PEEK (65)

The Unofficial OSI Users Journal

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

$1.75

SEPTEMBER 1984
VOL.5, NO. 9

INSIDE

Column One

The word is, the nine OSI sales contest winners who went to Sweden had a good time. Good for them and for us.

Speaking of sales contest winners, ISOTRON now has some 12Ø dealers, with ten more soon to be approved and more in the pipeline. There is life in the old company yet.

One of the signs of life is the appearance of MD Partner, DDS Partner vertical market software packages. There will be more in the future, but for now ISOTRON wants to see how these first packages do in the market place.

When more new vertical market packages do come out, they will probably be in the main stream of business - don't expect to see a widget distributor package anytime soon.

Other signs of life:

KeyBase T - a new DBMS for the 3ØØ series machines - no details released as yet, but we are told it should be out by the time you read this;

a new DBMS for the 2ØØ series machine to follow shortly;

OS-65U is being overhauled to allow for new "hardware changes". Nobody will say what changes they have in mind...

a national ad campaign to start with the October issues of various magazines. We are really anxious to see the new ads;

another new machine - the 235 - it fits in a 2ØØ box but hidden in there is a new 515 board, plus a single-board hard disk controller. The machine is either a 2- or 4-user device and comes with 4 serial I/O ports, one parallel port, and one network port. A 2-board, 4-user computer with an 8" floppy disk and a mini-winnie. The system will sell for about the same price as a 23Ø, which amounts to about a $6ØØ price reduction as we calculate it.

Last month we reviewed The Data System. We failed to mention in the review that The Data System supports OS-DMS files (type 1Ø), except for some of the automatic features of progams like automatic file calculation, automatic keyfile update when a record is added. This will be a great advantage for people who have systems with existing OS-DMS files they want to preserve, and of course, the files can still be converted to the new type 3Ø.

While on the subject of software, it looks as though the free listings of software will probably spill over into the October issue, as it did last year. What this means to you is that you have a few more days to get the form in the July issue filled in and on its way to us. Once again, don't miss this unique PEEK opportunity that is, frankly, free advertising.

A question for business users: What does it take to entice you to utilize your magazine in the same fashion that hackers do? Long ago, they learned that there is much to be gained in the sharing of information. Write us and tell the community what you have, what you are doing with it, what you want to do with it and your problems.

This month's issue is in my lap - in its usual prepublication form, with great holes where full page ads will appear (not to mention the great hole on the front cover where Column One will appear if I ever finish writing it!). Looking over this issue, I am struck once more by the amount of detailed technical information it contains. What other computer magazine is 25% listings? Where else can you learn how to solve the IRQ problem (the State Department would like to see that one, I betcha)? From Maryland to Tasmania, OSI users stick together and help each other!

Which brings me to a painful subject. During my vacation, I have been thinking very hard about PEEK and have come to the conclusion that, due to my many other commitments, I cannot continue to edit PEEK. The owners of PEEK have assured me that, other than the style of this column, PEEK(65) will continue as usual. Eddie Gieske, who has filled in for me during my vacation, will be taking over my post here. It has been a labor of love and I have had a ball, and I thank all of you for your help and support.

al

65Ø2 ASSEMBLY LANGUAGE PROGRAMMING CLASS

Part III

By: Richard L. Trethewey
Systems Operator for the
OSI SIG on CompuServe

Let's take another look at the last program I presented in lesson 2.

```
1Ø       *=$4ØØØ
2Ø;
3Ø       LDY #$ØØ      ; CLEAR Y
4Ø       LDA #$2Ø      ; LOAD ACC. WITH A <SP>
5Ø P1    STA $D6ØØ,Y   ; SAVE ACC. AT $D6ØØ + Y
6Ø       INY           ; INCREMENT INDEX
7Ø       CPY #$ØØ      ; IS Y A ZERO YET?
8Ø       BNE P1        ; IF NOT, GO BACK TO P1
9Ø       RTS           ; YET IT IS, QUIT
```

If we were to add enough STA $DxØØ,Y instructions, we would have a program that would clear the entire screen. But the resulting program would be larger than it had to be. The need to keep code compact will become clearer later.

All right, so we have the essential structure of the program we want to write, but how do we change it? Well, first of all, we need to change the memory address in line 5Ø so that it starts at the top of the screen instead of near the bottom. So we'll change line 5Ø to read:

```
5Ø P1  STA $DØØØ,Y  ; SAVE ACC. AT $DØØØ + Y
```

Okay, so far so good, but now we need to make the changes that will allow the program to clear the rest of the screen since what we have now will only clear the top 8th of it. Here again it's divide and conquer as we attack the problem.

We know that the area of memory we want to clear is one contiguous block (i.e. there are no gaps in the range of memory addresses involved). Therefore, our task will involve repeating the same process for each page (block of 256 bytes) of memory in the screen memory. Aha! Sounds like a job for a loop! Again, since we'll be dealing with successively higher addresses in memory, for each pass through the loop we'll be clearing the next consecutively higher page of memory. The 65Ø2 provides an instruction that will help us here, which is the "INC" instruction. "INC" increments the contents of a memory location. By adding an INC command to bump the Most Significant Byte (MSB) of the memory address in line 5Ø, we can have same code executed for each page of memory in the screen.

Copyright © 1984 by **PEEK (65)** Inc. All Rights Reserved.

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

Mention of products by trade name in editorial material or advertisements contained herein in no way constitutes endorsements of the product or products by this magazine or the publisher.

Of course, as with any loop, we also have to include a test to see if the loop needs to be executed again. We'll be using the "CMP" instruction for this. "CMP" stands for "Compare the Accumulator", and we'll be testing the MSB in line 5Ø that we've been INCrementing to see if we've cleared all 8 pages of the screen. The resulting program is as follows:

```
1Ø       *=$4ØØØ
2Ø;
3Ø       LDY #$ØØ      ; CLEAR Y
4Ø       LDA #$2Ø      ; LOAD ACC. WITH A <SP>
5Ø P1    STA $DØØØ,Y   ; SAVE ACC. AT $DØØØ + Y
6Ø       INY           ; INCREMENT INDEX
7Ø       CPY #$ØØ      ; IS Y AT ZERO YET?
8Ø       BNE P1        ; IF NOT, GO BACK TO P1
9Ø       INC P1+2      ; YES! INCREMENT ADDRESS MSB
1ØØ      LDA P1+2      ; LOAD ACC. WITH NEW P1 MSB
11Ø      CMP #$D8      ; IS IT PAST SCREEN END?
12Ø      BEQ P2        ; IF YES, GO TO P2 (QUIT)
13Ø      LDA #$2Ø      ; IF NOT, LOAD A <SP> AGAIN
14Ø      JMP P1        ; AND RE-ENTER THE LOOP
15Ø;
16Ø P2 RTS             ; EXIT POINT
```

This program has several inefficiencies. I left them in so we could examine them. The first is that line 7Ø is unnecessary and could be eliminated entirely. Why? Because the previous instruction in line 6Ø, "INY", conditions the Z flag that is tested by the "BNE" instruction in line 8Ø, automatically when it is executed.

Next, when we loaded the accumulator with the contents of "P1+2" in line 1ØØ, we lost the <SP> character that we had been saving to the screen. So, each time the loop was executed, we had to restore the <SP> in line 13Ø. But even that was inefficient If we had labeled line 4Ø as "PØ", we could have eliminated lines 13Ø and 14Ø by changing line 12Ø to read "BNE PØ". This change would have the effect of altering the point at which we re-enter the loop when a page of screen memory is completely cleared. Of course, we also get the added bonus of more compact code again. So our program could be improved to look like:

```
1Ø       *=$4ØØØ
2Ø;
3Ø       LDY #$ØØ      ; CLEAR Y
4Ø PØ    LDA #$2Ø      ; LOAD ACC. WITH A <SP>
5Ø P1    STA $DØØØ,Y   ; SAVE ACC. AT $DØØØ + Y
6Ø       INY           ; BUMP INDEX
7Ø       BNE P1        ; IF Y <> Ø, THEN GO TO P1
8Ø       INC P1+2      ; INCREMENT MSB OF P1 ADDR
9Ø       LDA P1+2      ; FETCH NEW P1 ADDRESS MSB
1ØØ      CMP #$D8      ; PAST END OF SCREEN ?
11Ø      BNE PØ        ; IF NOT, PØ RESTORE <SP>!
12Ø      RTS           ; IF SO, WE'RE DONE (QUIT)
```

This program still has one fatal flaw. Did you spot it? The flaw is that since the program alters the contents of memory when it is executed, it can only be run once. If you tried to run it a second time, the address at P1 would start out at $D8ØØ and the program would try to set all memory locations from $D8ØØ on up, flopping over after $FFØØ to begin again at $ØØØØ and ultimately crashing when it gets up to $4ØØØ again.

The solution to this problem is to reset the address at P1 to $DØØØ when the program is done clearing the screen. The following code would accomplish this:

```
111  LDA #$DØ  ; LOAD ACC. WITH SCREEN TOP MSB
112  STA P1+2  ; RESTORE ORIGINAL ADDRESS AT P1
```

The technique presented here of altering code within a program as that program is executed is often called "self-modifying code". I'm using the term a bit loosely here since we're altering an address instead of an actual instruction code, but the object here is to demonstrate the effect of the technique.

In BASIC, if you refer to a variable in an equation without previously setting the value of the variable, BASIC assumes the value to be used is zero. We enjoy no such luxury in Assembly language programming. Each and every pointer must be initialized before they can be used. Sooner or later, you will violate this principle and your program will fail. It happens to everyone. When your programs lock-up, look at your pointers first. 'Nuff said!

As you can see, the "INC" instruction is very handy for dealing with consecutive memory addresses. The 65Ø2 also has a complimentary instruction called "DEC" which reverses the process by decrementing the contents of memory locations.

You may have thought that the references in the programs presented here to "P1+2" was a bit strange. Ordinarily, we would think that to find the most significant byte (MSB) of the address we would look at the left-most location. The reason we had to use "P1+2" is that the 65Ø2 uses this reversed order of the MSB and LSB for instructions in programs. Thus, when we refer to specific memory locations in terms of labels within our programs, we must take this situation into account. Don't forget that labels represent specific memory addresses for the assembler, even though they are looked at as reference points in text for us.

BRANCHING

In several of the previously presented programs, I have used the instructions "BEQ" and "BNE". The 65Ø2 has several such instructions called "branching instructions". Their purpose is much like the "IF" statement in BASIC. They test to see if a particular condition exists and if so, program control is sent to a specified destination. If you'll recall our previous discussion of the internal registers in the 65Ø2, one of the registers is called the status register or status byte. The value of each bit in the status register tells us something about what has occurred in a program and it is the status register that is tested with the branching instructions.

Note that only 7 bits in the status register are actually significant. One bit was not defined in the original 65Ø2s.

Again, each significant bit in the status register tells a story, and each story is different. These bits are also referred to as "flags" and are further referred to by a particular name. The status flags are:

N or Negative flag
If set (i.e. value = 1), it means that the result of the last operation that affected this flag set bit 7 of the byte manipulated (whether the accumulator, X or Y register, or a memory location makes no difference) indicating a negative result.

V or Overflow flag
If set, it means that the last math operation that affected this flag caused an overflow.

B or Break flag
If set, indicates that a BRK instruction was executed.

D or Decimal flag
If set, the accumulator will perform math operations in Binary Coded Decimal. If clear, normal math is performed.

I or Interrupt flag
If set, the 65Ø2 will process software interrupts.

Z or Zero flag
If set, it means that the result of the last operation that affected this flag was not zero. If clear, result was zero.

C or Carry flag
The value of this flag is used as a kind of "9th bit" in math operations so that multiple precision math can be performed.

The eight branching instructions test the condition of only four of these flags. The two-to-one ratio of instructions to flags provides the ability for a branch to be taken (i.e. the tested for condition will exist) for all possible states of the four flags. The branching instructions are:

BCC or Branch on Carry Clear
If the carry flag is zero, the branch is taken.

BCS or Branch on Carry Set
Branch is taken if Carry = 1.

BEQ or Branch on Equal
Branch is taken if Zero flag = Ø.

BNE or Branch on Not Equal
Branch is taken of Zero flag = 1.

BMI or Branch on MInus
Branch is taken if N flag = 1.

BPL or Branch on PLus
Branch is taken if N flag = Ø.

BVC or Branch on oVerflow Clear
Branch is taken if V flag = Ø.

BVS or Branch on oVerflow Set
Branch is taken if V flag = 1.

The branching instructions have a limitation that must be remembered when they are used in programs. They can only branch to a point 127 bytes ahead of, or 128 bytes behind the address of the next memory address after the branch instruction itself. This is one reason why I stressed the need for keeping your code as compact as possible.

The status flag is affected by most of the instructions in the 65Ø2 instruction set. The exceptions are the instructions STA, STX, STY, JSR, JMP, RTS, NOP, PHA, PHP, and TXS. Then we have the instructions which are used to directly control the individual flags within status register; CLC, CLD, CLI, CLV, SEC, SED, and SEI.

There are two other instructions available in the 65Ø2 which transfer program control, which are "JMP" and "JSR". "JMP" stands for "JuMP" and causes an unconditional transfer to the specified address, just like "GOTO" in BASIC. "JSR" stands for "Jump SubRoutine" and executes the code beginning at the specified address until an "RTS" or "ReTurn from Subroutine" instruction is encountered, just like "GOSUB" and "RETURN" in BASIC.

★

OSI ROM ROUTINES

(Part 4)

By: Leroy Erickson
Courtesy of OSMOSUS NEWS
3128 Silver Lake Road
Minneapolis, MD 55418

The ROM routine for this month is SYNMON page 4, the ROM BASIC Support routines for 54Ø video and the polled keyboard. This routine occupies $FFØØ to $FFFF in any C4P, C2-4P or C8P cassette based system. Since it covers locations $FFFA thru $FFFF, it contains the NMI, RESET and IRQ vectors. Look at those locations in Listing 1 and you'll see that they are set to $Ø13Ø, $FFØØ and $Ø1CØ, respectively. Thus, on receiving a RESET (BREAK) in a BASIC-IN-ROM system, control is passed to $FFØØ - the beginning of this page. Now look at the code at that location. The following set of operations is executed:

1. Clear decimal mode (just in case).

2. Set the stack pointer to $Ø128.

3. Initialize the serial port, using a routine assumed to exist in the BASIC ROM(s).

4. Initialize several flags that BASIC will need.

5. Initialize the video cursor position.

6. Clear the screen.

7. Display the boot message 'C/W/M?' on the video screen, using another routine which is assumed to exist in the BASIC ROM(s).

8. Get an input character, using the routine in the keyboard driver ROM.

9. Test that character and do the following:- if 'M', go to the ROM Monitor at $FEØØ.

- if 'W', go to the warm start jump assumed to exist at $ØØØØ.

- if 'C', go to the cold start routine in BASIC-IN-ROM.

- if none of the above, go back to $FFØØ and start all over.

The code to handle the above tasks occupies about half of the page. The rest of the page contains the following routines:

1. $FF67 is a character display routine which first displays to the screen, then tests the 'SAVE' flag. If set, the character is also sent to the serial port. Also, if in 'SAVE' mode and a carriage return is being displayed, 1Ø nulls (ASCII 'ØØ') are written to the serial port as a delay.

2. At $FF89 is the code to handle the 'LOAD' command. When a 'LOAD' command is given, its flag is set (changed from $ØØ to $FF) and the 'SAVE' flag is cleared (set to Ø). The 'LOAD' flag is cleared later when a 'space' is typed on the keyboard.

3. At $FF94 is the code to handle the 'SAVE' command. All that happens is that the 'SAVE' flag is set to 1.

4. At $FF99 is the code to test for CTRL/C input from the keyboard. If not true, control is returned to the calling routine. If true, control is passed to $A636 in the BASIC ROM(s). The routine here will do the correct stop sequence for an executing BASIC program.

5. At $FFB8 is the character input routine. If the 'LOAD' flag is clear, control is passed to the ROM keyboard routine. If set, the routine continuously tests the keyboard for a space (ASCII '2Ø') input or the serial port for any input. If the serial port wins, that character is returned, otherwise the 'LOAD' flag is cleared and control is passed to the ROM keyboard handler.

6. At $FFEB is a set of 5 jumps to the 5 routines listed above. Presumably, BASIC-IN-

```
                ; *************************************
                ; ***                               ***
                ; ***       C4P BOOT ROM PAGE 4     ***
                ; ***                               ***
                ; *** ROM BASIC Support for 540 Video ***
                ; ***        and Polled Keyboard    ***
                ; ***                               ***
                ; ***   Comments by  Leroy Erickson ***
                ; ***            May  1982          ***
                ; ***                               ***
                ; *************************************
                ;
0000=            H0000 =$0000 ; BASIC Warm Start Location
                              ;
0200=           CRSPOS=$0200 ; Video Cursor Position
0203=           LOADFG=$0203 ; LOAD Flag
                             ; Non-zero ==> Serial Input
0205=           SAVEFG=$0205 ; SAVE Flag
                             ; Non-zero ==> Serial Output
0206=           TDELAY=$0206 ; Time Delay
0212=           CTLCFG=$0212 ; CTRL/C Flag
                             ; Non-zero ==> Disabled
                             ;
                             ; * BASIC-IN-ROM Routines *
                             ;
A636=           HA636 =$A636 ; CTRL/C Handler
BD11=           HBD11 =$BD11 ; Cold start entry
BF15=           HBF15 =$BF15 ; Serial output
BF22=           HBF22 =$BF22 ; Init serial port
BF2D=           HBF2D =$BF2D ; Video Driver
                             ;
D000=           SCREEN=$D000 ; Address of video memory
DF00=           KEYBRD=$DF00 ; Address of keyboard port
FC00=           SERPRT=$FC00 ; Address of serial port
FE00=           HFE00 =$FE00 ; ROM Monitor start address
FEED=           HFEED =$FEED ; Address of Jump to
                             ;  Keyboard Input Routine
                 ;
FF00                   * = $FF00
                 ;
FF00 D8          HFF00  CLD              ; Clear decimal mode
FF01 A228               LDX   #$28       ; Set stack pointer
FF03 9A                 TXS              ;
FF04 2022BF             JSR   HBF22      ; Init Serial Port
FF07 A000               LDY   #$00       ; Initialize Flags
FF09 8C1202             STY   CTLCFG     ; - CTRL/C Flag
FF0C 8C0302             STY   LOADFG     ; - LOAD Flag
FF0F 8C0502             STY   SAVEFG     ; - SAVE Flag
FF12 8C0602             STY   TDELAY     ; - Time Delay
FF15 ADE0FF             LDA   HFFE0      ; Initialize Cursor
FF18 8D0002             STA   CRSPOS     ;  Position
                                         ;
                                         ; * Clear the Screen *
                                         ;
FF1B A920               LDA   #$20       ; Get a blank
FF1D 9900D7      HFF1D  STA   SCREEN+$700,Y ; Clear last 8th
FF20 9900D6             STA   SCREEN+$600,Y ; Clear next 8th
FF23 9900D5             STA   SCREEN+$500,Y ; ditto
FF26 9900D4             STA   SCREEN+$400,Y ; ditto
FF29 9900D3             STA   SCREEN+$300,Y ; ditto
FF2C 9900D2             STA   SCREEN+$200,Y ; ditto
FF2F 9900D1             STA   SCREEN+$100,Y ; ditto
FF32 9900D0             STA   SCREEN,Y ; Clear top 8th
FF35 C8                 INY              ; Increment index
FF36 D0E5               BNE   HFF1D      ; Loop for a whole page
                                         ;
                                         ; * Display Boot Msg *
                                         ;
FF38 B95FFF      HFF38  LDA   BOOTMS,Y ; Get a char
FF3B F006               BEQ   HFF43      ; Exit if 0
FF3D 202DBF             JSR   HBF2D      ; Else, display it
FF40 C8                 INY              ; Increment index
FF41 D0F5               BNE   HFF38      ; Loop until all done
                                         ;
                                         ; Get & Test Response
                                         ;
FF43 20B8FF      HFF43  JSR   HFFB8      ; Get an input char
FF46 C94D               CMP   #'M        ; M ?
FF48 D003               BNE   HFF4D      ; No, skip
FF4A 4C00FE             JMP   HFE00      ; Yes, go to ROM Monitor
                                         ;
FF4D C957        HFF4D  CMP   #'W        ; W ?
FF4F D003               BNE   HFF54      ; No, skip
FF51 4C0000             JMP   H0000      ; Yes, go to Warm Start
                                         ;
FF54 C943        HFF54  CMP   #'C        ; C ?
FF56 D0A8               BNE   HFF00      ; No, start over
FF58 A900               LDA   #$00       ; Yes, clear A,X & Y
FF5A AA                 TAX              ;
FF5B A8                 TAY              ;
FF5C 4C11BD             JMP   HBD11      ; Go to BASIC Cold start
                                         ;
FF5F 43          BOOTMS .BYTE 'C/W/M ?',0 ; * Boot Message *
FF60 2F
FF61 57
FF62 2F
FF63 4D
FF64 20
FF65 3F
FF66 00
                                         ;
                                         ; * Display a char *
                                         ;
FF67 202DBF      HFF67  JSR   HBF2D      ; Display to the screen
FF6A 48                 PHA              ; Save the char
FF6B AD0502             LDA   SAVEFG     ; Test SAVE Flag
```

Continued on page 6.

D&N MICRO PRODUCTS, INC.

3702 N. Wells St.
Fort Wayne, Ind. 46808
(219) 484-6414

TERMS $3.00 shipping, Foreign orders add 15%, Indiana residents aaa 5% sales tax.

COMPUTER

MICRO-80 COMPUTER
Z-80A CPU with 4Mhz clock and CP/M 2.2 operating system. 64K low power static memory. Centronics parallel printer port. 3 serial ports. 4" cooling fan. Two 8" single or double sided floppy disk drives. IBM single density 3740 format for 243K or storage, double density format for 604K of storage. Double sided drives allow 1.2 meg on each drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane, 48 pin buss compatible with OSI boards.

MODEL 80-1200 2 8" Single sided drives	$2995
MODEL 80-2400 2 8" Double sided drives	$3495

MICRO-65 COMPUTER
6502 CPU with 2Mhz clock and DOS-65 operating system. 48K of low power static memory. 2 serial ports and 1 Centronics parallel port. 2 8" single or double sided drives. Satin finish extruded aluminum with vinyl woodgrain finish. 8 slot backplane, 48 pin buss compatible with OSI. Will run OSI 65D and 65U software.

MODEL 65-1 2 8" Single sided drives	$2995
MODEL 65-2 2 8" Double sided drives	$3495

BP-580 8 Slot Backplane . . . . . $ 47
OSI 48 pin Buss compatible

MEM-CM9 MEMORY/ FLOPPY CONTROLLER

24K memory/floppy controller card uses 2114 memory chips, 1 8K and 1 16K partition. Supports OSI type disk interface

24MEM-CM9	$325
16MEM-CM9	$260
8MEM-CM9	$180
BARE MEM-CM9	$ 50
Controller on assembled unit add	$ 90

BIO-1600 Bare IO card . . . . $ 50
Supports 8K of memory, 2 16 bit parallel ports, 5 serial ports, with manual and Molex connectors.

PRINTERS

Okidata

ML82A, 120 cps, 10"	$409
ML83A, 120 cps, 15"	$895
ML84 Parallel, 200 caps, 15"	$1150

C. Ioth

8510AP Prowriter, parallel 120 cps, correspondence quality	$419
8510APD Prowriter, serial	$585
F10-40PU Starwriter, parallel Letter quality daisy wheel	$1319
F10-40RU Starwriter, serial	$1319
F10-55PU Printmaster parallel, Letter quality daisy wheel	$1610
F10-55RU Printmaster, serial	$1610

DISK DRIVES AND CABLES

8" Shugart SA801 single sided	$385
8" Shugart SA851 double sided	$585
FLC-6 6 ft cable from D&N or OSI disk controller to 8" drive	$69
5¼" MPI B51 disk drive with cable, power supply and cabinet. Specify computer type.	$450
FLC-5¼ cable for connection to 5¼ drive and D&N or OSI controller, with data separator and disk switch. Specify computer type	$75

HARDWARE

OSI COMPATIBLE

IO-CA10X Serial Printer Port . . $125
Specify Device #3 or #8

IO-CA9 Parallel Printer Port . . $150

CMOS-MEM

64K CMOS static memory board, uses 6116 chips, 3 16K, 1 8K and 2 4K blocks, Partitionable for multi-user, OSI type disk controller, 2 IO mapped serial ports for use with D&N-80 CPU. Ideal way to upgrade from cassette to disk.

64K CMOS-MEM	$490
48K CMOS-MEM	$390
24K CMOS-MEM	$250
16K CMOS-MEM	$200
Controller add.	$ 90
2 IO mapped serial ports add. on assembled memory board	$125
Z80-IO 2 IO mapped serial ports for use with D&N-80 CPU card	$160
FL470 Disk Controller Specify 5¼ or 8" drive	$155

STANDARD CP/M FOR OSI

D&N-80 CPU CARD
The D&N-80 CPU allows the owner of an OSI static memory computer to convert to Industrial Standard IBM 3740 single density disk format and CP/M operating system. Double density disk operation is also supported for 608K of storage on an 8" diskette. When used with a 5¼" disk system 200K of storage is provided. Includes parallel printer and real time clock. Also available for polled keyboard and video systems. Compatible with C2, C3, C4 and 200 series OSI computers.

D&N-80-P	**$349**
CP/ M 2.2	**$150**
64K CMOS-MEM with D&N-80 CPU card	**$450**

HARD DISK DRIVER $140
Allows D&N-80 CPU board to control OSI 40 or 80 meg hard disk unit. Will not destroy OSI files. Will also allow for a true 56K CP/M system. Specify 40 or 80 meg drive.

BUSS TRANSFER $135
Allows for D&N-80 and OSI CPU to be in the computer at the same time. Toggle switch provides for alternate CPU operation.

DISK TRANSFER $100
Utility program to transfer OSI CP/M format disk to IBM 3740 single density format. Will also transfer IBM to OSI format.

SYSTEM HARDWARE REQUIREMENTS
D&N-80 CPU, D&N FL470 or OSI 470 controller, 48K memory at 0000-BFFF, 4K memory at D000--DFFF, two disk drive cables.

FORMAT TRANSFER $15
You supply software on 8" diskette D&N will transfer OSI CP/M format to IBM 3740 CP/M format. Can also transfer IBM 3740 CP/M format to OSI CP/M format. Original diskette returned.

ROM calls these 5 addresses so that the above code is not location dependent.

I'll finish off with 3 observations.

1. Nothing in this ROM page uses any location on page zero! This allows a 'RESET' and warm start to successfully work.

2. There are 7 unused bytes at $FFD9 to $FFDF and 10 unknown bytes (BASIC data?) at $FFE1 to $FFEA.

3. If the programmer at OSI had only taken the code at $FF1B to $FF36, moved it behind location $FFD8 and tagged an 'RTS' onto it, then placed a 'JSR' to this routine at $FF1B instead and packed everything back together again, the following would be true:

1. The code would work just the same way as it does now.

2. There would only be 3 unused bytes. (7 minus 3 for the 'JSR' and 1 for the 'RTS').

3. There would be a machine language 'screen clear' routine in ROM which could be directly called by a 'USR(X)' function from BASIC, thus nullifying several dozen magazine articles and/or letters in the last few years.

But OSI did'nt do things that way because nobody ever would want or need to use a routine like that, would they?

Next month, we'll cover SYNMON page 6, the Serial System ROM Monitor. See you then.

BEGINNER'S CORNER

By: L. Z. Jankowski
Otaio Rd 1, Timaru
New Zealand

GOT IT!

One of the problems with OSI BASIC has been the lack of an adequate 'GET-KEY' command, e.g., GET or INKEY as in other BASICs. Never mind, a halting 'GET-key' routine is easy to implement. For 65D 3.3, see line 310 in Listing 1. (Listing 1 is the third part of the 'Otaio Mailing List' - see June '84 issue). For DOS 3.2 change line 310 to:

```
310 DISK!"GO 252B":Y$=CHR$
    (PEEK(9815)):Y=VAL(Y$)
    :A=PEEK(9815)OR32.
```

```
FF6E F022          BEQ    HFF92     ; Skip is clear
FF70 68            PLA              ; Else regain output char
FF71 2015BF        JSR    HBF15     ; Write to Serial Port
FF74 C90D          CMP    #$0D      ; Is it Carriage Return ?
FF76 D01B          BNE    HFF93     ; No, Go Home
FF78 48            PHA              ; Yes, save it
FF79 8A            TXA              ;   and X
FF7A 48            PHA              ;
FF7B A20A          LDX    #$0A      ; Get a 10
FF7D A900          LDA    #$00      ;   and a Null
FF7F 2015BF HFF7F  JSR    HBF15     ; Write it to serial port
FF82 CA            DEX              ; Do that for 10 Nulls
FF83 D0FA          BNE    HFF7F     ;
FF85 68            PLA              ; Then regain X & A
FF86 AA            TAX              ;
FF87 68            PLA              ;
FF88 60            RTS              ; And Go Home
                                    ;
                                    ; * Handle LOAD Command *
                                    ;
FF89 48     LOADCM PHA              ; Save A
FF8A CE0302        DEC    LOADFG    ; Set LOAD Flag
FF8D A900          LDA    #$00      ; Clear SAVE Flag
FF8F 8D0502 HFF8F  STA    SAVEFG    ;
FF92 68     HFF92  PLA              ; Regain A
FF93 60     HFF93  RTS              ; And Go Home
                                    ;
                                    ; * Handle SAVE Command *
                                    ;
FF94 48     SAVECM PHA              ; Save A
FF95 A901          LDA    #$01      ; Get a 1 for SAVE Flag
FF97 D0F6          BNE    HFF8F     ; Go share code
                                    ;
                                    ; * CTRL/C Test *
                                    ;
FF99 AD1202 CTCTST LDA    CTLCFG    ; CTRL/C Enabled ?
FF9C D019          BNE    HFFB7     ; No, Go Home
FF9E A901          LDA    #$01      ; Else, test Row 0
FFA0 8D00DF        STA    KEYBRD    ;   for CTRL key
FFA3 2C00DF        BIT    KEYBRD    ;
FFA6 500F          BVC    HFFB7     ; Not down, Go Home
FFA8 A904          LDA    #$04      ; Else, test Row 2
FFAA 8D00DF        STA    KEYBRD    ;   for C Key
FFAD 2C00DF        BIT    KEYBRD    ;
FFB0 5005          BVC    HFFB7     ; Not down, Go Home
FFB2 A903          LDA    #$03      ; Else, get ASCII Value
FFB4 4C36A6        JMP    HA636     ; Go to BASIC ROM CTRL/C
                                    ;   Handler
                                    ;
FFB7 60     HFFB7  RTS              ; Go Home
                                    ;
                                    ; * Get Char Routine *
                                    ;
FFB8 2C0302 HFFB8  BIT    LOADFG    ; Test LOAD Flag
FFBB 1019          BPL    HFFD6     ; Skip if clear
FFBD A902   HFFBD  LDA    #$02      ; Else, test Row 1
FFBF 8D00DF        STA    KEYBRD    ;   Column 4 of the
FFC2 A910          LDA    #$10      ;   Keyboard - a Space
FFC4 2C00DF        BIT    KEYBRD    ;
FFC7 D00A          BNE    HFFD3     ; It's there, Skip ahead
FFC9 AD00FC        LDA    SERPRT    ; Else test serial status
FFCC 4A            LSR    A         ;
FFCD 90EE          BCC    HFFBD     ; Loop until 1 or the
                                    ;   other happens
FFCF AD01FC        LDA    SERPRT+1  ; If set get input char
FFD2 60            RTS              ; And Go Home
                                    ;
FFD3 EE0302 HFFD3  INC    LOADFG    ; If ' ' clear LOAD Flag
FFD6 4CEDFE HFFD6  JMP    HFEED     ; Go to ROM Keyboard Code
                                    ;
FFD9 00            .BYTE 0,0,0,0    ; * JUNK FILLER *
FFDA 00
FFDB 00
FFDC 00
FFDD 00            .BYTE 0,0,0      ;
FFDE 00
FFDF 00
                                    ;
FFE0 40     HFFE0  .BYTE $40        ; Initial video cursor pos
                                    ;
FFE1 3F            .BYTE $3F,$01,$00,$03 ; * JUNK (?) *
FFE2 01
FFE3 00
FFE4 03
FFE5 FF            .BYTE $FF,$3F,$00,$03
FFE6 3F
FFE7 00
FFE8 03
FFE9 FF            .BYTE $FF,$3F
FFEA 3F
                                    ;
                                    ;
FFEB 4CB8FF HFFEB  JMP    HFFB8     ; Character In Routine
FFEE 4C67FF HFFEE  JMP    HFF67     ; Character Out Routine
FFF1 4C99FF HFFF1  JMP    CTCTST    ; CTRL/C Test Routine
FFF4 4C89FF HFFF4  JMP    LOADCM    ; LOAD Command Handler
FFF7 4C94FF HFFF7  JMP    SAVECM    ; SAVE Command Handler
                                    ;
FFFA 3001   NMIVCT .WORD $0130      ; NMI VECTOR
FFFC 00FF   RESVCT .WORD $FF00      ; RESET VECTOR
FFFE C001   IRQVCT .WORD $01C0      ; IRQ & BRK VECTOR
                                    ;
                   .END
```

Congratulations Europe - You Now Have a DBI Distributor

DBI Welcomes P.M.C. ApS

P.M.C. ApS offers complete Dealer Support:

- Hardware Support (D.B.I. products in stock)
- Software Support (Specialized programs)
- Technical Support (Repair/Installation)

For Replacement and Upgrading O.S.I.* (65U*) 200 Series Computers with D.B.I. products.

P.M.C. ApS Professionals are:
Joergen Clausen
Niels Koldborg
Carsten Sillemann

P.M.C. ApS
Taarnfalkenvej
DK - 2650 Hvidovre
Copenhagen, Denmark

Telefon: 45-1-49 30 66
Telex: 42563 Fulvic DK

*O.S.I. and 65U are Trademarks of ISOTRON Inc.

Wir gratulieren - nun hat auch Europa eine Vertretung fuer DBI

DBI heisst P.M.C. ApS willkommen

P.M.C. ApS liefert vollstaendige Verkaufsunterstuetzung:

- Geraete Service (D.B.I. Componenten an Lager)
- Software (Spezielle Kundenprogramme)
- Technische Unterstuetzung (Reparaturen-Installationen)

Wir ersetzen und ergaenzen O.S.I.* (65U*) 200 Serien Computer mit D.B.I. Componenten.

Das zustaendige Personal der P.M.C. ApS sind:
Joergen Clausen
Niels Koldborg
Carsten Sillemann

P.M.C. ApS
Taarnfalkenvej
DK - 2650 Hvidovre
Copenhagen, Denmark

Telefon: 45-1-49 30 66
Telex: 42563 Fulvic DK

*O.S.I. and 65U sind registrierte Markenzeichen von ISOTRON INC.

Félicitations Europe - Vous avez, maintenant, un distributeur DBI.

DBI souhaite la bienvenue à P.M.C. ApS

P.M.C. ApS offre un soutien concessionnaire total, que ce soit en:

- machines (matériel D.B.I. en stock)
- ou en software (programmes spécialisés)
- avec appui technique (réparations/installations)

pour le remplacement et l'amélioration des ordinateurs O.S.I.* (65U*) série 200 avec produits D.B.I.

Les agents professionnels P.M.C. ApS sont:
Joergen Clausen
Niels Koldborg
Carsten Sillemann

P.M.C. ApS
Taarnfalkenvej
DK - 2650 Hvidovre
Copenhagen, Denmark

Telefon: 45-1-49 30 66
Telex: 42563 Fulvic DK

*O.S.I. et 65U sont des marques déposées de ISOTRON Inc.

DBi, Inc.
p.o. box 7276
denver, co 80207
(303) 428-0222

For ROM BASIC use:

```
310 POKE 11,0: POKE 12,253:
    X=USR(X)

312 Y$=CHR$(PEEK(531)):
    Y=VAL(Y$): A=PEEK(531)OR32
```

To see why Y$, Y and A are used, run line 310 with line 320:

```
320 PRINT Y$, Y, A: GOTO 310
```

Notice that the number in variable 'A' has the same value, irrespective of whether lower or upper case is used.

The 'GET-key' routine above is a halting one; it waits until a key is pressed. A non-halting 'GET-key' command would be extremely useful in program loops that must keep on doing something until a key is pressed. CTRL-C does the job but with an undesirable side effect - it stops the program also! Various attempts have been made to circumvent this limitation of OSI BASIC. The best solution would be to add the new command 'GET' to the BASIC Interpreter.

Creating records is done in the APPEND block, lines 1700-1800. Looking at line 1710, if Z Records have been created then the next Record R must be number Z+1. Records can continue being created until there are 'N' of them, the maximum allowed, as set in line 130. Notice how the user is prompted and helped to make the correct response.

Line 1770 demonstrates how arrays can be much more useful than simple variables. The field names could have been stored in 5 variables, N1$, N2$, N3$, N4$ and N5$. An INPUT into each of them would require up to 5 lines of code. Using array N$(C) and a FOR...NEXT loop reduces this requirement to one line only. As a result the program is shorter, faster and more elegant.

If 'STOP' is typed in response to INPUT in line 1770 then an exit is made from the block via line 1790. At this point 'Q' is one more than is required for 'Z', therefore 'Z' is set to 'Q-1'. It would be silly to have a final invisible Record with 'STOP' in it so the next FOR...NEXT loop, in line 1790, erases that Record. Finally, 'Q' is set to 'N' and 'C' is set to 'P'. Doing this forces the proper termination of these loops. If this is not done unwanted addresses are left on the

LISTING 1

```
290 REM LISTING 1.
300 REM Get a Key
310 DISK!"GO 2336":Y$=CHR$(PEEK(9059)):Y=VAL(Y$):A=PEEK
                                                  (9059)OR32
315 RETURN
1690 REM
1700 REM APPEND RECORDS
1710 R=Z+1:IFR>NTHENPRINT"* No more space left *":GOTO200
1720 REM
1730 FORQ=RTON
1740 PRINT!(28):PRINT"* To return to main menu type:-  STOP
                                                      *":PRINT
1750 PRINT:PRINT"Record "Q"of"N:PRINT:PRINT
1760 REM
1770 FORC=1TOP:PRINT:PRINT"* "N$(C)" " ;:INPUTD$(Q,C)
1780 IFD$(Q,C)=H$THENPRINT:PRINT:PRINT:GOTO1750
1790 IFD$(Q,C)=S$THENZ=Q-1:FORY=1TOP:D$(Q,Y)="":NEXTY:Q=N:
                                                     C=P
1800 NEXTC,Q:GOTO190
```

LISTING 2

```
5 REM LISTING 2
10 PRINT!(28); : POKE 56900,1: SUM=53514+4096
20 WIDTH=64:SCREEN=SUM-WIDTH: CHOICES=6: L=CHOICES-1
30 FOR COUNT=1 TO CHOICES
40 :     NUMBER=COUNT: PRINT TAB(10) "> CHOICE" STR$(NUMBER)
50 NEXT COUNT: POKE 56900,5
60 :
70 FOR KEY=1 TO CHOICES
80 :     GOSUB 130: DISK!"GO 2336": LOOK = PEEK(9059)
90 :       IF LOOK=13 THEN NUMBER=KEY: KEY=CHOICES
100 NEXT KEY: IF LOOK=13 THEN 200   ....IF CR, show choice.
110 GOTO 70   ....go do it again.
120 :
130 FOR COUNT=0 TO 9: NUMBER=COUNT: POKE SCREEN+NUMBER,0
140 NEXT COUNT
150 SCREEN = SCREEN + WIDTH
160 IF SCREEN > (SUM + L*WIDTH) THEN SCREEN=SUM
170 FOR COUNT=0 TO 9: NUMBER=COUNT: POKE SCREEN+NUMBER,1
180 NEXT COUNT: RETURN
190 :
200 PRINT: PRINT "CHOICE " STR$ (NUMBER) " MADE"
210 PRINT:INPUT"* READY ";Q$: RUN
```

LISTING 3

```
10 REM LISTING 3.
20 PRINT!(28): X=4: DIM A$(X)
30 A$(1)="PRIN":A$(2)="PACK":A$(3)="SORT":A$(4)="FIND"
40 INPUT "* COMMAND ";B$
50 FOR Y=1 TO LEN(B$)-3: C$=MID$(B$,Y,4)
60 :     FOR Z=1 TO X: IF C$=A$(Z) THEN GOSUB 200: Z=X
70 :     NEXT Z
80 NEXT Y
90 END
200 ON Z GOSUB 1000,2000,3000,4000,5000: RETURN
800 NEXTY
1000 PRINT"PRINTING":RETURN
2000 PRINT"PACKING":RETURN
3000 PRINT"SORTING":RETURN
4000 PRINT"FINDING":RETURN
```

Stack. This could be fatal to a program where FOR...NEXT loops are mixed with GOSUB calls. Experiment with the following program to gain a deeper understanding of how a FOR...NEXT loop works. Try different values for A and B, including negative numbers and zero. Try A>B as well as A<B. Note each time the final value of COUNT.

```
10 REM LOOP

20 A=...: B=...

30 FOR COUNT=A TO B

40 PRINT "HA!",,

50 PRINT "COUNT= " COUNT

60 NEXT COUNT

70 PRINT,"FINAL COUNT=
   "COUNT,,,,"DONE"
```

The commas merely space the output across the screen. It is possible to produce a loop which will run forever or until some condition is met - this is where a non-halting GET command would be useful. Add these lines and RUN:

```
2Ø A=1: B=1

55 COUNT=Ø.
```

Now add this line to halt LOOP:

```
51 X=INT( RND(1)*1Ø ): IF X=1
   THEN 6Ø
```

Jumping Jodhpurs! It works just like 'LOOP' in MODULA 2!

POINTS ARISING

OSI BASIC Boolean operators (AND, OR, NOT) closely follow the rules of Boolean Algebra. The inventor of Boolean Algebra was George Boole (1815-1864). He was a primary school teacher and soon found that he had to learn more mathematics. He did some reading and eventually wrote 'The Mathematical Analysis of Logic'. Two years later he was appointed Professor of Mathematics at Queens College, Cork, Ireland. Boole showed that an algebraic structure could be abstract. As a result of his work we know that propositional logic (AND, OR, NOT) will always work, including in computer programs! So what does the 'OR 32' in line 31Ø do?

Every character has its associated ASCII code number. The ASCII number for 'A' is 65 - this is 01000001 in binary (a 'sixty-four' and a 'one'). If this binary number is now 'ORed' with 32 we have:

```
Ø1ØØØØØ1 is 'A'

ØØ1ØØØØØ is just 32

--------

Ø11ØØØØ1 = 97 in base 1Ø.

--------
```

What happens if the ASCII number for 'a' (97) is 'ORed' with 32?

```
Ø11ØØØØ1 is 'a'

ØØ1ØØØØØ is just 32

--------

Ø11ØØØØ1 = 97!

--------
```

The result is precisely the same. The statement 'X=PEEK (9Ø59) OR 32', will put the same value into variable X irrespective of whether the key pressed was for upper or for lower case.

WAZZAT!

Making programs easier to use makes them much longer. Listing 2 is a case in point. It shows an alternative way of presenting a Menu and a suitable version of it could be substituted for line 28Ø in the OML. C1P users make these changes:

```
1Ø SUM= 53514

25 WIDTH= 32

8Ø GOSUB 13Ø: POKE 11,Ø: POKE
   12,253: X=USR(X): LOOK=PEEK
   (531)

13Ø POKE SCREEN+11,32

14Ø REM

17Ø POKE SCREEN+11,23

18Ø RETURN
```

The listing also illustrates how readable a BASIC program can be. Notice that there is no need for REM in lines 1ØØ and 11Ø. Works for GOSUB too!

One needn't stop there. It is possible to write a Menu program (see Listing 3) that will do the following: accept ANY English sentence, extract key words representing commands; execute those commands. In effect a Command file has been set-up for execution. Gadzooks, just like CP/M?!

GARBAGE!!

By: Earl Morris
3200 Washington
Midland, MI 48640

I seem to be attracted by garbage. Not the kind you put in barrels out at the curb, but the kind made by using strings in BASIC. The March and June 1981 issues of PEEK explained a bug in the ROM BASIC garbage collector. This bug does not exist in DISK BASIC. However, when the disk garbage collector runs, it can introduce long delays in your program. The delay is proportional to the square of the number of strings to be collected. Jim Butterfield is the guru of PET BASIC and many of his ideas can be adapted to OSI BASIC with only a change of address. In the June and July (84) issues of COMPUTE, Jim explains the reason for long delays in collecting garbage strings. A number of ideas are given on how to avoid creating string garbage in the first place. Building up a string character by character such as:

```
FOR X=1 TO 64 : A$=A$ + "*"  :
NEXT X
```

is one of the worst offenders, creating over 2K of garbage. Such constructions are often found in word processors written in BASIC.

Butterfield suggests avoiding making garbage if possible. Or, if you must make garbage, do a local clean-up immediately after. The idea is to force a collection only on the string you have just created and not all the strings in memory. Just before creating garbage, move the top of BASIC pointer down to the current string pointer. All existing strings are now outside of the BASIC workspace and are ignored by the garbage collector. Then make the necessary garbage in building up the desired string and get rid of this garbage by forcing a collection with FRE. The collection will run very quickly since there is now only one valid string in string space. Finally, restore the top of BASIC pointers to continue normally.

This technique becomes useful when you have over 1ØØ strings in memory. With fewer strings the collection delay is too short to be a concern. Following is an example program using the local garbage collection modified for OSI BASIC. Normally, this code would be part of a larger program. The address pointer for ROM and DISK BASIC are different, so use line 10 or 20 as appropriate.

```
5 REM USE LINE 10 or LINE 20
  but not both
10 BL=84 : BH=85 : SL=80 : SH
   = 81 :REM FOR DISK BASIC
20 BL=85 : BH=86 : SL=81 : SH
   = 82 :REM FOR ROM BASIC
30 :
40 REM  MAKE A LOT OF STRINGS
   HERE
50 :
100 AL=PEEK(BL):AH=PEEK(BH) :
    REM SAVE TOP OF BASIC
110 ZL=PEEK(SL):ZH=PEEK(SH) :
    REM SAVE STRING POINTER
120 POKE BL,ZL:POKE BH,ZH    :
    REM LOWER TOP OF BASIC
130 FOR X=1 TO 64
140 A$=A$+"*"                :
    REM MAKE GARBAGE
150 NEXT
160 Z=FRE(0)                 :
    REM FORCE LOCAL GARBAGE
    COLLECTION
200 POKE BL,AL:POKE BH,AH    :
    REM RESTORE TOP OF BASIC
```

★

KEYBOARD MUSIC

By: Gerald M. Van Horn
640 S.W. Addison Ave.
Junction City, OR 97448

Some comments on the enclosed program. Right now it is running on DISK V 3.2, but it was revised from ROM BASIC system and, therefore, easily revised. Just change POKE 2073,96 in 690 to POKE 530,1 and POKE 2073,173 in lines 350 and 1220 to 530,0. It's not fancy, but the kids should get a kick out of it for awhile. You play the upper two rows of the keyboard as a piano keyboard. The computer stores the notes as H(S) and they can be played back by pressing the space bar. The computer also has its own tunes randomly selected by the slant bar. More or other tunes can be added. By storing the complicated numbers required to calculate the tones, I have numbered them from 1 to 57. This makes it easy to develop a tune.

This was developed on an 8K machine and should run with 8K if the songs are not too long.

KEYBOARD MUSIC

```
10 PRINT TAB(15);"PLAY MUSIC.  G.VAN HORN"
20 GOSUB920
30 PRINT"JUST PLAY THE Q ROW FOR MAIN NOTES AND THE NUMBER ROW
40 PRINT"FOR SHARPS AND FLATS.  TO REPEAT YOUR TUNE, HIT SPACE BAR
50 PRINT"4P PLAYS A TUNE IF YOU HIT SLANT BAR (/)"
60 POKE 56832,3:T=57089
70 DIM F(57),P(255),H(255)
80 DIMG(255),L(255)
90 REM LOAD TONES AS F(L)
100 FORL=1TO57:READF(L):NEXT
110 X=0
118 REM PLAY THE NOTES
120 GOSUB690
130 IFX=0GOTO120
140 N=F(X)
150 I=INT(49152/N)
160 POKE T,I
170 GOSUB1080
180 GOTO120
190 REM FILE OF THE TONES AVAILABLE TO COMPUTER
200 DATA 196.0,207.7,220.0,233.1,248.9
210 DATA 261.6,277.2,293.7,311.1,329.6,349.2,370.0,392.0
220 DATA 415.3,448.0,466.2,493.9,523.2,554.4,587.3,622.3
230 DATA 659.2,698.4,740.0,783.0,830.6,884.0,932.3,987.8
240 DATA 1046.5,1108.7,1174.7,1244.5,1318.5,1395.9,1480.0
250 DATA 1568.0,1661.2,1760.0,1864.6,1975.6,2093.0,2217.5
260 DATA 2244.3,2489.0,2637.0,2793.8,2959.9,3135.9,3322.4
270 DATA 3520.0,3729.3,3951.1,4186.0,4434.9,4698.6,49152
280 REM SELECT COMPUTERS TUNE
290 E=INT(3*RND(1)+1):ONEGOTO310,490,600
300 REM AND PLAY IT
310 READA:FORY=1TOA:READX,P:N=F(X)
320 I=INT(49152/N):POKET,I
330 FORL=3TO75*P:NEXT
340 FORQ=1TO10:POKET,1:NEXT
350 NEXTY:POKE2073,173:STOP
360 END
370 REM YANKEE DOODLE
380 DATA 64
390 DATA11,2,11,2,13,2,15,2,11,2,15,2,13,2
400 DATA6,2,11,2,11,2,13,2,15,2,11,4,10,4
410 DATA11,2,11,2,13,2,15,2,16,2,15,2,13,2
420 DATA11,2,10,2,6,2,8,2,10,2,11,4,11,4
430 DATA8.3,10,1,8,2,6,2,8,2,10,2,11,4
440 DATA6,3,8,1,6,2,4,2,3,4,6,4
450 DATA8,3,10,1,8,2,6,2,8,2,10,2,11,2
460 DATA8,2,6,2,11,2,10,2,13,2,11,4,11,4
470 DATA57,4,11,2,6,1,6,1,8,2,6,2,57,2,10,2,11,5
480 REM YELLOW ROSE OF TEXAS
490 READB:FORY=1TOB:READG,L:NEXT:GOTO310
500 DATA58
510 DATA13,1,11,1,10,2,13,2,13,2,13,2,15,2,13,4
520 DATA11,2,10,2,13,2,18,3,20,1,22,6
530 DATA13,2,13,2,22,2,22,2,22,2,22,2,20,4
540 DATA18,2,17,2,18,2,20,2,22,2,20,6
550 DATA13,1,11,1,10,2,13,2,13,2,13,2,15,2,13,2,13,3
560 DATA11,1,10,2,13,2,18,3,20,1,22,6
570 DATA13,1,13,1,13,2,23,2,23,2,23,2,23,2,22,2
580 DATA20,3,18,1,18,2,13,2,22,2,20,2,18,4,30,4
590 REM FOR THE SAKE OF AULD LANG SYNE
600 READC:FORY=1TOC:READM,O:NEXT:GOTO490
610 DATA31
620 DATA6,1,11,3,11,1,11,1,15,2
630 DATA13,3,11,1,13,3,15,1,13,1
640 DATA11,2,11,2,15,2,18,2,20,3
650 DATA57,1,20,2,18,3,15,1,15,2
660 DATA11,2,13,3,11,1,13,2,15,1
670 DATA13,1,11,3,8,1,8,2,6,2,11,4
680 REM LOOK-UP KEY DEPRESSED
690 K=57088:POKE2073,96:P=P+1:POKEK,2
700 IFPEEK(K)=128THENX=1:RETURN
710 IFPEEK(K)=2THENX=17:RETURN
720 POKEK,32:IFPEEK(K)=32THENX=15:RETURN
730 POKEK,16:IFPEEK(K)=128THENX=3:RETURN
740 IFPEEK(K)=64THENX=5:RETURN
750 IFPEEK(K)=32THENX=6:RETURN
760 IFPEEK(K)=16THENX=8:RETURN
770 IFPEEK(K)=8THENX=10:RETURN
780 IFPEEK(K)=4THENX=12:RETURN
790 IFPEEK(K)=2THENX=13:RETURN
800 POKEK,64:IFPEEK(K)=64THENX=14:RETURN
810 IFPEEK(K)=32THENX=16:RETURN
820 POKEK,128:IFPEEK(K)=64THENX=2:RETURN
830 IFPEEK(K)=32THENX=4:RETURN
840 IFPEEK(K)=8THENX=7:RETURN
850 IFPEEK(K)=4THENX=9:RETURN
860 IFPEEK(K)=2THENX=11:RETURN
870 POKEK,2:IFPEEK(K)=16GOTO1160
880 IFPEEK(K)=8GOTO290
890 IFX=0THENRETURN
898 REM 57 IS A REST IN YOUR TUNE
900 X=57:RETURN
910 REM PRINT OUT KEYBOARD
920 FORI=1TO32:PRINT:NEXT
930 M=54538:N=161:O=32:R=161
940 FORI=1TO4:FORJ=0TO29STEP3
950 POKEM+J,N:POKEM+J+1,O:POKEM+2+J,R:NEXTJ
960 IFI=2THENN=136:R=32
970 IFI=3THENN=209:O=128:R=128
980 M=M+64:NEXTI
990 M=M-64*4-1:GOSUB1020:M=M+9:GOSUB1020
1000 M=M+12:GOSUB1020:M=M+9:GOSUB1020
1010 M=M+65:POKEM+64,136:POKEM+64*2,136:RETURN
1020 POKEM,32:POKEM+1,136:POKEM+64,32:POKEM+65,136:RETURN
1040 POKEK,2
1050 IFPEEK(K)=16GOTO1160
1060 IFPEEK(K)=8GOTO290
```

```
1070 X=57:RETURN
1078 REM SAVE HOME MADE MUSIC
1080 IFH(S)()XGOTO1100
1090 RETURN
1100 P(S)=P
1110 S=S+1
1120 H(S)=X
1130 P=1
1140 RETURN
1150 REM PLAY BACK HOME MADE MUSIC
1160 FORA=1TOS
1170 N=F(H(A))
1180 I=INT(49152/N)
1190 POKE T,I
1200 FORL=1TO35*P(A):NEXT
1210 FORQ=1TO10:POKET,1:NEXT
1220 NEXTA:POKE2073,173:STOP
```

★

TRY US!

WE MEAN BUSINESS

SC5/80-1 Computer System

With 1 Denver Board Multi Processor
Wired for 6 Users
Expandable to 14 Users

$6990.00

For Each Add'l Multi Processor User Add $1,000.00

- **Includes Super Utility Package**

Plus our incredible new super system data base manager. With brand new Disk Tech One, 2 year warranty on internal hard disk components. Beautiful hand finished oak cabinet over steel frame. Instant access to all components. Available with casters or plastic feet

BEAUTIFUL AS WELL AS FUNCTIONAL

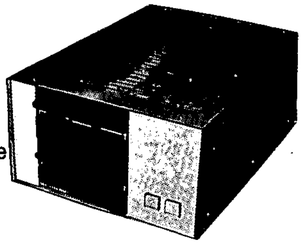
CLOSE OUT! C3-OEM

2 MHz/56K*
With 6502 and Z80 Processors

With Centronics Parallel Printer Interface

While they Last!

$1499.00

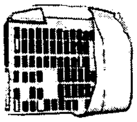


- Buy for a Spare
- Add to a Spare H.D.
- Develop Programs

*Includes 8K Exec Memory for CP/M or Multi-User

Spare 510CPU Board with Z80+6502 $169.00

NEW CONTROLLER/INTERFACE!



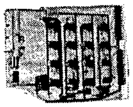
$599.00
Quantity 1

The new 9590 Controller replaces the OSI 590/525 set with 1 single board. Unlike others you've seen, it works perfectly with OSI or Denver boards, and all versions of 65U

It is compatible with old Style 592 Interface with 3 cable adapter, or attaches with single ribbon cable to our new 9592 H.D. Interface Board. Unlike others, our interface completely supports the ready/fault indicator on the 80 meg. hard disk.

9592
Quantity 1

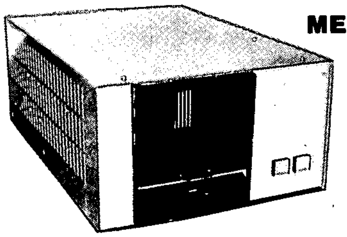
$269.00



MULTI PROCESSOR!

COMPLETE 10 MEGABYTE SYSTEM

$3999.00



Boot From Hard or Floppy!

- 8″ Hard Disk
- 8″ Floppy Disk

Single user with Centronics parallel printer interface expandable to eight users.

Enclosed in table top cabinet as shown or mounted in deluxe floor cabinet, as above left. Add $400.00

12 SLOT BUS

Fits standard OSI mount holes can be connected in "T" or daisy chained by cable for expansion.

$79.00

10 meg. Subsystem WOW!

$1999.00

Add to any existing OSI floppy based computer.

Just plug in one board and set this little gem on top or next to your computer and voila! Welcome to speed and convenience. Completely self contained with power supply. 9598 Hard Disk Controller also available separately at $699.00

DEALERS! **If you're still alive, call us and ask about our aggressive new price structure. We'll send you an unbelievable spares price list.**

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

MORE AND BIGGER DRIVES FOR OSI or DON'T BUTCHER DRIVE "B"

By: Ron Rose
Courtesy of OSMOSUS NEWS
Box 18801
Minneapolis, MN 55418

This simple modification to the 470 board and the A12 (paddle board) will allow the use of four single sided or 2 double sided drives. Also, it is not necessary to modify the drives in any way.

This scheme provides four discrete select lines, two of which can be used as side select for double sided drives. The four lines are provided by decoding the two outputs (pins 8 & 15) of the PIA (6821). The drive select codes are as follows:

PIN 8	PIN 15	SELECT
HI	HI	A
LO	HI	B
HI	LO	C
LO	LO	D

I selected a 7442 as the decoder (1 of 10) and mounted it in the proto area at U6A, connected +5V and ground, connected pins 12 and 13 to ground, then made the following trace cuts:

Component side -

CUT 1- at W2 from pin 8 of PIA (near pin 20 of PIA)

CUT 2- AT W3 (.8" LEFT OF PIA PIN 17)

CUT 3- AT W5 from pin 15 of PIA (3rd trace above pin 11 of U4C)

On solder side -

CUT 4- at W4 between pin 8 of U4C and plated thru hole

CUT 5- at pin 11 of PIA

CUT 6- at pin 14 of PIA

If done properly, you will have plated thru holes in which to mount wire wrap pins.

Next make the following connections:

Pin 8 of PIA to pin 15 of 7442

Pin 15 of PIA to pin 14 of 7442

Pins 3 and 11 of U1A to pin 1 of 7442 (DS4)

Pins 3 and 11 of U2A to pin 2 of 7442 (DS3)

Pins 1 and 13 of U4A to pin 3 of 7442 (DS2)

Pins 9 and 5 of U2A to pin 4 of 7442 (DS1)

The next step is to change the A12 board to conform to the following list (only a few wires need be changed, but it is best to check all):

MOLEX CONNECTOR J2		50 PIN DRIVE CONNECTOR
1 Head load	----------->	18 Head load
2 Low current	----------->	2 Write current
3 DS1	----------->	26 DS1
4 DS4 (new)	----------->	32 DS4
5 Step	----------->	36 Step
6 Step in	----------->	34 Dir select
7 DS3 (new)	----------->	30 DS3
8 Write enable	----------->	40 Write gate
9 Write data	----------->	38 Write data
10 Sep clk	----------->	50 Sep clk
11 Sep data	----------->	48 Sep data
12 Ground	----------->	all odd number pins
13 Ground	----------->	all odd number pins
14 N/C		
15 -9V		(connect only if necessary)
16 N/C		
17 Index	------------>	20 Index
18 DS2	------------>	28 DS2
19 Write Protect	------------>	44 Write protect
20 Ready drive 2	------------>	22 Ready (optional)
21 Sector	------------>	24 sector (optional)
22 N/C		
23 Track 0	------------>	42 track 0
24 Ready drive 1	------------>	22 Ready

Note: Both pins 20 and 24 of J2 connect to pin 22 of the drive connector. If you wish discrete ready lines, use the "radial ready" scheme in your drive manual. The sector line is necessary only if you have "hard sectored" drives.

All of the "strapping" on the single sided drives should be the same as on your original drive A with the exception of the drive select jumpers, DS1 thru DS4. DS1 is drive A etc.

Shugart 850's are jumpered as shipped except as follows:

HL	-	open
HLL	-	open
FS	-	jumper
Z	-	open
X	-	open
Y	-	jumper

Move S2 jumper to S3 (allows use of drive sel as side sel)

Recommended:
Drive A should be jumpered DS1 and 4B

Drive B should be jumpered DS2 and 3B

The above jumpering sets C as the other side of A and D as the other side of B. Different combinations of DSn and nB will allow you to configure any surface as any logical drive A thru D. If you are using the new OSI CP/M V2.25, the drives must be configured as suggested or you will not be able to use a double sided drive as 1 logical drive (570K).

OS65U INPUT TIPS FOR VERSION 1.2 DIE-HARDS

By: Julia A. Goodman
412 2nd Street
Radford, VA 24141

Anyone who has attempted to write a text input program in OS65U BASIC (v.1.2) knows that under normal conditions the statement "INPUT A$" or "INPUT %1,A$" fails to assign the exact input value to A$ in the following cases: (Assuming the input value is not begun with the double-quote as a delimiter.)

1. String contains comma or colon.

2. String contains a leading double-quote (as a character--not as a delimiter.)

3. String is null, and user wants to simply press RETURN or ENTER.

4. String contains an underline symbol.

5. String contains an "at-sign" (@).

6. String has leading spaces.

7. String has more than 71 characters (70 under LEVEL III).

A solution to each of the above problems is given below with usage warnings and considerations.

It is assumed that the EDITOR is enabled when the INPUT statement is executed. Previous PEEK(65) articles have led this author to suspect that not all Version 1.2 systems have the EDITOR routine located at the same address. The solutions below have been developed on OS65U (v.1.2) with the EDITOR routine at 15155 (decimal). Your version 1.2 system is probably the same if the following PEEK's hold true:

PEEK (1Ø243) is 51
PEEK (1Ø244) is 59
PEEK (1Ø268) is 51
PEEK (1Ø269) is 59

(Note: 51 + 256*59 = 15155)

SOLUTIONS

1.TO ACCEPT COMMAS AND COLONS:

For INPUT A$ and INPUT%1,A$:
Use POKE 2976,13 for comma, and POKE 2972,13 for colon.

Restore with POKE 2976,44 and POKE 2972,58.

Caution: Restore before using statements such as:

    READ A$,B$
    DATA JUDY,MARY

or A$ will be assigned the value "JUDY,MARY", and "?OD ERROR" will probably occur.

2. TO ACCEPT THE DOUBLE-QUOTE AS A LEADING CHARACTER:

For INPUT A$ and INPUT%1,A$:
Use POKE 297Ø,Ø. Restore with POKE 297Ø,7.

Note: This POKE does not affect the use of quotes around string constants in other parts of the program--except in DATA statements, where it will cause the double-quote to be treated as part of the data value--not as a delimiter.

3. TO ACCEPT NULL STRING--BY PRESSING "ENTER" OR "RETURN" FROM KEYBOARD--OR FROM A NULL STRING STORED IN A DATA FILE (STORED SIMPLY AS A CARRIAGE-RETURN, ASCII 13):

FOR INPUT A$ or INPUT%1,A$:
Use POKE 2888,Ø. Restore with POKE 2888,27.

Caution: Assigns Ø to A when INPUT A is executed.

Also prevents exit from program by null input regardless of FLAG 21/ FLAG 22 setting.

Figure 1: Loader of Alternate Input Machine Language

```
10 REM              FILENAME:  LDINP         4/7/84
99 :
100 REM This program loads a machine language routine at 24700-24746
110 REM to be called instead of the OS65U routine at 1368 (decimal)
120 REM for execution of INPUT stringvar. or INPUT% channel, stringvar
199 :
200 FOR I=24700 TO 24746: READ X: POKE I,X: NEXT I
250 END
499 :
500 DATA 169,44,141,0,96,162,0,32,245,39,201,13,240,21,201,32,144,245
510 DATA 201,128,176,241,224,121,176,237,157,1,96,232,32,238,10,208
520 DATA 228,169,0,157,1,96,162,0,160,96,76,210,63
```

Figure 2: Hex and Assembly Code for Alternate Input Routine

```
            :                               ; ML6  ( 47  BYTES)
            :                               ; LOAD AT 24700=124+256*96=$607C
            :                               ; Replacement input rtn. for 1368
            : BUFF=$6001
A9 2C       : LDA #$2C                      ; STORE COMMA AT BUFFER-1
8D 00 60:     STA BUFF-1
A2 00       : LDX #00                       ; BUFFER PTR.
20 F5 27:     JSR $27F5      (NEXT)         ; GET INPUT CHAR (IF CRT, GET ALL)
C9 0D       : CMP #$0D                      ; IF IT'S CR THEN END OF INPUT
F0 15       : BEQ CR
C9 20       : CMP $#20                      ; IF IT IS LESS THAN A SPACE OR . .
90 F5       : BCC NEXT                      ; GREATER THAN 127 THEN IGNORE
C9 80       : CMP #$80
B0 F1       : BCS NEXT
E0 79       : CPX #$79                      ; IF X >= 121 THEN FULL; IGNORE
B0 ED       : BCS NEXT
9D 01 60:     STA $6001,X                   ; STORE CHAR IN BUFFER
E8          : INX                           ; INCREMENT BUFFER PTR.
20 EE 0A:     JSR $0AEE                     ; SEND CHAR TO CRT
D0 E4       : BNE NEXT                      ; ALWAYS TRUE
A9 00       : LDA #$00       (CR)           ; STORE 0 AT END OF INPUT STRING
9D 01 60:     STA $6001,X
A2 00       : LDX #$00                      ; RETURN X,Y = BUFF-1
A0 60       : LDY #$60
4C D2 3F:     JMP $3FD2                     ; OS65U EXIT
```

Figure 3: Skeleton Program for Extended Input Applications

```
10 REM                   FILENAME: INP            4/7/84
20 :
30 REM Run under OS65U v.1.2 with EDITOR enabled.
99 :
100 REM Skeleton program for input of any characters into string
110 REM variables from console or data file.  Strings up to 120 chars.
115 REM The prompting question mark and space for console input is
116 REM suppressed.
119 :
120 REM Alternate input buffer at 24576.
130 REM Alternate input routine at 24700.
140 REM BASIC program begins at 24832 = 24576 + 256.
141 :
150 REM Use GOSUB 20000 just before input of strings; and GOSUB 21000
152 REM just after string input (or input loop).
153 :
155 REM Between those two GOSUB's, remember to use POKE 204,0 and
156 REM POKE 204,243: if leading spaces are expected in the input
157 REM values.  No spaces in program between the two POKES on 204.
199 :
200 GOSUB 30000           :REM INITIALIZATIONS
399 :
400 REM  *****  I/O APPLICATION ****************************************
401 :
849 :
895 REM  *****  END APPLICATION ****************************************
899 :
950 GOSUB 31000           :REM RESTORES
995 END
999 :
20000 REM --------- SUBRTN. TO CHANGE OS65U CALL TO INPUT LOOP----
20001 :
20100 POKE 9328,124: POKE 9329,96: POKE 8362,122: REM CALL OUR RTN.
20110 POKE 4926,0: REM GET VAR.LOCATOR TO RECOGNIZE DIFFERENT BUFFER
20120 POKE 15156,1: POKE 15160,96: REM CHANGE BUFF. ADDR. IN EDITOR
20130 POKE 15322,120: POKE 15334,120: POKE 15365,120: POKE 15441,120
20140 POKE 15344,118:    REM STRING MAX LEN & BELL RING POS. IN EDITOR
20195 RETURN
20999 :
21000 REM --------- SUBRTN. TO RESTORE OS65U INPUT LOOP -------------
21001 :
21100 POKE 9328,88: POKE 9329,5: POKE 8362,98
21110 POKE 4926,11
21120 POKE 15156,27: POKE 15160,0
21130 POKE 15322,70: POKE 15334,70: POKE 15365,70: POKE 15441,70
21140 POKE 15344,68:    REM STRING MAX LEN & BELL RING POS. IN EDITOR
21195 RETURN
21999 :
30000 REM --------SUB FOR INIT'S ------------------------------------
30001 :
30100 POKE 2976,13: POKE 2972,13: POKE 2970,0: REM FIX (, : ")
30110 POKE 15300,0: POKE 15308,0:       REM FIX UNDERLINE & AT-SIGN
30190 REM NEXT TWO LINES FOR NO PROMPT ON INPUT (NO "? ")
30200 POKE 2898,234: POKE 2899,234: POKE 2900,234
30210 POKE 18166,234: POKE 18167,234: POKE 18168,234
30995 RETURN
30999 :
```

Continued on next page

Note: Not needed when #7 below is used for long strings.

4. ACCEPT UNDERLINE SYMBOL:

For INPUT A$:
Use POKE 15300,0. Restore with POKE 15300,119.

For INPUT%1,A$:
No problem--underline is normally accepted.

5. ACCEPT THE AT-SIGN:

For INPUT A$:
Use POKE 15308,0. Restore with POKE 15308,189.

For INPUT%1,A$:
Use POKE 1392,0. Restore with POKE 1392,225. Not needed for data file input when #7 below is used for long strings.

6. ACCEPT LEADING SPACES:

For INPUT A$ and INPUT%1,A$:
Use POKE 204,0 just before the INPUT statement or short loop containing the INPUT statement. Restore immediately after INPUT statement, or short loop, with POKE 204,243.

CAUTION! CAUTION! CAUTION! In your program, there must be no spaces between "POKE 204,0" AND "POKE204,243" OR YOU WILL GET A "?SN ERROR". Valid examples of this "POKE":

```
100 POKE 204,0:INPUTA$,B$,T$
    (I):POKE204,243: PRINT B$
500 POKE204,0:FORI=1TON:INPUT%
    1,T$(I):NEXTI:POKE204,243
501 PRINT T$(2): REM NOW IT'S
    OK TO HAVE SPACES
```

7. ACCEPT STRINGS UP TO 120 CHARACTERS LONG: (Can be adapted for lengths up to 255.)

This one requires work--like machine language!

The idea is to set up a different buffer for the input string since the buffer used by 65U (at address 27,decimal) cannot be extended without clobbering part of the operating system code. All references to that buffer address during execution of a string INPUT statement must be altered to access our new buffer.

The procedure is given below.

7a. CREATE a BASIC file called INP with size of about 5000 bytes. Also CREATE a BASIC file called LDINP (load INP) with the same size. LDINP will be run to set up initial contents of INP. The file INP may be used as a starter file for any program which needs to input long strings which contain any printable ASCII characters.

```
31000 REM -------SUB FOR RESTORES OF INIT'S -------------------------
31001 :
31100 POKE 2976,44: POKE 2972,58: POKE 2970,7: REM RESTORE (,:")
31110 POKE 15300,119: POKE 15308,189:  REM RESTORE UNDERLINE & AT-SIGN
31200 POKE 2898,32: POKE 2899,233; POKE 2900,10: REM RESTORE ?-SPACE
31210 POKE 18166,32: POKE 18167,236: POKE 18168,10
31995 RETURN
31999 :
```

Figure 4: Sample Application to Insert in INP

```
410 DIM T$(200): OPEN"FILE1",1
420 PRINT"ENTER TEXT LINES (ENTER ## TO EXIT INPUT MODE):
429 :
430 GOSUB 20000:        REM ALTERNATE INPUT
440 FOR I=1 TO 200
450 POKE 204,0:INPUTT$(I):POKE204,243:     REM WATCH! NO SPACES
460 IF T$(I)="##" THEN 500
470 NEXT I
471 :
500 GOSUB 21000:        REM RESTORE NORMAL INPUT
501 :
510 NL = I - 1:         REM NL=NUMBER LINES OF TEXT ENTERED
520 PRINT:PRINT:PRINT"TEXT ENTERED WAS:
530 FOR I=1 TO NL: PRINT T$(I): NEXT I
531 :
540 PRINT:PRINT"NOW STORE TEXT IN DATA FILE 'FILE1' . . .
550 PRINT%1,NL
560 FOR I=1 TO NL: PRINT%1,T$(I): NEXT I: CLOSE
561 :
570 PRINT:PRINT"NOW READING AND PRINTING CONTENTS OF 'FILE1' . . .
580 OPEN"FILE1",1:  INPUT%1,NL
590 GOSUB 20000:        REM ALTERNATE INPUT
600 POKE204,0:FORI=1TONL:INPUT%1,A$:PRINTA$:NEXTI:POKE204,243
610 CLOSE
620 GOSUB 21000:        REM RESTORE NORMAL INPUT
```

7b. Reserve the beginning of the BASIC workspace for our longer input buffer and an alternate input loop by entering:

NEW 256

Then enter the BASIC program shown in Figure 1. The "NEW 256" causes the BASIC program to be stored at 24832 (24576+256) instead of at the usual 24576. We will use 24576-24699 as our input buffer; 24700-24746 to store an alternate input loop in machine language; and leave 24747-24831 as free space for future use (should you need it). Enter: SAVE"LDINP

7c. Now, RUN the program you have just entered from Figure 1. The program will read the machine language routine from the DATA statements at the end of the program and store the codes at 24700-24746. (The Assembly listing of the alternate input routine is shown in Figure 2.)

7d. If the RUN is successful, SAVE your program into the INP file. The 256 bytes reserved in front of your program are stored along with the program, thus saving the machine language routine.

(In case the INP file gets messed up, you still have LDIND which can always be run again to alter itself in the first 256 bytes for storage as a refreshed INP file.)

7e. Now that the necessary machine language is hidden away at the beginning of our INP file, let's get rid of the BASIC loader program, and enter the rest of the necessary alterations as the BASIC portion of the INP file as follows:

LOAD"INP
NEW 256 <------ (Clears out the BASIC part, but leaves first 256 bytes intact!)

(Now, enter the BASIC program shown in Figure 3.)

SAVE"INP

INP now contains a skeleton starter for any program in which you want to INPUT all kinds of character strings--no delimiter quotes required.

Notice the extra POKES in Subroutine 30000 to remove the question mark and space that are usually displayed for an INPUT. Now the operator can enter character #1 in position #1 on the CRT.

SAMPLE APPLICATION OF SKELETON PROGRAM

CREATE a BASIC file called IO and a DATA file called FILE1. LOAD"INP, change the filename in Line 10 to "IO", insert the lines shown in Figure 4, SAVE" IO", RUN, and test by entering lines such as:

```
ABC,DEF,GHI,_ _ _@@@
"LEADING QUOTE
OTHER QUOTES "AROUND THINGS"
  OR 8" DISK LEADING SPACES
NEXT LINE IS NULL

LONG LINE....................
........................END
LAST LINE
##
```

The lines you enter, should be printed back to you after you enter ## to exit. Then they are stored in the data file FILE1, read back in and printed again. All lines should be preserved exactly as you entered them!

CAUTIONS

Call Subroutine 21000 as soon as possible after an input operation. If your program bombs before 21000 restores input to the normal buffer, immediate commands go off into the wild blue yonder because the operating sytem doesn't find them in the right buffer--IF THAT HAPPENS, REBOOT AT ONCE--YOU HAVE NO CONTROL OVER THE MACHINE UNTIL YOU DO.

It is important to make your program as foolproof as possible when using the alternate input routine to prevent abnormal exits. Lock out CTRL/C, for example. Check lengths of filenames which are entered. Limit length of numeric values which are entered to prevent OV errors. There are ways to do those things, too. Watch for those methods as well as how to use INPUT#4 and PRINT#4 and make the TAB key work--in later articles.

* * * * * *

From the author
After countless hours of plowing through a disassembly of OS65U, PEEKing, and POKEing, reading PEEK(65), developing the solutions listed above and others, and applying them in writing a word processor which works under OS65U v.1.2 BASIC, and finally, writing this article--I looked up, as an after thought, the word die-hard (see title) in the The Merriam Webster Dictionary, 1974, and found:

die hard \'di,hard\ n: one who resists against hopeless odds.

Tell me about it!

★

COMBINED DIRECTORY UTILITIES FOR OS-65D

By: Bruce Spainhower
4015 S.W. Canyon Rd.
Portland, Oregon 97221

I have used OS-65D ever since expanding my C4P to a disk system in 1980. I feel the operating system has a number of advantages, the main one being the degree of control the programmer has over disk and memory access. One of the other nice things about 65D is that it begs to be improved. It is ideal for the programmer who enjoys the challenge of improving software efficiency like that of a good game of chess. I have made several improvements to the operating system, mostly at the assembly language level. But to start with, I attacked the utilities package that comes with 65D. The result is the BASIC program "DIRUTL" which combines the functions of DIR, CREATE, DELETE, RENAME, & ZERO. At the same time, I have added a number of features which clean up the operation of the program and speed execution.

First, the directory is printed in true four column format, i.e. entries are read top down, not left right. The full directory is displayed on one screen and is never scrolled off. After the directory is printed, the user is prompted for any changes. All interchange between user and program occurs on a single line, so that the directory is always visible. Also, a single keystroke selects any menu option. The directory data is kept on the disk sorted by track number. The CREATE function actually does an insert in the directory while moving remaining entries up. A DELETE repacks the data. The corrected directory is then re-displayed. Tracks are initialized and zeroed at the user's request. I used the string concatenation trick (lines 740,750) published in PEEK some years ago to provide the 3K of nulls in memory for this function. The original use for this is to build a series of strings containing blanks (ASCII 32) in the video memory area as a screen clear.

The time required for any of these functions is a fraction (literally) of what the original 65D v3.3 programs take, even including the insert and packing features. It functions equally well (excepting the CLS) under 65D versions 3.2 and 3.3, both serial and video. And, if typed in without spaces or REM lines, the entire program fits on one track of an eight-inch disk (two on a 5-1/4 inch). So what's the catch? Well, there is one: the program requires the directory to be sorted and packed to begin with (you've been meaning to do that anyway, right?). For that purpose, I have included the short program "DIRFIX". Simply run DIRFIX on each of your existing 65D disks and you'll never need to worry about it again. I have been using DIRUTL for nearly two years now, and it has methodically kept my disk directories in order.

Now for some details: In line 60, ES$ is defined. It is the 65D v3.3 video screen clear. By defining the CLS as a string, you avoid the problem of losing the print extensions in favor of the arctangent function in BASIC. The screen handling is always on line. It is only the "!(xx)" construct that is disabled. So use CHR$(27)CHR$(xx) instead for reliability. ES$ is redefined in line 7Ø for a DEC VT-52 terminal (the one I have at work). Change this line to conform to the serial terminal you may be using.

In line 8Ø, EL$ (erase line) is defined. It consists of a line of blanks preceded and followed by carriage returns. The purpose of this is to allow a line erase without scrolling. Line 37Ø inputs a single character from the console and returns its ASCII value in the variable K. Lines 38Ø-45Ø input a string without scrolling the screen. Line 16Ø is a compact way of adding a printer to the output. And the POKES in the disk I/O section keep the head loaded during the entire time of access to reduce disk drive clatter. Line 21Ø adds a blank entry onto the end of the directory data in memory for the DELETE function.

Don't forget to change the track numbers in the disk I/O section if you are using a 5-1/4 inch system. You'll also need to change the 76 (tracks) in line 65Ø to 39, and the ",1=54ØØ/C" statement in line 77Ø to ",1=54ØØ/8".

Lines 22Ø-29Ø print the four column directory, dropping a column each time a blank entry is found. This speeds up the display of the directory which is severely limited by BASIC. The "check for valid entry" section scans the directory data in memory and returns with F% set to zero if the requested entry is not found, otherwise F%=1 and D points to the found entry.

While the CREATE section may be a bit hard to follow, its basic function is to find the first available space on the disk large enough to hold the requested file. Then lines 68Ø to 7ØØ perform the directory insert and adjust. Line 56Ø simply tests to see if the directory is full and disallows a CREATE. The DELETE section is essentially the inverse of the CREATE, except that no calculation is needed. RENAME simply checks for valid entry names, and performs an overlay.

Each of the directory modification sections vectors back through the print directory section to update the data on the screen. You also get a second chance for another directory on the way out in case you are scanning several disks.

```
1 REM     OS-65D DIRECTORY UTILITIES PROGRAM v1.7
2 REM            by Bruce Spainhower
3 REM            4015 S.W. Canyon Rd.
4 REM            Portland, Oregon 97221
5 REM            (503) 222-2828 x51
6 REM
7 REM - Setup
8 REM
10 DEFFNA(X)=10*INT(X/16)+X-16*INT(X/16)
20 DEFFNB(X)=16*INT(X/10)+X-10*INT(X/10)
30 B=20480:L=35:U=10081
40 Q$=CHR$(34):CR$=CHR$(13):H$="File   Tracks
50 C$(1)="Create":C$(2)="Delete":C$(3)="Rename
60 ES$=CHR$(27)+CHR$(28):DV%=PEEK(10950):PRINT
70 IFDV%=1THENES$=CHR$(27)+CHR$(72)+CHR$(27)+CHR$(74)
80 FORX=0TO61:EL$=EL$+" ":NEXT:EL$=CR$+EL$+CR$
84 REM
85 REM - Initial messages & prompts
86 REM
90 PRINTES$CR$TAB(17)"OS-65D Directory Utilities":PRINT
100 PRINTTAB(21)"Which drive? ("CHR$(PEEK(9820)+64)") ";
110 GOSUB370:IFK<65ORK>68THEN130
120 DISK!"SE "+CHR$(K)
130 PRINTEL$TAB(17)"Printer output also? (No) ";:GOSUB370
140 IFK<>89THEN190
150 PRINTEL$TAB(20)"Device Number? ";:GOSUB370
160 PRINTCHR$(K);:POKE8994,DV%OR(2^(K-49)):GOTO190
164 REM
165 REM - Disk I/O
166 REM
170 POKEU,96:DISK!"SA 08,1=5000/1
180 POKEU,169:DISK!"SA 08,2=5100/1":GOTO220
190 POKEU,96:DISK!"CA 5000=08,1
200 POKEU,169:DISK!"CA 5100=08,2
210 FORP=B+512TOB+517:POKEP,L:NEXT:POKEB+518,0:POKEB+519,0
214 REM
215 REM - Print Directory
216 REM
220 PRINTES$CR$TAB(20)"OS-65D Disk Directory":PRINT
230 PRINTH$;:FORX=1TO3:PRINT"   "H$;:NEXT:PRINT
240 FORX=0TO61:PRINT"-";:NEXT:PRINT:C=48
250 FORY=0TO15:FORX=0TOCSTEP16:PRINTTAB(X);
260 IFPEEK((X+Y)*8+B)=LTHENC=C-16:GOTO290
270 FORE=(X+Y)*8+BTO(X+Y)*8+B+5:PRINTCHR$(PEEK(E));:NEXT
280 PRINTTAB(6)FNA(PEEK(E))"-"FNA(PEEK(E+1));
290 NEXT:PRINT:NEXT:PRINT:POKE8994,DV%:GOTO310
294 REM
295 REM - Prompt for changes & select
296 REM
300 FORT=0TO2500:NEXT
310 PRINTEL$TAB(14)"Create, Delete, or Rename a file? ";
320 GOSUB370:C%=K-66:IFK=82THENC%=3
330 ONABS(C%)GOTO560,780,830
340 PRINTEL$TAB(20)"Another Directory? ";:GOSUB370
350 IFK=89THENPRINTEL$;:GOTO100
360 PRINTEL$;:END
364 REM
365 REM - Single key input
366 REM
370 DISK!"GO 2339":K=PEEK(9059):RETURN
374 REM
375 REM - String input (enter at line 410)
376 REM
380 IFLEN(I$)THENPRINTCHR$(8)CHR$(32)CHR$(8);
390 IFLEN(I$)<2THENI$="":RETURN
400 I$=LEFT$(I$,LEN(I$)-1):RETURN
410 I$=""
420 GOSUB370:IFK=13THENPRINTEL$;:RETURN
430 IFK<32THEN420
440 IFK=95ORK=127THENGOSUB380:GOTO420
450 I$=I$+CHR$(K):PRINTCHR$(K);:GOTO420
454 REM
455 REM - Check for valid entry in directory
456 REM
460 PRINTEL$TAB(20)"File name to "C$(C%)"? ";
470 GOSUB410:P$=I$
480 F$=I$:IFI$=""THEN310
490 IFASC(F$)<65ORASC(F$)>90THEN460
500 IFLEN(F$)<6THENF$=F$+" ":GOTO500
510 FORD=BTOB+504STEP8
520 FORE=0TO5:IFPEEK(E+D)<>ASC(MID$(F$,E+1))THEN540
530 NEXT:F%=1:PRINTEL$;:RETURN
540 IFPEEK(D)<>LTHENNEXTD
550 F%=0:PRINTEL$;:RETURN
554 REM
555 REM - Create
556 REM
560 IFPEEK(20984)<>LTHEN320
570 GOSUB460:IFF%=0THEN590
580 PRINTTAB(14)Q$P$Q$" is already in the directory";:GOTO300
590 PRINTTAB(16)"Number of tracks for "Q$P$Q$" ";:GOSUB410
600 N%=VAL(I$):IFN%<1THENPRINTEL$;:GOTO590
610 FORD=BTOB+504STEP8
620 TE=FNA(PEEK(D+7)):TB=FNA(PEEK(D+14))
630 IFTB-TE>N%THEN660
640 IFTBTHENNEXT
650 IF76-TE<N%THENPRINTTAB(21)"No room for "Q$P$Q$;:GOTO300
660 PRINTTAB(5)"Space available beginning on track"TE+1;
670 FORH=B+504TODSTEP-8:IFPEEK(H)=LTHENNEXT
680 FORM=H+7TODSTEP-1:N=PEEK(M):POKEM+8,N:NEXT
690 FORI=1TO6:POKED+7+I,ASC(MID$(F$,I)):NEXT
700 POKED+14,FNB(TE+1):POKED+15,FNB(TE+N%)
704 REM
705 REM - Initialize
706 REM
710 PRINT"- INITIALIZE? ";
720 GOSUB370:IFK<>89THEN170
```

Continued

```
730 PRINTEL$TAB(23)"Initializing";
740 SL=PEEK(128):SH=PEEK(129):POKE128,0:POKE129,96:S$=CHR$(0)
750 FORC=0TO75:S$=S$+CHR$(0):NEXT:POKE128,SL:POKE129,SH
760 FORI=TE+1TOTE+N%:T$=RIGHT$(STR$(I+100),2)
770 DISK!"IN "+T$:DISK!"SA "+T$+",1=5400/C":NEXT:GOTO170
774 REM
775 REM - Delete
776 REM
780 GOSUB460:IFF%THEN800
790 PRINTTAB(16)Q$P$Q$" isn't in the directory";:GOTO300
800 PRINTTAB(22)"Deleting "Q$P$Q$;
810 FORW=DTOB+511:V=PEEK(W+8):IFPEEK(W)=LTHENIFW/8=INT(W/8)THEN170
820 POKEW,V:NEXT:GOTO170
824 REM
825 REM - Rename
826 REM
830 GOSUB460:J=D-1:IFF%=0THEN790
840 PRINTTAB(20)"New name for "Q$P$Q$"? ";:GOSUB410
850 GOSUB480:IFF%THENP$=I$:GOTO580
860 FORW=1TO6:POKEW+J,ASC(MID$(F$,W)):NEXT
870 PRINTTAB(18)Q$P$Q$" is changed to "Q$I$Q$;:GOTO170
```

```
10 REM  "DIRFIX"  by Bruce Spainhower  7/8/84
20 REM
30 PRINT"DIRECTORY SORTER & PACKER FOR OS-65D
40 REM
50 PRINT:INPUT"Which drive ";D$:DISK!"SE "+D$:PRINT
60 REM
70 PRINT"Loading ...
80 REM
90 DIME$(63),T%(63):DEFFNA(X)=10*INT(X/16)+X-16*INT(X/16)
100 B=20480:UL=10081:B$="######"+CHR$(0)+CHR$(0)
110 POKEUL,96:DISK!"CA 5000=08,1
120 POKEUL,169:DISK!"CA 5100=08,2
130 FORX=BTOB+504STEP8:IFPEEK(X)=35THEN160
140 FORY=0TO7:E$(C)=E$(C)+CHR$(PEEK(X+Y)):NEXT
150 T%(C)=FNA(PEEK(X+6)):C=C+1
160 NEXT:C=C-1:C%=C
170 REM
180 PRINT"Sorting ...
190 REM
200 C%=C%/2:IFC%=0THEN320
210 A=0:B=C-C%
220 D=A
230 E=D+C%
240 IFT%(D)<T%(E)THEN290
250 T$=E$(D):E$(D)=E$(E):E$(E)=T$
260 T%=T%(D):T%(D)=T%(E):T%(E)=T%
270 D=D-C%
280 IFD>-1THEN230
290 A=A+1:IFA>BTHEN200
300 GOTO220
310 REM
320 PRINT"Saving ...
330 REM
340 DISK!"ME 5000,5000
350 FORX=0TOC:PRINT#5,E$(X);:PRINT#9:NEXT
360 FORX=CTO63:PRINT#5,B$;:PRINT#9:NEXT
370 POKEUL,96:DISK!"SA 08,1=5000/1
380 POKEUL,169:DISK!"SA 08,2=5100/1
390 PRINT:INPUT"Another run ";YN$
400 IFASC(YN$)=89THENRUN50
```

DIRUTL has made life with 65D much more pleasant for me. I hope that you will also find it useful. You may also find some of the routines and "tricks" useful in other programs. Next time, I'll send along a machine code directory program which, while not as sophisticated as DIRUTL, does fit within OS-65D. It gives you a disk DIR command in place of the original 65D sector DIRectory command in a direct byte-for-byte code replacement. It is compatible with v3.2, v3.3, and all of the enhanced versions of 65D which don't already use the sector directory code space.

★

★

TIME & DATE FOR OSI

and

SOLVING THE OSI IRQ PROBLEM

By: L. Z. Jankowski
Otario Rd 1, Timaru
New Zealand

The OKI MSM5832RS clock chip has been available for about 3 years. It provides a 12/24 hour clock, date with leap year corrections, and a sophisticated Interrupt-provision capability. Added attraction are the CMOS low-power package and low-cost implementation. A clock card suitable for OSI computers could be put together for about $3Ø. An example of one design is the Tasker Bus clock-card, the clock chip receives its data via a 6821 PIA and the OSI 16 line I/O bus. The diagram shows how one side of the 6821 PIA is interfaced to the MSM 5832 clock chip.

It's a good idea to take PIA pin 34 to the RESET line. This guarantees that the clock will keep running when the <BREAK> key is hit. Otherwise, hitting <BREAK> may coincide with HOLD being high, and the clock will stop. If the clock stops for more than one second, it loses time. With RESET, zeros are written to the PIA registers starting the clock again.

The software presented in this article consists of two parts: a BASIC program to set the time and date and to test that the hardware is functioning correctly; an Interrupt-driven machine-code program that puts the time and date on the screen. Both programs assume that the hardware is arranged as shown in the diagram. The BASIC program is not affected by the exact nature of the Interrupt configuration.

The 5832 chip puts out 4 reference signals on DØ-D3 when CS, READ and AØ-A3 are all high and HOLD is low. For DØ, the 1Ø24Hz signal is not dependent on HOLD input level. Any one of these four signals can be sent to the PIA so that the PIA, in turn, can issue an Interrupt signal to the CPU. The obvious choice is the reference signal from D1, a pulse every second. This signal is routed to one of the PIA control lines.

Deciding which control line to use will depend on the type of PIA Output required, if any. The source code listing assumes control line CB2 will be used. If CB1 is required, then make the following change in the source code:

line 45Ø - change 'LDA #$ØC' to 'LDA #$Ø5'.

If CA1 or CA2 are to enable Interrupt output, then change line 38Ø to 'LDA #$Ø5' or to 'LDA #$ØC', respectively, Line 45Ø would now have to be changed to 'LDA #$Ø4' and line 54Ø would become 'LDA PIA'.

The machine-code program can be entered either at TOGGLE (toggles clock on/off) or at INIT. If you are running HOOKS, consider adding the command 'K*' which takes a jump to TOGGLE. Having such a command at hand is extremely useful - saves typing 'DISK!"GO F55A"' or some-such like! Do not be tempted to remove line 54Ø; reading the PIA at this point is essential to its correct functioning - the read clears the PIA Interrupt flag. Also, the jump in line 1Ø4Ø is essential if the 5832 is to be read correctly. The 5832 is a rather slow CMOS device. CPU speed does not affect the accuracy of the clock or the associated software.

It is advisable to turn the clock off when accessing disk - reading is OK, but writing to disk will freeze about 1 time in 1Ø.

For DOS 3.2, change line 6ØØ in the BASIC listing to:

```
6ØØ DISK!"GO 252B" :
Y$=CHR$(PEEK(9815)) :
Y=VAL(Y$) :RETURN
```

SOLVING THE OSI IRQ PROBLEM

OSI put the IRQ vector in the stack, at $Ø1CØ. To be at all useful it needs to be moved. I put mine at $F7FD. To do this, it is necessary to make a new Monitor using an EPROM programmer. In the Monitor, change two bytes at $FFFE and $FFFF - from $CØ and $Ø1, to $FD and $F7, or whatever.

But what about all that software that uses $Ø1CØ? No problem. From BEXEC* (BASIC), poke up the values for $4C CØ Ø1 to $F7FD, F7FE and F7FF, or to whatever the new IRQ address is. For example:

```
5 REM New IRQ at $F7FD=63485

1Ø X=63485  :POKE  X,76:  POKE
X+1,192: POKE X+2,1
```

Any software that now wants to use $Ø1CØ will be able to do so - via the new IRQ vector address which now contains the JMP to $Ø1CØ!

A more elegant method of writing the three bytes is to do so from the Monitor.

In my Monitor, at $FF4Ø, there is a JSR to $FFBA, i.e. 2Ø BA FF. This is the code which is waiting for a 'D/E/W/M'. So, at $FF4Ø substitute the address of the following patch. (In my case the patch is at $FB23, so I write at $FF4Ø - 2Ø 23 FB). Add to the patch the JSR to $FFBA and finish off with an RTS.

PATCH

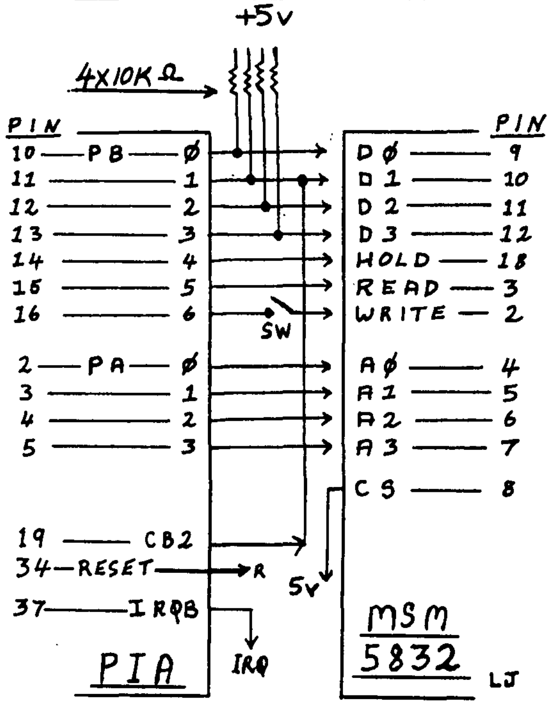
```
$FB23   LDA #$4C
.       STA IRQ
.       LDA #$CØ
.       STA IRQ+1
.       LDA #$Ø1
.       STA IRQ+2
.       JSR $FFBA
.       RTS
```

Continued on next page

LISTING 1

```
10 PRINT!(28) :REM CLS OS65D3.3
20 PRINT :PRINT TAB(16)"===== 12/24 HOUR CLOCK =====" :PRINT :PRINT
30 :
40 PIA = 50436 : REM $C504
50 :
60 DIM CR(12), T(6), D$(6) :FOR C=0 TO 6 :READ D$(C) :NEXT
70 DATA Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
80 :
90 PRINT"Do you wish to READ the time ? ";:GOSUB600 :IF Y$="Y" THEN 410
100 :
110 REM Get time & date
120 GOSUB 630 :PRINT!(28):PRINT"Close switch on Clock chip Write line.
130 PRINT :INPUT "Year (eg. 84) ";T(6)
140 INPUT "Month      ";T(5)      :INPUT "Day        ";T(4)
150 INPUT "Hour       ";T(2)      :INPUT "Minute     ";T(1) :T(3)=0
160 PRINT :PRINT"24 Hour Clock ? "; :GOSUB600 :PRINTY$ :PRINT
170 IF Y$="Y" THEN T(3)=8 :GOTO 190
180 PRINT"PM. ? ";:GOSUB600 :IF Y$="Y" THEN T(3)=4 :PRINTY$
190 PRINT :PRINT"Day of week (Sunday=0) ? ";:GOSUB600 :CR(6)=Y :PRINTY
200 PRINT :PRINT"Leap Year (Y/N) ? ";:GOSUB600 :PRINT Y$ :LY$=Y$
210 :
220 REM Fill array with time/date data
230 CR(0)=0 :CR(1)=0 :T=1
240 FOR R=2 TO 12 STEP 2 :IF R=6 THEN R=7
250 :    CR(R)=T(T)-10*INT(T(T)/10) :CR(R+1)=INT(T(T)/10)
260 :    T=T+1 :IF T=3 THEN T=4
270 NEXT :CR(5)=CR(5)+T(3) :IF LY$="Y" THEN CR(8)=CR(8)+4
290 :
300 REM WRITE the time & date.
310 POKE PIA,15 :POKE PIA+1,4 :POKE PIA+2,95 :POKE PIA+3,4
320 POKE PIA+2,16 : REM Pull HOLD high
330 FOR R=0 TO 12
340 :    POKE PIA,R :POKE PIA+2,CR(R)+80 :REM Pull HOLD & WRITE high
350 :    POKE PIA+2,CR(R)+16 :REM Write data & pull WRITE low (strobe)
360 NEXT :PRINT :PRINT :PRINT"Ready for GO! ? ";:GOSUB600
380 POKE PIA,0 :POKE PIA+2,0
390 :
400 REM READ the time & date.
410 PRINT!(28) :PRINTTAB(18)"==== Read the Clock ====":PRINT:PRINT
420 RH=48 :Z=0 :I=12 :F=15 :GOSUB 630
430 POKE PI,255 :POKE PI+1,4 :POKE PI+2,240 :POKE PI+3,4
440 P$="am." :POKEPI+2,RH :REM Pull READ & HOLD high
450 FOR X=Z TO I :POKE PI,X :CR(X)=PEEK(PI+2) AND F :NEXT
460 POKE PI+2,0 :POKE PI,0
470 H=10*(CR(5) AND 3)+CR(4) :M=10*CR(3)+CR(2) :S=10*CR(1)+CR(0)
480 MO=10*CR(10)+CR(9) :DM=10*(CR(8) AND 3) +CR(7) :Y=10*CR(12)+CR(11)
490 IF CR(5) AND 8 THEN P$=""
500 IF CR(5) AND 4 THEN P$="pm."
510 :
520 REM Write time & date to screen.
530 PRINT :PRINT :PRINT " HR : MI : SS" TAB(21) "MO / DM / YR"
540 PRINT :PRINT H ":" M ":" S; P$ TAB(20) MO "/" DM "/" Y :PRINT
550 DY$=D$(CR(6)) :PRINT :PRINT"Today is "DY$ :PRINT
560 PRINT:PRINT"Again ? ";:GOSUB600 :IFY$="Y" THEN PRINT!(28) :GOTO 440
570 PRINT :END
580 :
590 REM Get a key
600 DISK!"GO 2336" :Y$=CHR$(PEEK(9059)) :Y=VAL(Y$) :RETURN
610 :
620 REM Reset PIA
630 POKE PIA+1,0 :POKEPIA,0 :POKEPIA+3,0 :POKEPIA+2,0 :RETURN
```

6821 PIA to MSM 5832



Listing 2 on next page

LISTING 2

```
  10                ; MSM 5832 24 HOUR CLOCK & DATE PROGRAM
  20                ; by LZ JANKOWSKI.
  30                ; Time & date displayed on screen.
  40                ; This program assumes pulse
  50                ; to CB2 (PIA) - but see line 450.
  60                ;
  70 F55A                  * = $F55A
  80 C504=               PIA = $C504
  90 D0F7=            SCREEN = $D0F7
 100 F7FD=               IRQ = $F7FD
 110 0041=              SLOB = $41     Screen LOw Byte.
 120                ;
 130 F55A 48        TOGGLE PHA
 140 F55B AD6EF5           LDA SWITCH
 150 F55E F010             BEQ KON
 160                ;
 170 F560 78        KOFF   SEI           Disable IRQ.
 180 F561 A900             LDA #0
 190 F563 8D07C5           STA PIA+3
 200 F566 8D06C5           STA PIA+2
 210 F569 8D6EF5           STA SWITCH
 220 F56C 68               PLA
 230 F56D 60               RTS
 240                ;
 250 F56E 00        SWITCH .BYTE $0,$0
 250 F56F 00
 260                ;
 270 F570 EE6EF5    KON    INC SWITCH
 280 F573 A94C      INIT   LDA #$4C  Load IRQ vector.
 290 F575 8DFDF7           STA IRQ
 300 F578 AD1BF6           LDA LOHI
 310 F57B 8DFEF7           STA IRQ+1
 320 F57E AD1CF6           LDA LOHI+1
 330 F581 8DFFF7           STA IRQ+2
 340 F584 A900             LDA #00   Configure PIA port A.
 350 F586 8D05C5           STA PIA+1
 360 F589 A9FF             LDA #$FF
 370 F58B 8D04C5           STA PIA
 380 F58E A904             LDA #$04
 390 F590 8D05C5           STA PIA+1
 400 F593 A900             LDA #00   Configure PIA port B.
 410 F595 8D07C5           STA PIA+3
 420 F598 A9F0             LDA #$F0
 430 F59A 8D06C5           STA PIA+2
 440                ;
 450 F59D A90C             LDA #$0C  bit 5=0, bit 3=1, bit 2=1.
 460                ;
 470 F59F 8D07C5           STA PIA+3
 480 F5A2 2010F6           JSR RUPT
 490 F5A5 68               PLA
 500 F5A6 58               CLI          Enable IRQ.
 510 F5A7 60        RETURN RTS
 520                ;
 530 F5A8 48        CLOCK  PHA
 540 F5A9 AD06C5           LDA PIA+2 Clear PIA Interrupt flag.
 550 F5AC 8A               TXA
 560 F5AD 48               PHA
 570 F5AE 98               TYA
 580 F5AF 48               PHA
 590 F5B0 A930             LDA #$30   Pull HOLD & READ high.
 600 F5B2 8D06C5           STA PIA+2
 610                ;
```

```
 620 F5B5 A008             LDY #8        Read Time loop.
 630 F5B7 A200             LDX #0
 640 F5B9 20FAF5    READ   JSR LOOP1
 650 F5BC 99F7D0    STORE  STA SCREEN,Y
 660 F5BF 88               DEY
 670 F5C0 A93A             LDA #$3A
 680 F5C2 C006             CPY #6
 690 F5C4 F0F6             BEQ STORE
 700 F5C6 C003             CPY #3
 710 F5C8 F0F2             BEQ STORE
 720 F5CA E8               INX
 730 F5CB E006             CPX #6
 740 F5CD D0EA             BNE READ
 750                ;
 760 F5CF A041             LDY #SLOB      Read Date loop.
 770 F5D1 A20A             LDX #10
 780 F5D3 C049      READ2  CPY #SLOB+8
 790 F5D5 F01A             BEQ OUT
 800 F5D7 20FAF5           JSR LOOP1
 810 F5DA 99F7D0    PUT    STA SCREEN,Y
 820 F5DD C8               INY
 830 F5DE A92F             LDA #$2F
 840 F5E0 C043             CPY #SLOB+2
 850 F5E2 F0F6             BEQ PUT
 860 F5E4 C046             CPY #SLOB+5
 870 F5E6 F0F2             BEQ PUT
 880 F5E8 CA               DEX
 890 F5E9 E006             CPX #6
 900 F5EB D002             BNE GO
 910 F5ED A20C             LDX #12
 920 F5EF D0E2      GO     BNE READ2
 930                ;
 940 F5F1 2010F6    OUT    JSR RUPT
 950 F5F4 68               FLA
 960 F5F5 A8               TAY
 970 F5F6 68               PLA
 980 F5F7 AA               TAX
 990 F5F8 68               PLA
1000 F5F9 40               RTI
1010                ;
1020                ; Write address to 5832 - read data.
1030 F5FA 8E04C5    LOOP1  STX PIA        Write address.
1040 F5FD 20A7F5           JSR RETURN  Waste time.
1050 F600 AD06C5           LDA PIA+2     Load data.
1060 F603 293F             AND #$3F      HOLD/READ/15 only.
1070 F605 E005             CPX #5        24/pm/am.
1080 F607 F004             BEQ DAN
1090 F609 E008             CPX #8        Leap year.
1100 F60B D002             BNE BACK
1110 F60D 2933      DAN    AND #$33      HOLD/READ/3 only.
1120 F60F 60        BACK   RTS
1130                ;
1140                ; Enable pulse from D# on 5832.
1150 F610 A90F      RUPT   LDA #15     Pull address lines high.
1160 F612 8D04C5           STA PIA
1170 F615 A920             LDA #32     Full READ high.
1180 F617 8D06C5           STA PIA+2
1190 F61A 60               RTS
1200 F61B A8F5      LOHI   .WORD CLOCK
```

References:

OKI MSM5832RS data sheet. 65Ø2 Assembly Language Programming by L. Leventhal, pages 11-15 etc.

LETTERS

ED:

I have been asked by Ian Mutch, Brisbane, Australia, to attempt to document for PEEK readers the modifications and details of changes to D & N Micro Products to allow them to Run at -

2 Mhz operation with option for WAIT line.

Multi-user High order address line operation.

2716 EPROM in lieu of 27Ø8 (old 3 supply type).

and generally tidy up a few items that are potential trouble spots.

Let me start by saying that the D & N 16Ø5 CPU and Floppy controller works as specified without change and so does the 16ØØ (OSI 555 Jungle Board =). However, while the 16ØØ board will extend an OSI system that has a 51Ø or similar CPU board, D & N does not have any combination of boards to allow you to build up a Multi-user system or to upgrade a non 51Ø style system. But as supplied, the 16ØØ board is an excellent board for the addition of printers and bit of RAM. So, too, is the 16Ø5 board ideal as an upgrade to a floppy system. However, a lot of problems cropped up in putting together a working system. Let's tackle the 16Ø5 CPU/Floppy Board.

The 16Ø5 board as supplied has provision for a 65Ø2 CPU at 1 Mhz. A Serial Port at $FCØØ as #1 (RS232 levels). An OSI Floppy Disk controller, A 3 supply 27Ø8 EPROM, and a Motorola Baud rate generator I.C. for the Baud rates (switch selectable) for #1.

The 27Ø8 EPROM just had to go. Difficult to buy, difficult to program, and not compatible with other OSI style monitors that are available. The circuit details are shown in Fig. 1. D & N emulated OSI in decoding the top three (as required) pages of memory ORing them to provide the EPROM CS (Chip Select). This presented a bit of a timing problem and in any case because the #1 port is at $FCØØ, you would only ever want the top three pages, i.e.,$FD,$FE,$FF.

A 2716 device is 2K of Memory. So, consider that address line A1Ø normally on pin 21 controls whether we address the upper or lower 1K out of the device, i.e., if A1Ø is a Logic Ø, then we will address data in the lowest part of the 2716. If A1Ø is a Logic one, then we address the upper 1K.

Logically then (no pun) if we jumper connect pin 19 to either Ø (ground) or 1 (+5) then we can select one of two different programs within the 2716. By ensuring that the correct pages in the EPROM are programmed, I was able to have a video based monitor in the lower part of the EPROM, and a flick of a switch to change pin 19 to Logic 1 allowed me to have a serial based monitor in the upper part. A table appears as Table 1 to show how the pages of the EPROM are organized.

The Motorola Baud rate generator I.C. (1411) and 1.832 Mhz crystal pair cost, back in late 1982, about $25 if you could find a supply. As we were using the 555 board on some systems and the D & N 1600 board, I simply used line 13 on the backplane to extend a selected baud rate onto the bus from the 1600 board. Fig 2 shows how. Cut a trace to disable the generator line on the 1605 board and connect the #1 Tx and Rx clock to line 13. This freed up about 4 square inches of board space where the B/R gen' was located. In Australia that's 5Øx5Ømm., not very big as Australia goes, but enough room to install a small piggyback printed circuit board that has a WAIT state controller on it. The circuit for this is Fig 3. Simply a WAIT circuit as used

Continued

TABLE 1

Hex (φ) Page	PAGE	CONTENTS
$FF	7	RESET VECTORS MONITOR
FE	6	/ HARD DISK BOOT
FD	5	FLOPPY BOOTSTRAP
$ FC	4	NOT USED FILL WITH DATA "FF"
$ FF	3	RESET VECTORS MONITOR
FE	2	/ SCREEN DRIVERS
FD	1	FLOPPY BOOTSTRAP
$ FC	φ	NOT USED FILL WITH DATA "FF"

PIN 21 = "1" (+5) SERIAL SYSTEM

PIN 21 = "φ" (GND) VIDEO SYSTEM

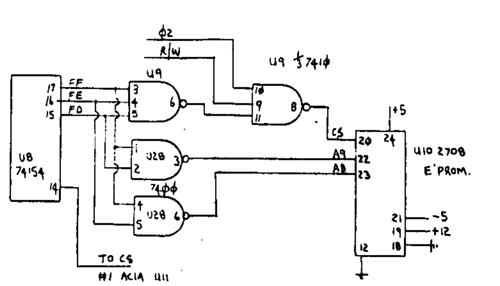
LAST 2 bytes of OSI monitors changed to give new IRQ vector of "D800"

ie. FFFE = φφ
FFFF = D8

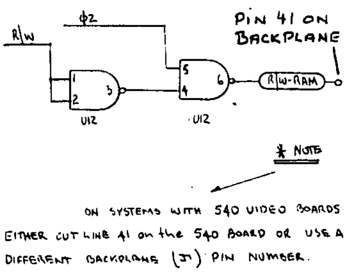
TABLE 2

DELETE
R25 - R33
R35 - R34
R15,16,17
R22,23,24
R18 - 21
ALL 47Ø ohms

FIG.1

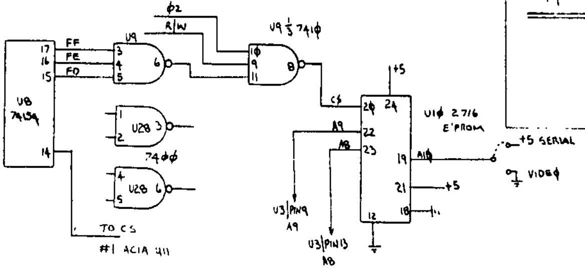


ORIGINAL D & N ROM SELECT FOR 2708

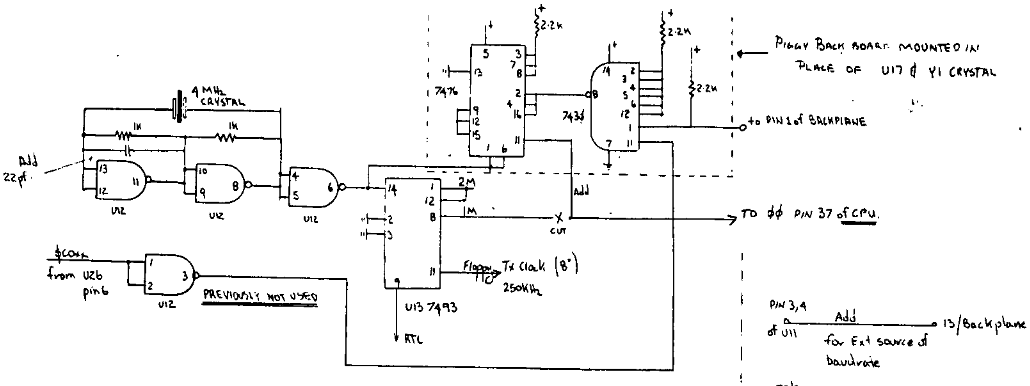


ON SYSTEMS WITH 540 VIDEO BOARDS EITHER CUT LINE 41 on the 540 BOARD OR USE A DIFFERENT BACKPLANE (J1) PIN NUMBER.

R/W RAM SIGNAL



REVISED CIRCUIT FOR 2716



CHANGES TO GIVE 2MHz OPERATION WITH 1MHz SWITCHING (WAIT LINE)

FIG.3

PIN 3,4 of U11 — Add for Ext source of baudrate — 13/Backplane

J3/3 — Add to provide system reset — 14/Backplane

FIG.2

on this and many other non-OSI system detects a Logic condition (Logic Ø) on a Bus line that is generated by some device while it is being selected. OSI allocate line 1. When this line is detected as going low, it causes a 74LS3Ø to change state, the output is driven to Logic 1 which in turn causes a 74LS76, which before was dividing only by 2, (4MHz to 2MHz) to now divide by 4 down to 1MHz for the duration of the "Wait" line being low (Logic Ø). The circuitry is arranged so that no bumps are added during switching from 2 to 1 MHz. This allows us to use slow devices on the system. For example, in a Multi-user mode we could use an on Hand memory card that is normally only reliable at 1MHz as a not very busy user. Then use a newer but more reliable Card (probably more expensive) at the full 2 MHz for a busy user. Obvious advantage is to save money when upgrading to Multi-user.

The 16Ø5 had (in my opinion) a couple of design errors which caused bus loading problems on some systems. I did discuss these with D & N who agreed that some pullup resistors may cause problems, but (I agree) only under certain conditions. Let me explain, - normally the ribbon cable to the disk drives is terminated with 15Øohm resistors at the last physical drive. All manufacturers recommend OPEN Collector drivers from the Controller (i.e., CPU end) onto the lines. This allows typically up to 1Ø feet of line at optimum performance. OSI use 7417 O/C bus drivers with 47Øohm pullups on each line. D & N used 74LS367 bus drivers (they have provision for tri-stated outputs but not used here) with 47Ø ohm resistors. While the 74LS367 is an excellent device it just couldn't drive the 47Øohm and 15Øohm in parallel reliably. Also, the address buffers use 74LS367 with 47Øohm resistors. OSI terminate each bus line on their motherboard with 47Øohm too. Again, as the number of boards in a system (multi-user) increased, we ran into reliability problems. Particularly on a 17 slot OSI backplane as it has 47Ø ohms resistors at each end of the bus lines that made 3 x 47Ø in parallel. A simple fix was just to leave the 47Øs of the D & N board (see table 2). Also, the 74LS367 are NOT open collector devices so pullup to Logic 1 by themselves.

One added bonus of rearranging the 27Ø8 was to free up a couple of gates out of package U28 (a 74ØØ). This allowed me to wire up and generate a new signal onto the backplane on Pin 41. This is called R/W-RAM (i.e., Read/Write RAM). This removes reliability timing problems on static memory cards. The simplest way to use it is to cut the normal R/W line as it goes into the static memory cards and connect the board onto the new line in lieu of the R/W. This is very beneficial to getting operational NMOS and CMOS 48K RAM cards on OSI 17 slot backplanes. Also, I found by using a storage Oscilliscope a difference in timing for the 51Ø board and other CPU boards. The result in practice was that the 51Ø board would work sometimes with CMOS 6116s, never reliably with NMOS equivalents and hardly ever on a 17 slot backplane with either devices. The addition of a 22 pf capacitor at pins 13 - 11 of U12 will improve crystal oscillator starting. I lay no claim to the R/W RAM signal idea as both Rockwell and Synertec take great pains to point out the special need of such a line for static memory devices. I would certainly welcome comments (crits) from any others on this matter, in particular from any ISOTRON designers (nee OSI).

I anticipate that this letter runs off at the mouth a bit (my style) so I will carry the description of the 16ØØ (555 I/O) board over into a second letter. The second installment will cover a description of the hardware needs and operation to implement multi-user operation, and the methods to do this with D & N products. If we all hold together, I will round off the story with my own product description, (unsolicited) of a CPU I/O do-all board that I have available for the 48 line bus that uses CMOS memory for 4 x 2K of RAM, 6 serial ports, Centronics Printer, CPU, full 4K of MONITOR ROM with port masking, 16 Pin I/O bus, OKI MSM 5832 CMOS clock battery supported, and pagination to allow on board BASIC-IN-ROM (in a 2764) or other utilities to swap with the upper 8K of 48K RAM with software control. All hardware 1ØØ% OSI compatible.

David Tasker
Tasmania, Australia 73Ø3

* * * * *

ED:

I was glad to see the review of DOS/65 in the July and August issues of PEEK. In general I have no specific argument with the review, however, I would like to add a few comments.

1. DOS/65 has now been adopted by Rockwell as the standard OS for their products. In conjuction with Rockwell, the system has been ported to the Rockwell Design Center and is being ported to the AIM/65 and System 65 by Rockwell. In the long run I expect that more DOS/65 compatible software will now be available!

2. The system has been used with hard disks and provides exceptional performance and capability. In my main development system (not an OSI machine), I have two eight-inch drives, two five-inch drives and a 19 megabyte Winchester organized as two logical devices.

3. For the most part DOS/65 files are also compatible with CP/M 2.2. As the reviewer points out, the problem with OSI is their very non-standard disk controller (same with Apple) that does not allow diskettes to be interchanged. If MODEM.ASM is used for file transfer over a serial line, the files can be interchanged with any CP/M system.

4. I have implemented Microsoft BASIC for the system. I have been unable to get Microsoft to listen to me and hence cannot yet distribute it as part of the package or as a stand alone option. If that log-jam is broken, I will let all DOS/65 users know as soon as possible.

5. The 1K to 16K block length is a function of the Disk Control Block (DCB) that is under user control. For all distributed systems it is 1K but can be altered if desired by the user. This kind of change is most useful for those having hard disks.

6. The problems reported by the reviewer are being investigated. The problem with FILESTAT is probably a problem with duplicate array dimensioning in FILESTAT and not a problem with BASIC-E/65 itself. The reason for the ^R not working is not understood but will be checked. The compile-time option problem with BASIC-E/65 has been fixed. EXP will be added to the BASIC-E/65 documentation if it is missing.

7. I agree that two drives is really a much better system configuration than one drive.

Thank you for the review.

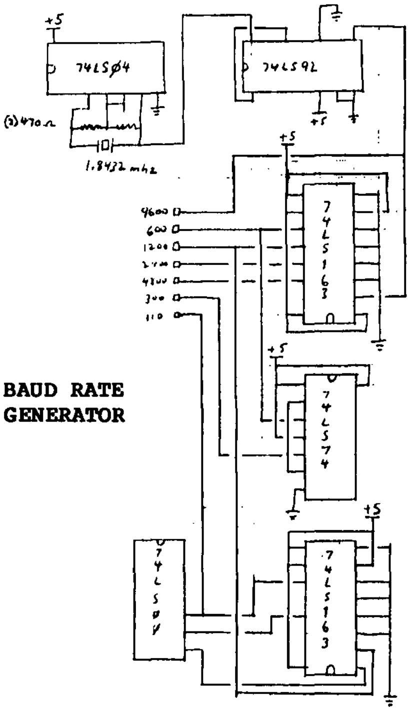
Richard A. Leary
Micro Systems Technology
Norristown, PA 194Ø1.

* * * * *

ED:

Recently I was reviewing the baud clock circuits used in the Challenger series, and decided to design a better one. The short cut, I figured, would be to use a single dedicated baud-rate generator IC. One look at their price tags, however, quickly changed my mind.

So I fetched my pocket calculator and began dividing various crystal frequencies. The results are illustrated by the enclosed schematic. A number of mail-order IC dealers sell the 1.8432 mhz crystal for about $5. This is the single most expensive part, but it's about half the price of most baud-rate generator chips.



BAUD RATE GENERATOR

The divider chain uses off-the-shelf TTL chips whose total cost should be less than the cost of the crystal. As shown, all of the baud rates are right on the nose, except for the 110 which works out to an acceptable 109.

If any of your readers have had any experience with the Exatron Stringy Floppy tape storage system, I'd like to correspond with them.

Bruce Showalter
857 Cedar
Abilene, TX 79601

* * * * *

ED:

I recently acquired an OSI computer and am now a regular subscriber to PEEK(65). My problem is I have what must be an odd model, CD 8 S DF, for which I haven't seen anything written. It looks much like a C 8 P DF but, I understand, the S is for serial, it does have a serial terminal as device 1. How do I relate program listings and advertisements for hardware and software to my model? Would games run on a serial terminal? What about PEEK and POKE locations?

Two specific questions. How can I get into money mode on a data field in DMS? I have the 9/79 version of OS-DMS Nucleus, updated by Ron Fial. If money mode is not possible, then would I be able to enter leading spaces in order to justify a column to the right when entering dollar amounts in a data field? I thought of using a dollar sign and then spaces, but I thought this might mess up the statistical part of the report writer when totaling the figures.

Another question, I've seen ads for 6502A and 6502B chips (2 and 3 Mhz) and wonder if it is possible (or advantageous) to replace my 1 Mhz unit with either of these? What would I need to change besides the chip and the crystal? I have two 1Mhz memory boards and one 2 Mhz memory board.

I understand that one way to "pay" for help is to help others. I'm so new I haven't much to offer, however, I have written a program that calculates look-angles to syncronous satellites, if anyone would be interested.

Dwight Finger
Anchorage, AK 99504

Dwight:

Your CD 8 S DF is a new one to us too, but that doesn't mean you are not right. As you say, it's probably just the serial version of the C8P DF. As such, anything written for the C2-OEM (220) and up, should be on target, programwise. In short, anything written for OSI serial machines. Only those PEEKs and POKEs addressing video vs serial terminal systems would change.

We don't know what Ron has done to DMS, but generally speaking, money mode is not available. With customizing, it could be added to the report writers. Adding leading spaces won't help - they are truncated. $_ _10.00 will make the columns look right, but you are in trouble for totals. All DMS entries are strings. For OS65U see page 14, item 6, this issue.

2 Mhz will double your CPU speed, but you must make sure that your memory will handle 2 Mhz too. Have the 1 Mhz boards checked. Many of the chips may pass the 2 Mhz test.

Let's hear more about "look-angles."

Peek Staff

AD$

C-2 OEM (2 cases), dual 8" drives, 48K RAM, serial & parallel ports, printer interface, RS-232 for terminal, 65D 3.3, 65U 1.42, OS-DMS Nucleus, Sort, Planner Plotter, A/R, A/P, G/L, Inventory I, Inventory II, Payroll, Purchasing, Query, Education, OS - AMCAP Small Business Accounting System, WP6502 Word Processor, DQ Mail, Plot Basic, Home Control - $1800.00. C-2 converted to C3-S1 (2-cases), Dual 8" drives, 56K RAM, 510 3 Processor 2 Mh CPU, Centronics Parallel Port, Diablo Parallel Port, 5 serial cluster ports, RS-232 for terminal or configure for multi-user, same software as above - $2100.00. UCSD PASCAL/FORTRAN system - $200.00, extra boards and much software - send S.A.S.E. for complete list. Thomas Technical Service, RD #1, Box 135, Linden, PA., 17744. (717) 398-1893 evenings.

* * * * *

FOR SALE: C8P complete system, two 8-inch disk drives (ss), 48K, Zenith green monitor, Centronics printer #779, all manuals, OS-65D V3.2, OS-65U, WP6502, DQ-MAIL, SARGON II, OS-DMS SOFTWARE, GAMES, DISKS and other software. $750 firm. In Maryland (301) 263-1560.

* * * * *

OSI C2P single disk not working DOS 65D V3.2, $300 or best offer. D. Starshine, 1025 N. Rodney, Helena, MT 59601, (406) 442-5720.

* * * * *

Send for free catalog, Aurora Software, 37 South Mitchell, Arlington Heights, IL 60005. Phone (312) 259-4071

* * * * *

Good prices on collection of OSI equipment and accessories. Send SASE for complete list. Ricky Peterson, 206 Pine Valley, Warner Robins, GA 31093.

* * * * *

FOR SALE: 1 - C3A 48K computer with dual sided 8" disks (1.2MB). 1 - C2OEM 48K computer with dual sided 8" disks(1.2MB). Various software and manuals to go with both systems. Phone 303-384-9030 or 303-384-4221 ask for Danny. Write Tra-Sta Data Systems, Box 427, Swink, CO 81077.

* * * * *

FOR SALE: C28P-MF, 48K, 2MHz, RS Lineprinter 1, 65D3.3, Planner Plus, OSI WP3-1, $650. C3 w/Hazeltine 1500 w/o drives, $400. Sanyo 7.5 MHz RGB Monitor & modified 540, $400. For details call Craig Borst (616) 399-3109.

* * * * *

OSI 350 JJ 6 User Computer with 2 80 Megabyte Hard Disc Drives. Bought in 1983. $14,000. Also, 4 visual 50 green screen CRTs at $450 each, NEC Spinwriter 7730 with tractor $2,050, NEC Spinwriter 7710 with tractor $1,800. All basically new. Also, a C3-OEM Computer $1,000. Feel free to make offers. Call Ron at 1-509-248-7512.

* * * * *

FOR SALE: OSI C3-B with dual single sided 8 inch floppy disks, 74MB hard disk, Centronics 702 parallel printer (and interface), (4) 48K user ports, and (2) Hazeltine 1500 CRTs. Various software packages included. Make an offer. We will consider a trade for IBM PC or PC compatibles. Call or write: Lee D. Hoffmann, FEECO International, Inc., 3913 Algoma Road, Green Bay, WI 54302, 414-468-1000.

PEEK (65)
The Unofficial OSI Users Journal

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

DELIVER TO:

[illegible]

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. Due to U.S. bank surcharges, all funds payable to PEEK (65) must be in **U.S. Dollars** and be drawn on a **U.S. Bank** or be an **International Money Order.**

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.**
( ) $27.00 Enclosed. South & Central America, Europe & all other. **Surface.**

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 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

Vol 4, 1983

( ) 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 5, 1984

( ) JAN #1 ( ) FEB #2 ( ) MAR #3 ( ) APR #4 ( ) MAY #5 ( ) JUN #6
( ) JUL #7 ( ) AUG #8 ( ) SEP #9

INDEXES ARE INCLUDED IN THE JAN. & DEC. 1981 AND DEC. 1982/3 ISSUES