawsk's letter in the October '82 issue. I had the same problem using my Heathkit H14 printer. The fix is very simple. Refer to the May '82 issue, page 17. The (cr) is generated by the subroutine between hex 3263 and hex 3268. Just change this subroutine to a JMP CRLF in the operating system. The address of the subroutine is hex 2D6A, so hex 3263 on becomes 4C 6A 2D. These three bytes can be changed on the disk or poked into place every time by BEXEC*.

The Heathkit printer doesn't print any graphic characters so I can't use them on my system.

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

* * * * *

ED:

The review of DQFLS's WP6502 version 1.3a by F.S. Schaeffer in the August issue has prompted me to write this.

I have tried their WP6502 version 1.3 on a 24K C4PMF under 65D (3.2) and have been rather disappointed; it seems to have quite a few bugs. In particular the 'insert' and the 'move' operations lose trailing text. Two exchanges (updates?) from New York have not improved the situation.

So I'm wondering if others

have experienced similar difficulties with their (C4P) 1.3 under 65D or am I doing something wrong?

W.E. Wilson
Richland, WA 99352

* * * * *

ED:

I have a Model 33 ASR Teletype Machine that I would like to use as a printer for my Ohio Scientific Computer, Challenger 1P.

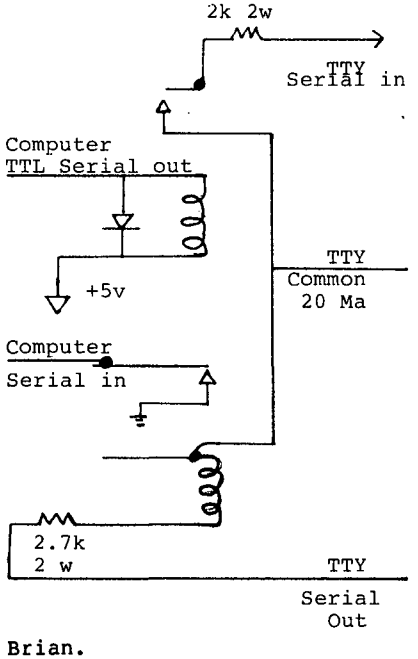
I wrote to Ohio Scientific about this and they recommended that I write to you for possible assistance.

Please advise.

Edward L. Radtke
Louisville, KY 40205

Mr. Radtke:

The model 33 ASR Teletype needs data sent to it at 110 baud, 20 MA current loop. The circuit below will provide the interface:



Brian.

* * * * *

ED:

I own an OSI C1P series II computer and a Radio Shack "Lineprinter VII" and this combination introduces a second linefeed by the printer therefore doublespacing each printed line.

Apparently, Radio Shack computers have an interpreter which doesn't send a linefeed and therefore the printer must provide one.

I would appreciate it if your readers could offer some help, or if you could refer me to someone who may be able to help me out. This printer performs well and I'd hate to exchange it because of this annoying problem. Thank you.

Ray Audette
Canada

Readers.....?

Al.

* * * * *

ED:

I am trying to locate a program for sending and transmitting Morse Code C.W. on the amateur radio bands. Does anyone know a source for such a program? I have the Superboard II with 20K of RAM.

Robert L. Dingle
Venice, FL 33595

* * * * *

ED:

RE your question after my letter published in PEEK(65) Vol. 3, No. 11, pages 24-25.

The purpose of my letter was to describe formatting a positive integer value after all arithmetic was completed. To print a negative amount would, of course, require a different routine. A test of <0 would be performed on the integer before deciding which routine to use.

Bruce Showalter
Abilene, TX 79601

* * * * *

FOREIGN SUBSCRIPTIONS

You've often heard that PEEK runs on a shoe string, but when foreign checks (even ones written in U.S. dollars) pass through our local banks they get hit with something like a $6.00 to $15.00 processing charge and that makes the shoe string more like a banana peel.

We could raise the rates, but we won't! We could get the banks to rescind their new rules, but we can't! The only fair thing for us to do is require that your checks be written in U.S. dollars on a U.S. bank or use an International Money Order. Your local bank or Post Office should be able to accommodate you.

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:

PEEK (65)

P. O. BOX 347

OWINGS MILLS, MD. 21117

SUBSCRIPTION RATES FOR 12 ISSUES (ONE YEAR), EFFECTIVE WITH THE JULY, 1981 ISSUE. ALL RATES QUOTED IN U. S. DOLLARS.

PLEASE FILL OUT AND RETURN WITH CHECK OR MONEY ORDER.

( ) $15.00 ENCLOSED. U. S. (MARYLAND RESIDENTS ADD 5% SALES TAX.)
( ) $23.00 ENCLOSED. CANADA AND MEXICO, 1ST CLASS, SURFACE.
( ) $35.00 ENCLOSED. SOUTH AND CENTRAL AMERICA. AIR MAIL.
( ) $35.00 ENCLOSED. EUROPE. AIR MAIL.
( ) $40.00 ENCLOSED. ALL OTHER. AIR MAIL.

NAME _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ STREET _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

CITY _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ STATE _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

ZIP CODE _ _ _ _ _ _ _ _ _ _ _ _ _ COUNTRY _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

PLEASE SEND THE FOLLOWING BACK ISSUES. I ENCLOSE:

( ) $2.00 EA. U. S. SURFACE. (MARYLAND RESIDENTS ADD 5% SALES TAX.)
( ) $2.50 EA. CANADA AND MEXICO. SURFACE.
( ) $3.00 EA. SOUTH AND CENTRAL AMERICA. SURFACE.
( ) $3.00 EA. EUROPE. SURFACE.
( ) $3.50 EA. ALL OTHER. SURFACE.

Vol 1. 1980

( )	JAN #1	( )	FEB #2	( )	MAR #3	( )	APR #4
( )	MAY #5	( )	JUN #6	( )	JUL #7	( )	AUG #8
( )	SEP #9	( )	OCT #10	( )	NOV #11	( )	DEC #12

Vol 2. 1981

( )	JAN #1	( )	FEB #2	( )	MAR #3	( )	APR #4
( )	MAY #5	( )	JUN #6	( )	JUL #7	( )	AUG #8
( )	SEP #9	( )	OCT #10	( )	NOV #11	( )	DEC #12

Vol 3. 1982

( )	JAN #1	( )	FEB #2	( )	MAR #3	( )	APR #4
( )	MAY #5	( )	JUN #6	( )	JUL #7	( )	AUG #8
( )	SEP #9	( )	OCT #10	( )	NOV #11	( )	DEC #12

INDEXES ARE INCLUDED IN THE JAN. & DEC. 1981 & DEC. 1982 ISSUES.