PEEK (65)

The Unofficial OSI Users Journal

AUGUST 1985
VOL.6, NO.8

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

INSIDE

Column One

Each month we try to give you the latest on the manufacturing front. Try though we may, this month is a void. But if you stop and think about it a moment, even Big Blue doesn't make an announcement every month! It's just not reasonable to have twelve hot new items every year.

What we do have is the long overdue report on the OSI 700 series machines. Granted, they are mostly OSI's words, but since we have not had our hands on one yet, we will get started this way. There is also some information on corporate and people matters that should answer many frequently asked questions.

One item has reached us in a timely fashion. Paul Chidley and the boys at TOSIE have come up with a new paddle board for the floppy cable that not only is better than OSI's, but allows simple conversion for most any drive to replace the old MPI. Although we haven't checked it all out yet, the thought of putting two half-highs in the OSI floppy box certainly sounds enticing. In the meantime, we understand that the bare boards are available from Paul and/or TOSIE for about $20.

While on the subject of floppies, the saga of "floppy control" continues in this issue. Dave Pompea gives us the wherewithal to install his mod on the 4P (although not yet fully tested) and Bob Ankeney gives us his tried and well proven quick fix. Once again, we say, take your pick, depending upon your degree of expertise, but do install one or the other. You won't regret it!

There is also more on FORTH this month. Charles Curley's follow-up article should give you enough information to determine if FORTH is for you.

As more of you upgrade to disk, the interest continues in converting old cassettes. Jim McConkey's article on the conversion of DEPTH CHARGE provides a lot of insight as to this particular program and establishes the ground rules for others too.

From Down Under comes John Whitehead's report on DAGUB 3. Here's a monitor that has some 5,000 copies behind it and, at last, we can share its inner workings.

Remember RESOURCE? If you don't, it is highly suggested that you go back to the May and June issues and re-read it, particularly now that Dana (Skip) Skipworth has spent many days smoothing over the rough spots to come up with the details and how-to's to get you going.

If the going is still too tough for you, tune in to the current episode of Beginner's Corner. Leo Jankowski comes through again with some very down to earth tips on structure and writing - of course, with a couple of "winners" thrown in for good measure.

For the OSU programmer, Roger Clegg has done it again with a whole bunch of vital programmer helpers. I can remember all too well how long it took to get my first dollar handler sorted out. So I know how much time you will save by using these vital subroutines in your next effort. But please don't let Roger feel like the lone ranger, send us your little "wiz-bang" too.

For the statistical buffs, there is plenty to think about in Richard Puckett's Optimization Algorithm - not to mention that this is a first for OSI. I don't know that it will put the mainframes to shame, but at least it gets a tough job done well.

Lastly, we end on a sad note and with the greatest regret inform you, that after a long illness, Mr. Ian Eyles died on the 8th of June. We received this unfortunate news the morning after the July issue went to press, informing PEEK readers of Ian's ill health.

Ian, with two helpers, was the instigator of KAOS, and with his wife Rosemary, became the driving force that shaped the Australian OSI club and its Newsletter. To Rosemary, his family and many OSI friends, we offer our deepest condolences, and with them mourn the passing of a truly marvelous man in Ian Eyles.

Eddie

USING RESOURCE FOR MAPPING MACHINE LANGUAGE CODE WITH THE C2-4P MF & OS65D V3.1

By: Dana Skipworth
2055 W. 87th St.
Cleveland, OH 44102

The RESOURCE ARTICLE appeared in the Feb./Mar. issues of PEEK(65). It was written for the eight inch disks and is not directly applicable to the five and one quarter inch disks "as is", but it is easily modified for use with the C2-4P or C4P systems.

This article will present a step by step procedure that will create and load disk files, using RESOURCE.

There are several places in this article where duplications of the original article appear, making this article easier to follow. The original article should be read and understood, as this article does not replace the original, it is only an adaptation for users of 5 inch disks.

The ASAMPL program on your OS-65D system disk will be used as a sample program because it is a short program and reduces the number of disks needed to illustrate the use of "RESOURCE". Larger programs can use up to five disks including the disk with the Resource Programs.

Create three copies of your system disk and delete all file entries. One disk will be used for a PROGRAM DISK, and two disks will be used as TEXT DISKS.

Enter the Resource program names into the directory of one of the disks. (THIS WILL BE YOUR "PROGRAM" DISK!) Simple names like Pass for Resource 1, Pass 2 for Resource 2, etc. work well and are easy to remember. Create a "Fiload" file with three stacks. Load the Resource programs into memory one at a time and make the changes listed below. As the changes are completed, save the programs on the new "PROGRAM" disk.

RESOURCE 1, Lines 60,70 and 80

```
60 POKE 9000,00:POKE 9001,136
70 POKE 9006,00:POKE 9007,136
80 POKE 9008,00:POKE 9009,144
```

RESOURCE 2, Lines 150,160, & 170

```
150 POKE 9000,00:POKE 9001,136
160 POKE 9006,00:POKE 9007,136
170 POKE 9008,00:POKE 9009,144
```

RESOURCE 3, Lines 90,100, & 110

```
 90 POKE 9000,00:POKE 9001,136
100 POKE 9006,00:POKE 9007,136
110 POKE 9008,00:POKE 9009,144
```

RESOURCE 4, Lines 80,90,100, 690,700,710, & 720

```
 80 POKE 9000,00:POKE 9001,136
 90 POKE 9006,00:POKE 9007,136
100 POKE 9008,00:POKE 9009,144
690 PRINT TAB(10)"Z PAGE CROSS
          REF.FILE:"ZF$
700 PRINT TAB(10)"Z PAGE
          EQUATE FILE:"ZE$
710 PRINT TAB(10)"PASS 4
          COMPLETED"
720 PRINT:PRINT:END
```

DELETE LINES, 730 THROUGH 1210

CREATING THE TEXT DISKS AND FILES

1. Using your system disk, (NOT THE PROGRAM DISK) load the ASSEMBLER and type !LOAD ASAMPL.

2. Type A3, the ASAMPL program is now assembled in memory at $1600. Type EXIT, RE EM.

3. Remove your system disk and insert one of your new TEXT DISKS.

4. Your first file will use tracks 13 through 38. This provides more than enough room for your file and it will be trimmed to size and named later.

5. Using the EXTMON and the commercial "AT" sign, set the buffer values as listed below:

HEX ADDR	HEX VALUE	COMMENTS	
2326	7E	Buffer starting address	(LO)
2327	32		(HI)
2328	7E	Buffer ending address	(LO)
2329	3A		(HI)
232A	13	First track of file	(BCD)
232B	38	Last track of file	(BCD)
232C	13	Current Track	(BCD)
232D	00	Dirty buffer flag	
23C3	7E	Disk output address or-	(LO)
23C4	32	Current buffer address.	(HI)

NOTE!- If you make a mistake in the above procedure, stop,- and start over again from $2326.

6. Still using the EXTMON,type !IO ,22 and <return>.

7. Start the disassembler with Q1600 <return>, on larger programs as with this one, hold the linefeed down until the last line of the program has been disassembled.

8. After the disassembly is completed, create an error to turn off the output to the disk by typing !XIT <return>.

9. Type W!XIT>327E,3A7E (searches for end of file). If all is well a message will appear...VVVV/21.... where VVVV is the address of "!" in the expression !XIT.

10. Using the commercial "AT" sign and quotes, check for the following:

```
VVVV/21    =!
VVVV+1/58  =X
VVVV+2/49  =I
VVVV+3/54  =T
```

Make the following change:

VVVV/21 0D

The new SOURCE file is now properly terminated.

11. Make following changes:

23C3/YY 7E
23C4/WW 3A

12. Check the track number at $232C/TK. NOTE! This will be the last track of your source file.

13. Type !IO ,22 <return>, the source file is now on the disk, starting on track 13 and ending on track "TK".

14. Use a nondestructive create program, or delete line number 20290 in the OSI Create program and create a SOURCE file, from track 13 to track "TK", plus one (this should take two tracks).

15. Restore line number 20290 to your Create program and create the following files in addition to SOURCE, keeping all files on the same disk. Be sure to run the Zero program on each of the new files before you use them.

```
SCRACH - 3 TRACKS
SYMBOL - 2 TRACKS
EQUATE - 2 TRACKS
REF-J  - 2 TRACKS
REF-B  - 2 TRACKS
REF-M  - 2 TRACKS
REF-Z  - 2 TRACKS
```


published monthly

Editor - Eddie Gieske

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

Subscription Rates	Air	Surface
US		$19
Canada & Mexico (1st class)		$26
So. & Cen. America	$38	$30
Europe	$38	$30
Other Foreign	$43	$30

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 (301) 363-3268

ZQUATE - 2 TRACKS
OBJECT - 3 TRACKS
STORE - 3 TRACKS

At this point you are ready to use the Resource programs you entered on disk.

16. Load Resource 1, remove the program disk and insert the TEXT disk, type RUN <return>.

17. Repeat this procedure for all 4 Resource listings. Note that there are 4 files to be created with Resource 3.

The ultimate use of the resourced program is up to you, but to prove the integrity of the program, use the following procedure to merge, files Zquate, Equate, and Object for processing by the assembler.

LOADING ALTERNATE ONE

18. Create two buffers for the following program and enter it in the FILOAD file on the PROGRAM DISK.

```
10 REM-LOADER OF FILES
20 POKE 2972,13:POKE 2976,13
30 DIM X$(1500)
40 INPUT"STORAGE FILE NAME";
   OF$
50 INPUT"NUMBER OF INPUT
   FILES";N
60 FOR I=1 TO N
70 INPUT"FILE NAME";F$(I)
80 NEXT I
90 REM-
100 Z=1
110 FOR I=1 TO N
120 DISK OPEN,6,F$(I)
130 INPUT#6,X$(Z)
140 IF X$(Z)="XIT" THEN
    DISK CLOSE,6:GOTO 190
150 IF LEFT$(X$(Z),1)<"1" OR
    LEFT$(X$(Z),1)>"9" THEN
    190
160 PRINT X$(Z)
170 Z=Z+1
180 GOTO 130
190 NEXT I
200 DISK OPEN,7,OF$
210 PRINT:PRINT"WRITING TO
    DISK":PRINT
220 FOR R=1 TO Z
230 PRINT #7,X$(R)
240 PRINT X$(R)
250 DISK PUT
260 NEXT R
270 PRINT #7,"XIT"
280 PRINT #7,"E"
290 PRINT #7,"E"
300 PRINT #7,"E"
310 DISK CLOSE,7
320 REM-
330 DISK CLOSE,6
340 REM-
350 PRINT"MISSION ACCOMPLISH-
  ED."
 END
```

?#19-Remove the PROGRAM DISK and insert the TEXT DISK. With the Fiload program still in memory, type RUN..

Type in STORE for the storage file name.

The first input file is Zquate.

The second input file is Equate.

The third input file is Object.

The three files are now merged and written to the file STORE.

The next step is to load this file into indirect memory, for transfer to the assembler. The size of your memory will dictate the size of the indirect memory. Use the following table as a guide for use with your memory.

MEM SIZE	POKE 9554 & 9368 WITH DECIMAL	START. ADDR.
24K	80	$5000
32K	96	$6000
40K	112	$7000
48K	128	$8000

20. Boot up using the text disk.

Enter both POKEs on the same line and <return>.

Type EXIT and <return>.

Type CA <address>=TK,1 and <return>.

TK=starting track of the STORE file.
(address)=the indirect memory starting address.

Type ASM (loading the assembler).

Type 10 *=$1600 and <return>.

Type Control/X

The merged files are now in the assembler and several errors will be noted at the bottom of the screen.

END OF ALTERNATE ONE

21. Type <return>.

Type P <return> (prints out the Source program).

Type A <return> (prints out the assembled program).

Type A3 <return> (the program is assembled in memory, starting at address $1600).

Type !GO 1600 <return> (and the message printed is):

---THIS IS A SAMPLE PROGRAM---

Anything else and you have an error somewhere.

LOADING ALTERNATE TWO

Alternate two will handle larger programs because memory space is not used for indirect memory, but it is a little more involved than alternate one.

To use alternate two, substitute this listing for the listing in ALTERNATE ONE!

A-Load the assembler and EXIT to the EXTMON.

Using the commercial "AT" sign, set the following buffer values.

ADDRESS($)	ADDRESS(D)	VALUE
2326	8998	00
2327	8999	50
2328	9000	00
2329	9001	58
232A	9002	N
232B	9003	M
232C	9004	N-1
232D	9005	00
23AC	9132	00
23AD	9133	58

Memory addresses $2326 through $2329 contain the offset addresses for the disk buffer.

N is the first track of your file.

M is the last track of your file.

N-1 is the current track.

B-Re-enter the assembler.
C-Input your file by typing !IO 20 <return>.
D-Repeat steps A through C until all files are loaded.

Your files are now merged and in the assembler. Be sure to inspect them carefully before assembling.

END OF ALTERNATE TWO

★

BEGINNER'S CORNER

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

The idea for this program grew out of a real need. From time to time, I have wanted a simple program that would write short sequential data files to disk. The data in these files could then be loaded and used by other programs.

One example of such an arrangement is in the "Stop the Dwarve" program when it reads a sequential file of words.

Nothing is done to those words; they are merely used as data by the "Dwarve" program. Incidentally, the "Dwarve" program is an excellent way of teaching vocabulary and spelling. Words can be guessed even if never seen before - for example, foreign place names. Do not use the programs for tests in vocabulary, yuk! The Sequential File Editor program can be used to create the three files of words required by "Dwarve." The files could contain words suitable for a particular reading age. Where to find the words? In the appropriate books!

A PLAN

Beginner programmers are sometimes urged to draw a flowchart for a program before writing it. The latest fashion is for structure diagrams and top-down programming. On the fringes are structured flowcharts, Nassie-Schneidermann diagrams and the "Jackson" method. All these methods have their problems (what is perfect?) and they are difficult to learn and use. I have always found that flowcharts are best drawn when the program is finished!

The diagram method used is not so important. What is sometimes forgotten is what stands behind all the technicalities: the need to have a plan before writing a program. And by that I don't mean a detailed description of the algorithms and everything else that will be in the program, as would be demanded by a structure diagram. Doing that is tedious and extremely time-consuming. A non-hierarchial description/picture of the program structure should be sufficient. Specific algorithms can always be diagrammed later, as the need arises.

The Seq. File Editor program, see listing, makes full use of the screen editing commands found in OS65D 3.3. The program will read up to "X" entries from a sequential disk file. The maximum length "X" is set in line 120. After the disk read, null entries are deleted from the end of the file stored in RAM. The file can be inspected, edited and saved back to disk. Files can be merged by reading them one after the other from disk.

WARNING!

The program will only read files from tracks that have been previously zeroed. When

```
10 REM Sequential Data File Editor. (c) LZ Jankowski 1985
20 REM All Rights Reserved 1985. Version May 12 '85
30 :
40 POKE2972,13:POKE2976,13:POKE2888,0:POKE8722,0:POKE2073,173
50 :
60 CLEAR:PRINT!(21)!(25):C=0:L=0:Y=0:N=0:K=0:Y$="":F$="-------"
70 F1$="-------":F2$="-------":E$="No disk errors":ER$=E$:Y$=CHR$(135)
80 :
90 FORC=1TO4:U1$=U1$+Y$:NEXT:FORC=1TO6:U2$=U2$+Y$:NEXT:U$=U1$+U2$
100 Y$=CHR$(145):FORC=1TO21:U3$=U3$+Y$:NEXT
110 Y$=CHR$(144):FORC=1TO21:U4$=U4$+Y$:NEXT
120 X=2000:DIM D$(X):L$=CHR$(8):L1$=L$+L$+L$
130 L2$=L1$+L1$+L$+L$:RI$=CHR$(16):C$=CHR$(27)+CHR$(28):B$=CHR$(96)
140 N1$="null":N2$="NULL":G=13026:D=50:F5=55:T2=32:CU=128
150 :
160 S$(1)="Load a file":S$(2)="Save file":S$(3)="View file"
170 S$(4)="Edit file":S$(5)="Append to file"
180 S$(6)="Load SORT program":S$(7)="Zero out file"
190 S$(8)="U> End":NF$="* File not found *":SR$="SQSORT"
200 FORC=1TO7:N$(C)=STR$(C)+"> ":NEXT:ZZ=0:POKE49154,0:GOTO240
210 :
220 GOSUB5370
230 REM ---------------- MAIN MEMU ---------------------------
240 PRINTC$
250 POKEG,T2:T=22:Y=1:V=2:PRINT&(T,Y)"---------------------"
260 PRINT&(T,Y+1)"!    SEQUENTIAL    !"&(T,Y+2)"!    "U$"    !"
270 PRINT&(T,Y+3)"! DATA FILE EDITOR !"
280 PRINT&(T,Y+4)"---------------------":Y=8:T=6
290 PRINT&(T+1,Y)U3$&(T+30,Y)U3$
300 PRINT&(T+1,19)U4$&(T+30,19)U4$
310 Y=9:PRINT&(T+9,Y)"MENU"&(T+37,Y)"Status"
320 PRINT&(T+9,Y+1)U1$&(T+37,Y+1)U2$:H=10
330 TRAP7000:FORC=1TO7:PRINT&(T,H+C)N$(C)S$(C):NEXT
340 PRINT&(T+1,H+C)S$(8):POKEG,63:T=38
350 PRINT&(T,H+1)F$&(T,H+2)F1$&(T-1,H+4)L"entries exist"
360 PRINT&(T-1,H+5)X-L"entries free"&(T,H+6)SR$&(T,H+7)F2$&(T,H+8)ER$
370 PRINT&(7,H+4+C)"Choice ";:GOSUB5400:IFY$="u"THEN5000
380 POKEG,CU:ER$=E$:IFY=0ORY>7THEN400
390 GOSUB1520:ONYGOSUB430,520,600,800,1250,1460,1620
400 GOTO240
410 :
420 REM -------------- LOAD A FILE of DATA -------------------
430 PRINT"LOAD. Name of file   "F$L2$;:INPUTY$:IFY$="x"THEN490
440 IFY$<>""THENF$=Y$
450 PRINTC$&(16,10)"* LOADing file from disk *"&(0,0):C=0
460 GOSUB5370:DISK OPEN,6,F$:TRAP8000:L=L+1
470 FORC=LTOX:INPUT#6,D$(C):NEXTC
480 DISK CLOSE,6:C=C-1:GOSUB5370:GOSUB9000
490 RETURN
500 :
510 REM ------------ SAVE A FILE of DATA ---------------------
520 PRINT"SAVE. Name of file   "F1$L2$;:INPUTY$:IFY$="x"THEN570
530 IFY$<>""THENF1$=Y$
540 PRINTC$&(16,10)"* SAVEing file to disk *"&(0,0)
550 GOSUB5370:DISK OPEN,6,F1$:FORC=1TOL:PRINT#6,D$(C):NEXTC
560 DISK CLOSE,6:GOSUB5370
570 RETURN
580 :
590 REM ------------ VIEW a FILE of RECORDS ----------------
600 PRINT"VIEW. How many entries on screen   20"L1$L$;:INPUTY$
610 F=VAL(Y$):IFY$="x"ORF>LTHEN770
620 IFF<1THENF=20
630 PRINT&(22,10)!(15)"From entry #   1"L1$;:INPUTY$:IFY$="x"THEN770
640 N=VAL(Y$):IFN=0THENN=1
650 IFN>LTHENPRINT&(16,8)"Too large!"&(16,10);:GOTO600
660 :
670 V=2:PRINT&(22,10)!(15)"Device # 2"L$;:GOSUB5400:IFYTHENV=Y
680 IFY$="x"THEN770
690 PRINT&(0,2)!(22,62,20):FORK=NTOLSTEPF:PRINT&(0,0)!(24);:POKEG,T2
700 FORC=0TOF-1:PRINT#V,K+CTAB(10)D$(K+C)
710 IFC+K=LTHENC=F-1:NEXTC:GOTO740
720 NEXTC:POKEG,CU:GOSUB5400:IFY$="x"THENK=L:NEXTK:GOTO770
730 IFY>0THENK=K+100*Y-F:IFK>LTHENK=K-100*Y+F
740 NEXTK
750 POKEG,CU:PRINT"Again ? No"L$L$;:GOSUB5400
760 IFY$="y"THENPRINT!(21):GOSUB1520:GOTO600
770 PRINT!(21):RETURN
780 :
790 REM ----------------- EDIT ------------------------------
800 RETURN
1230 :
1240 REM ------------------ APPEND -------------------------
1250 L1=L:IFL=XTHENPRINT&(11,6)"APPEND. File is full!":GOTO1270
1260 POKEG,T2:PRINT&(11,6)"APPEND. Begin with entry #"L1+1
1270 PRINT&(19,8)"Previous entry    -->  '-'"
1280 PRINT&(19,9)"Next entry        -->  <RETURN>"
1290 PRINT&(19,11)"Null entry, type  -->  null (or NULL)"
1300 PRINT&(0,16)!(22,62,5);:GOTO1320
1310 IFL1<0THENL1=0
1320 L1=L1+1-INT(L1/X)*X:POKEG,T2
1330 IFL1=1THENPRINT&(0,0)!(24)"Entry"X" is   "D$(X):GOTO1350
1340 PRINT&(0,0)!(24)"Entry"L1-1" is   "D$(L1-1)
1350 PRINT&(0,3)"Entry"L1" is   "D$(L1):PRINT&(12,3);
1360 IFL1>9THENPRINTRI$;:IFL1>99THENPRINTRI$;:IFL1>999THENPRINTRI$;
1370 POKEG,CU:INPUTY$:IFY$="x"THEN1430
1380 IFY$="-"THENL1=L1-2:GOTO1310
1390 IFY$=""THEN1320
1400 IFY$=N1$ORY$=N2$THENY$=""
1410 D$(L1)=Y$:IFL1>LTHENL=L1
1420 GOTO1320
1430 PRINT!(21):RETURN
1440 :
1450 REM ------------Load Sort Program -----------------------
1460 PRINTC$&(16,8)"Load SORT program.":GOSUB1570
1470 PRINT&(16,10)!(15)"Load SORT. Ready ? Yes"L1$;:GOSUB5400
1480 IFY$="-"ORY$="y"THENGOSUB5370:PRINTC$:RUNSR$
1490 RETURN
1500 :
```

Continued on page 6

DOS reads a sequential file off disk, it will produce an error #D when it reads zeros (i.e. nulls) off the disk. If there is no error #D to report, then the program crashes and a return to BASIC is impossible. The TRAP command is used to retain program control after an error #D report. This is done in line 460 with "TRAP 8000". Without the trap the program would drop into Immediate Mode. A track-zeroing subroutine is provided within the program - see next month's issue.

CONSTANTS AND VARIABLES

The POKEs in line 40 enable "," and ":" and null to be accepted by INPUT. The final POKE in line 40 can be used to disable CTRL C - replace 173 with 96. In line 60, "PRINT!(21)" sets the screen for B/W.

The values of constants are set between lines 40 and 210. In line 60 often used variables are defined before anything else. The number of fields that the program will handle is "X=2000", set in line 120. L$ and RI$ are the move the cursor left and right codes. "G", in line 140, holds the address of the character used for the cursor. The new cursor value is in "CU" in line 140. Constant "F5" is the number of characters stored on an edit line in the EDIT block. (EDIT next month.)

In line 190 "SR$" holds the name of a file sort program. This could be "GSOSRT" if you use it. "GSOSRT", as provided with OS65D 3.3, seems to be a plain insertion sort - no Shell algorithm. A faster sort utility will be written in the near future.

THE MAIN MENU

A professional program not only works well but also looks good. It is worth spending some time on screen layout and making it look just right. The menu screen in "Seq. File Editor" goes one step further and displays the status of each option. For example, the name of the program saved or loaded, and whether a disk error occurred during the most recent disk access. Such information is valuable to the user. Program control is always retained by the menu block, lines 230-400.

LOAD, SAVE AND VIEW

A program should be easy to use. A simple aid that can always be provided is default

```
1510 REM ------------------ EXIT MESSAGE --------------------
1520 PRINTC$;SPC(19)"To exit, type an 'x'.":Y$="-"
1530 FORC=1TO64:PRINTY$;:NEXTC:PRINT:PRINT&(16,10);:RETURN
1540 :
1550 REM ---------- WARNING, SAVE CURRENT FILE? -------------
1560 POKEG,CU:PRINTC$
1570 PRINT&(16,10)"SAVE current file ? No"L$L$;
1580 GOSUB5400:IFY$="y"THENGOSUB1520:GOSUB520
1590 RETURN
1600 :
1610 REM -------------- ZERO OUT A FILE ---------------------
1620 RETURN
4980 :
4990 REM ------------- SUBROUTINES OTHER -------------------
5000 GOTO5420
5350 :
5360 REM STOP/START DISK
5370 ZZ=NOT(ZZOR254):POKE49154,ZZ*-1:FORW9=1TO1200:NEXTW9:PRINT
5380 RETURN
5390 :
5400 DISK!"GO 2336":Y$=CHR$(PEEK(9059)OR32):Y=VAL(Y$):RETURN
5410 :
5420 GOSUB1560:PRINT!(28)&(16,12)"To RESTART type:-  GOTO 220"
5430 POKE13026,128:PRINT&(16,14)"Bye !":POKE2073,173:GOSUB5370:END
6020 :
6990 REM ----------- TRAP 1 FOR MAIN PROGRAM -----------------
7000 PRINT&(16,10)!(15):ER$="DISK ERROR":GOSUB5370:GOTO250
7010 :
7990 REM ----------- TRAP 2 FOR ERR #D ERROR -----------------
8000 GOSUB5370:GOSUB9000:ER$="ERR #D. Don't worry!":GOTO240
8010 :
8990 REM ------ REMOVE NULL ENTRIES FROM END OF FILE --------
9000 PRINTC$&(16,10)"* Editing. Please wait! *"
9010 IFD$(C)=N$THENC=C-1:GOTO9010
9020 L=C:RETURN
```

input - the user merely presses <RETURN> in response to the requested input. One advantage to this is speed; the user is saved unnecessary typing. Secondly, a user may not know what value to type so the default option as shown on screen is an obvious choice. Lines 430 and 520 illustrate how a default file name is used when <RETURN> only is pressed. Of course, the option to actually type a file name could be taken and this can always be done.

Never present the user with a blank screen! A polite message as to what is happening will be appreciated - see lines 450 and 540. The program should also be able to cope with "funny" input, recover and print an error message - see line 650.

The "print at" command is used extensively in the VIEW block. A very smooth-running, professional-looking program can be produced with the aid of "print at", "clear to end of line" and the window facility. DOS 3.2 die-hards - have a look at the power of these and the other cursor addressing commands!

A window is set in line 690 in order to prevent the exit message from being scrolled off the screen - all scrolls occur in the window only.

APPEND

The append block also allows simple editing. The cursor is always positioned on the first character of the entry! - see line 1360. It would be nice if the entry also appeared in the input buffer to save retyping the whole entry when editing. OK, see next month's WAZZAT column.

TRAP

If you have never used the TRAP command in your programs, have a look at line 7000. The trap is set in line 330 with "TRAP 7000". On disk error, control is transferred to line 7000. This line then prints an error message and hands control back to the main menu. Dropping out to immediate mode is prevented. Users are saved much anguish and their language stays clean!

If you are running "HOOKS" have a look at the program in this issue's "WAZZAT" column; it can be merged with "Seq. File Editor." The changes required after the merge are in line 5420, i.e., add "GOSUB 1560" and the line number onto the end of the line. This month's listing is self-contained and can be used immediately. Before typing in the program, create 1 buffer for the sequential file. All REM and spacing lines can be omitted. Remember, "Seq. File Editor" will only load those tracks that were zeroed before data was written to them.

★

A COMPACT OPTIMIZATION ALGORITHM

By: Richard H. Puckett
706 Clarmar Street
Johnson City, TN 37601

A key part of many computer applications for scientific, engineering, or operations research purposes is finding the maximum or minimum of a function. Such routines are common on mainframes, but rare on micro-computers and rarer still on Ohio Scientific machines. The routine below has been coded, run, and tested on an Ohio Scientific C8PDF under OS-65U and is compact and fast, as such routines go. The program is based on the Broyden, Fletcher, Shanno algorithm with pseudo-code by J. C. Nash. (See J. C. Nash, COMPACT NUMERICAL METHODS FOR COMPUTERS, NEW YORK: John Wiley & Sons, Inc. 1979, pp. 159-160.)

The program will minimize any function in n variables (n specified by the user), not subject to constraints. However, the program is even more general than it seems.

First, remember that finding a maximum of a function is equivalent to findng a minimum of its negative. For example, maximizing

F = -x^2

is equivalent to minimizing

G = x^2.

Also, a problem with equality

```
1000 REM                          MIN
1010 REM MINIMIZES ANY FUNCTION WITH RESPECT TO N VARIABLES NOT SUBJECT
1020 REM TO CONSTRAINTS.  USER MUST SUPPLY THE NUMBER OF VARIABLES, N;
1030 REM INITIAL FEASIBLE VALUES OF THE VARIABLES, A(1), ..., A(N); THE
1040 REM FUNCTION TO BE MINIMIZED; & ITS PARTIAL DERIVATIVES.  (SEE
1050 REM LINES 1610, 1640-1660, & SUBROUTINE 4000.)  THE PROGRAM IS
1060 REM BASED ON THE BROYDEN, FLETCHER, SHANNO ALGORITHM AND PSEUDO-
1070 REM CODE PROVIDED BY J.C. NASH, COMPACT NUMERICAL METHODS FOR
1080 REM COMPUTERS, NEW YORK: JOHN WILEY & SONS, 1979, PP. 159-161.
1200 REM                 NOTATION
1210 REM             VECTORS & MATRICES
1220 REM A(N)    = CURRENT PARAMETER VALUES
1230 REM AL(N)   = VARIABLE VALUES WHERE FUNCTION IS LOWEST SO FAR
1240 REM B(N,N)  = APPROXIMATE INVERSE OF HESSIAN
1250 REM G(N)    = CURRENT GRADIENT
1260 REM GL(N)   = GRADIENT WHERE FUNCTION IS LOWEST SO FAR
1270 REM S(N)    = SEARCH DIRECTION
1280 REM Y(N)    = CHANGE IN GRADIENT FROM PREVIOUS TO CURRENT PARAMETER
1290 REM           VALUES
1300 REM W(N) =    TEMPORARY WORK SPACE
1400 REM           SCALARS
1410 REM AD = ADJUSTMENT IN STEP SIZE IN LINEAR SEARCH
1420 REM E1 = EXPRESSION FOR UPDATE OF B(N,N)
1430 REM E2 = EXPRESSION FOR UPDATE OF B(N,N)
1440 REM F =  CURRENT VALUE OF FUNCTION
1450 REM FE = ESTIMATED CHANGE IN FUNCTION GIVEN STEP SIZE = 1
1460 REM FL = LOW VALUE OF FUNCTION SO FAR
1470 REM MG = MAXIMUM # OF EVALUATIONS OF GRADIENT
1480 REM MR = MINIMUM RATIO OF ACTUAL TO EST DECREASE IN FUNCTION
1490 REM N =  # OF VARIABLES IN FUNCTION
1500 REM NA = # OF VARIABLES UNCHANGED FROM PREVIOUS POINT
1510 REM NF = # TIMES FUNCTION EVALUATED
1520 REM NG = # TIMES GRADIENT EVALUATED
1530 REM NI = # OF GRADIENT EVALUATIONS WHEN B(N,N) LAST SET = I
1540 REM PO = 1: PRINT CURRENT POINT & GRADIENT
1550 REM ST = STEP SIZE FOR SEARCH ALONG LINE
1600 REM                  INITIALIZE
1610 N = 3: REM INSERT # OF VARIABLES HERE<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
1620 D = (N*N-N)/2 + N
1630 DIM A(N), AL(N), B(D), G(N), GL(N), S(N), W(N), Y(N)
1640 REM<<<<<<<<<INSERT INITIAL VALUES OF VARIABLES BELOW<<<<<<<<<<<<<
1650 A(1) = 1: A(2) = 1: A(3) = 0
1660 REM<<<<<<<<<INSERT INITIAL VALUES OF VARIABLES ABOVE<<<<<<<<<<<<<<
1670 AD = .5
1680 MG = 200
1690 MR = .0001
1810 GOSUB 4000: REM TO EVALUATE FUNCTION
1820 NF = NF+1
1830 FL = F
1850 NG = NG+1
1999 REM                   MAIN ROUTINE
2000 REM                  SET B(N,N) = IDENTITY MATRIX
2010 FOR I = 1 TO N
2020 FOR J = I TO N
```

constraints can be transformed into an equivalent problem with no constraints. Suppose we want to minimize

F = (x-3)^2+(y+2)^2

with respect to x and y subject to

x+y = 1.

In this case it is easy to solve for y in terms of x and convert the problem to one of unconstrained minimization in terms of x. Alternatively, the problem can be transformed by forming a penalty function on the linear constraint, adding it to the original function

G = (x-3)^2+(y+2)^2+1000*
(x+y-1)^2,

then minimizing with respect to x and y. The penalty function 1000*(x+y-1)^2 has a leading constant chosen to be large and positive so that the minimum of G occurs only if (x+y-1)^2 is zero. When (x+y-1)^2 is zero (x+y-1) must be zero, and the constraint is satisfied.

More generally, consider

Min: F = f(x(1), ..., x(n))

where minimization is with respect to x(1), ..., x(n) and the n variables are subject to a constraint

g(x(1), ..., x(n)) = 0.

The problem is equivalent to one with no constraint

Min: G = f(x(1), ..., x(n))
+c*g(x(1), ..., x(n))^2

if c is sufficiently large and positive. By applying this transformation repeatedly, we can handle any number of equality constraints.

But what about inequality constraints? Modifying the previous example slightly, suppose we want to minimize

F = (x-3)^2+(y+2)^2

with respect to x and y subject to

x+y >= 1.

Transform the problem to

Min: G = (x-3)^2+(y+2)^2+1000*
(x+y-1-z^2)^2,

and minimize with respect to x, y, and z. As before, we add a penalty function with a large positive constant so that the minimum of G must

```
2022 K = (I-1)*(N-(I-2)/2-1)+J
2030 B(K) = 0
2040 IF I = J THEN B(K) = 1
2050 NEXT J,I
2060 NI = NG
2100 REM                TOP OF MAIN LOOP
2110 GOSUB 10700: REM TO PRINT ROUTINE
2120 REM           SAVE PARAMETERS & GRADIENT WHERE FUNCTION LOWEST SO
2130 REM           FAR
2140 FOR I = 1 TO N
2150 AL(I) = A(I)
2160 GL(I) = G(I)
2170 NEXT
2200 REM           COMPUTE SEARCH DIRECTION
2210 FOR I = 1 TO N
2214 S(I) = 0
2220 FOR J = 1 TO N
2222 IF I <= J THEN K = (I-1)*(N-(I-2)/2-1)+J
2224 IF I >  J THEN K = (J-1)*(N-(J-2)/2-1)+I
2230 S(I) = S(I) - B(K)*G(J)
2240 NEXT J,I
2300 REM           COMPUTE EST CHANGE IN F FOR STEP = 1
2310 FE = 0
2320 FOR I = 1 TO N
2330 FE = FE + S(I)*G(I)
2340 NEXT
2400 REM           CHECK EST CHANGE IN FUNCTION
2410 IF FE >= 0 AND NI = NG THEN 3500: REM CONVERGENCE, TO END
2420 IF FE >= 0 AND NI <> NG THEN 3400: REM TO RESET B(N,N) = I
2430 REM ELSE FE < 0
2500 REM           PERFORM LINEAR SEARCH
2510 ST = 1: REM INITIAL STEP LENGTH
2520 REM      UPDATE A(N)
2530 NA = 0
2540 FOR I = 1 TO N
2550 A(I) = AL(I)+ST*S(I)
2560 IF A(I) = AL(I) THEN NA = NA+1
2570 NEXT
2600 REM      COMPUTED PT <> PT WHERE FUNCTION LOWEST SO FAR?
2610 IF NA = N AND NI = NG THEN 3500: REM CONVERGENCE, TO END
2620 IF NA = N AND NI <> NG THEN 3400: REM TO RESET B(N,N) = I
2630 REM ELSE HAVE NEW POINT
2640 GOSUB 4000: REM TO EVALUATE FUNCTION
2650 NF = NF+1
2700 REM      CHANGE IN FUNCTION MUCH LESS THAN ESTIMATED?
2710 IF F-FL < MR*FE*ST THEN 2720: REM TO CONTINUE ITERATION
2712 REM       ELSE CHANGE MUCH LESS THAN ESTIMATED
2714 ST = AD*ST: REM ADJUST STEP SIZE
2716 GOTO 2520: REM TO RECAL A(N) USING SMALLER STEP SIZE
2720 REM           CONTINUE
2730 FL = F
2750 NG = NG+1
2800 REM           COMPUTE Y(N), S(N), E1
2820 E1 = 0
2830 FOR I = 1 TO N
2840 S(I) = ST*S(I)
2850 Y(I) = G(I)-GL(I)
2860 E1 = E1 + S(I)*Y(I)
2870 NEXT
2880 REM CHECK E1 > 0 TO ASSURE A POSITIVE DEFINITE B(N,N)
2890 IF E1 <= 0 THEN 3400: REM TO RESET B(N,N) = I
2900 REM           COMPUTE (Y(N)^T)*B(N,N)*Y(N)
2920 E2 = 0
2930 FOR I = 1 TO N
2940 FOR J = 1 TO N
2942 IF I <= J THEN K = (I-1)*(N-(I-2)/2-1)+J
2944 IF I >  J THEN K = (J-1)*(N-(J-2)/2-1)+I
2950 E2 = E2 + Y(I)*B(K)*Y(J)
2960 NEXT J,I
2970 E2 = 1 + E2/E1
2980 REM           COMPUTE W(I) = B(N,N)*Y(N)
2990 FOR I = 1 TO N
3000 W(I) = 0
3010 FOR J = 1 TO N
3012 IF I <= J THEN K = (I-1)*(N-(I-2)/2-1)+J
3014 IF I >  J THEN K = (J-1)*(N-(J-2)/2-1)+I
3020 W(I) = W(I) + B(K)*Y(J)
3030 NEXT J,I
3100 REM           UPDATE B(N,N)
3110 FOR I = 1 TO N
3120 FOR J = I TO N
3122 K = (I-1)*(N-(I-2)/2-1)+J
3130 B(K) = B(K) - (S(I)*W(J) + W(I)*S(J) - E2*S(I)*S(J))/E1
3150 NEXT J,I
3200 REM           CHECK # OF ITERATIONS
3210 IF NG < MG THEN 2100: REM TO TOP OF MAIN LOOP
3220 IF NG >= MG THEN 3500: REM TO END ALGORITHM
3300 REM                BOTTOM OF MAIN LOOP
3400 GOTO 2000: REM TO RESET B(N,N) = I
3500 REM                END ALGORITHM
3510 PO = 1
3520 GOSUB 10700: REM TO PRINT ROUTINE
3540 END
4000 REM***************************************************************
4010 REM                EVALUATE FUNCTION & GRADIENT
4020 P1 = A(1)-3: P2 = A(2)+2: P3 = A(1)+A(2)-1-A(3)*A(3)
4030 F = P1*P1+P2*P2+1000*P3*P3: REM VALUE OF FUNCTION<<<<<<<<<<<<<<<<
4040 G(1) = 2*P1+2000*P3: REM PARTIAL W. RESPECT TO A(1)<<<<<<<<<<<<<<<
4050 G(2) = 2*P2+2000*P3: REM PARTIAL W. RESPECT TO A(2)<<<<<<<<<<<<<<<
4060 G(3) = 2000*P3*(-2*A(3)): REM PARTIAL W. RESPECT TO A(3)<<<<<<<<<<
4100 RETURN
10700 REM**************************************************************
10710 REM                PRINT
10720 M1$ = "ITERATION"
10740 M3$ = "FUNCTION = "
10750 M4$ = "VARIABLES"
10760 M5$ = "PARTIAL DERIVATIVES"
10770 IF NF = 1 THEN PRINT: PRINT: PRINT#5,: PRINT#5,
```

Continued

```
10780 IF NF=1 OR NG=10*INT(NG/10) OR PO=1 THEN PRINT M3$; FL, M1$; NG
10782 IF NF=1 OR NG=10*INT(NG/10) OR PO=1 THEN PRINT#5,M3$; FL, M1$; NG
10786 REM      PRINT POINT & GRADIENT ON 1ST & LAST ITERATIONS
10790 IF NF > 1 AND PO = 0 THEN 10860: REM TO RETURN
10792 PRINT#5, M1$; NG TAB(40) M2$; NF
10800 PRINT SPC(2) M4$ TAB(30) M5$
10810 PRINT#5,SPC(2) M4$ TAB(30) M5$
10820 FOR I = 1 TO N
10830 PRINT SPC(3) "A("; I ") = "; A(I) TAB(31) "G("; I ") = "; G(I)
10840 PRINT#5,SPC(3) "A("; I ") = "; A(I) TAB(31) "G("; I ") = "; G(I)
10850 NEXT
10860 RETURN
```

occur when the penalty function is zero. For the penalty function to be zero requires $x+y-1 = z^2$. Because z^2 is non-negative, $x+y-1$ must be non-negative as well, assuring the constraint is satisfied.

In general, we have

$$\text{Min: } F = f(x(1), \ldots, x(n))$$

subject to

$$g(x(1), \ldots, x(n)) >= 0.$$

Convert the problem to

$$\text{Min: } F = f(x(1), \ldots, x(n)) + c*(g(x(1), \ldots, x(n)) - x(n+1)^2)^2$$

where c is large and positive, and minimize with respect to $x(1), \ldots, x(n+1)$.

We can handle as many restrictions as we like by adding new penalty functions and variables. If there is a constraint with the sense of the inequalities reversed (<= rather than =>) it can be transformed by multiplying both sides by -1. Strict inequalities (> rather than >=) can be altered by adding a small, arbitrary constant to the right-hand side, making a mixed (>=) constraint.

As set up in the listing, the program runs the example with two variables and an inequality constraint. In the program notation, the problem is

Min: F = (A(1)-3)^2+(A(2)+2)^2 +1000*(A(1)+A(2)-1-A(3)^2)^2.

The minimum is 0 at A(1)=3, A(2)=-2, A(3)=0.

In running the program, take some care in choosing starting values. Run times and, if there are multiple local minima, properties of the solutions will depend on the initial values.

★

THE OSI 700 SERIES COMPUTERS

This article is a compilation of excerpts from an Isotron Press Kit. Although specifically aimed at the new 700 series of machines, we have included information on more general corporate happenings at the request of a number of subscribers.

In November 1983, Isotron, Inc., a subsidiary of Investment AB Beijer, acquired the assets of Kendata, Inc. Investment AB Beijer, the largest investment corporation in Scandanavia, has interests in a number of other advanced technology firms. Beijer's belief in the strong prospect for the multi-user market and its respect for the OSI hardware were factors in the decision to invest.

Additionally, Beijer's long range plans called for developing advanced Unix based computers that could be manufactured and marketed by Isotron.

The growing industry interest in multi-user computers and the market potential for the new Ohio Scientific Unix series led a second Swedish company, Ahlsell AB, to invest in Isotron in early 1985. As a result of this investment, a Xenix compatible computer developed by Ahlsell's subsidiary, Data Industrier AB, has been incorporated into the Ohio Scientific line.

Why a series of UNIX/XENIX compatible computers? President Bob Lewis says "It has always been an Ohio Scientific tradition to be at the forefront of microcomputer technology. UNIX offers tools that will enable software developers to create sophisticated programs with more usability and marketability than those developed with any other currently available operating system. We've had a license for Xenix since 1980 but have been waiting for technology that provides the proper combination of hardware and software."

The small business market, one of the most rapidly growing segments of the computer industry, requires multi-user systems with the data storage capacity of hard disks. Ohio Scientific products fit into a niche between personal and mainframe computers.

Bob Lewis says, "We have no plans to discontinue either the OSI 200 or 300 line."

Users demand systems that can be expanded without loss of performance, are easily networked, contain moderate to large disk capacity, and offer easy-to-use application software. At the same time the systems should be price/performance leaders.

The Ohio Scientific multi-user, multi-processing systems exceed the performance of all competitive offerings. The OSI systems give each user in a multi-user environment his own computer and can be easily expanded to include additional users. The networking scheme that is used to tie these systems together is unmatched in the market in terms of simplicity and low cost. The new Unix/Xenix compatible 700 series provide state of the art hardware designed to meet the needs of both the end user and the dealer. Attention has been focused on creating a versatile computer that will enable software developers to take advantage of the transportability and power of the Unix environment.

The 700 Series are high performance 32 bit supermicros designed for speed and flexibility. The OSI 710 and OSI 720, first two models in the series, support RTIX, Ohio Scientific's real time Unix/Xenix compatible operating system as standard, which made its debut at COMDEX Spring in Atlanta, Georgia. RTIX is the standard operating system for the 720 with Unisoft's Uniplus+ System V available as an option.

RTIX was chosen over XENIX or UNIX because RTIX is a universal operating system, offering more flexibility than a single licensed version of UNIX or XENIX. RTIX makes the OSI 710 and OSI 720 fully compatible with all UNIX/XENIX application software written for the 68000 micro-processors. RTIX requires less memory, thus providing more user program space. RTIX is real time, and is significantly faster than standard versions. However, to accommodate users who may wish to own an additional license, the Xenix license and a full set of Xenix utilities are available as an option for the OSI 710. Unisoft's Uniplus+ System V is available for the OSI 720.

"The computers in our new 700 Series are packed with features not found in other computers in this price range. They are designed to meet the needs of both end users and OEM's," says Robert V. Lewis, president of Isotron, Inc. Both are compatible with each other as well as with application software written for Unix version 6 and 7, Systems III and V, Xenix, and the pending IEEE Unix standards.

Based on the 68010 microprocessor, the OSI 710 has 2 MB RAM expandable to 8 MB, with 20 MB of storage as standard. In a basic configuration, this computer will support six users but is expandable to sixteen.

Communications is a strong feature of the OSI 710 which offers easy interface and data exchange with other mini and mainframe computers via protocols such as IBM 3270, BSC/SNA, IBM 2780/3780, IBM 3770, Univac UTS 400 and others. It also communicates with other OSI computers via low cost, high speed networking.

The OSI network is inexpensive and simple to use. It features remote file sharing that allows record and file locking functions to be used in the network.

The OSI 720 accommodates 16 users standard with expansion up to 32. It utilizes three microprocessors: one 68010 or 68020 CPU, one 68000 and one 68008 co-processor. The OSI 720 offers 2 - 16 MB no wait state RAM, a 50-150 MB Winchester drive, 5 1/4" floppy drive, and supports streamer magnetic tape memory back-up.

Continued on page 12

ESSENTIAL OS-U PROGRAMMER SUBROUTINES

By: Roger Clegg
Data Products Maintenance Corp.
9460 Telstar
El Monte, CA 91731

Every OS-U programmer should have these routines in his arsenal. The fact that Roger is sharing them is guaranteed to make your life a lot easier, not to mention save you hours working them out for yourself. We hope that Roger's lead will prompt you to share little ditties too.

```
1 REM *********************   M I S C   ******************************
100 REM   A GENERALIZED MONEY FORMATTING ROUTINE
101 :
102 REM   "GOSUB 110" returns X$ with two decimal places and no leading
103 REM   blank,  unless padded to length W.
104 REM   "GOSUB 120" prints X, left justified with either leading
105 REM   blank or minus, then $ sign, then number with 2 places.
106 REM   "GOSUB 130" and "GOSUB 140" print X right justified,  with
107 REM   rightmost digit at TA-1, with or without | following.
108 :
110 W%=1
120 L%=1
130 C%=1
140 X$=STR$(INT(X*100+.5)/100): IF W% THEN L%=0: C%=0
150 IF ASC(RIGHT$(X$,2))=46 THEN X$=X$+"0"
160 IF ASC(RIGHT$(X$,3))<>46 THEN X$=X$+".00"
170 IF L% THEN PRINT#D,LEFT$(X$,1)"$"MID$(X$,2);: L%=0: C%=0: RETURN
180 IF ASC(X$)=32 THEN X$=MID$(X$,2)
190 IF W% AND LEN(X$)<W THEN X$=RIGHT$("              "+X$,W)
200 IF W% THEN W%=0: RETURN
210 PRINT#D,TAB(TA-LEN(X$))X$;: IF C% THEN PRINT#D,"|";: C%=0
220 RETURN
230 :
240 D  is the output device number
250 W  is the minimum width, if wanted, for GOSUB 110.
260 TA is the tab for the right-justification in GOSUB 130 or 140.
270 :
280 :
300 REM   ACCEPTS X, RETURNS X$ WITH TWO DECIMAL PLACES
310 :
320 X$=STR$(INT(X*100+.5)/100): IF ASC(X$)=32 THEN X$=MID$(X$,2)
330 IF ASC(RIGHT$(X$,2))=46 THEN X$=X$+"0": RETURN
340 IF ASC(RIGHT$(X$,3))=46 THEN RETURN
350 X$=X$+".00": RETURN
360 :
370 :
400 REM   PRINTS X WITH TWO DECIMAL PLACES, RIGHT-JUSTIFIED AT TA-1
410 :
420 X$=STR$(INT(X*100+.5)/100): IF ASC(X$)=32 THEN X$=MID$(X$,2)
430 IF ASC(RIGHT$(X$,2))=46 THEN X$=X$+"0"
440 IF ASC(RIGHT$(X$,3))<>46 THEN X$=X$+".00"
450 IF TA THEN PRINT#D,TAB(TA-LEN(X$)) X$;
460 RETURN
470 :
480 If the column width is adequate then the second half of line 420,
485 which strips the leading blank,  can be omitted.
490 :
495 :
500 REM   ACCEPTS X$ IN BASE 2, 8, 10 or 16,  RETURNS DECIMAL X
501 :
502 REM   Prefix $ or H denotes Hexadecimal
503 REM   Prefix & or O denotes Octal
504 REM   Prefix % or B denotes Binary
505 REM   No prefix denotes Decimal
506 :
510 X=ASC(X$): IF X=36 OR X=72 THEN BASE=16: GOTO 550
520 IF X=38 OR X=79 THEN BASE=8: GOTO 550
530 IF X=37 OR X=66 THEN BASE=2: GOTO 550
540 X=VAL(X$): RETURN
550 X=0: FOR II=2 TO LEN(X$)
560 Y=ASC(MID$(X$,II))-48: X=X*BASE+Y+7*(Y>9): NEXT: RETURN
570 :
580 :
600 REM   ACCEPTS DECIMAL X AND BASE,  RETURNS X$
610 :
620 X$=""
630 Y=INT(X/BASE): Z=X-Y*BASE: X$=MID$("0123456789ABCDEF",Z+1,1)+X$
640 IF Y THEN X=Y: GOTO 630
650 RETURN
660 :
670 The above two routines handle the natural numbers only.
680 :
690 :
700 REM   ACCEPTS DATE WITH SLASHES, RETURNS DATE IN WORDS
710 :
720 INPUT"Date (M/D/Y) ";DATE$
730 X=VAL(DATE$): IF X<1 OR X>12 THEN PRINT CHR$(7);: GOTO 720
740 DATA January,February,March,April,May,June,July,August,September
750 DATA October,November,December: RESTORE: FOR II=1 TO X:READ X$: NEXT
760 X=3: IF MID$(DATE$,3,1)="/" THEN X=4
770 DATE$=X$+STR$(VAL(MID$(DATE$,X)))+", 19"+RIGHT$(DATE$,2): RETURN
780 :
790 :
800 REM CONVERTS DATE TO NUMBER SINCE 1/1/1900, AND GIVES DAY OF WEEK
810 :
820 DIM M(12),D$(6): C365.25: M$=" 312831303130313130313031"
825 X=0: FOR I=1 TO 12: M(I)=X+.01*I-.025: X=X+VAL(MID$(M$,I*2,2)):NEXT
830 D$(0)="Sunday": D$(1)="Monday": D$(2)="Tuesday": D$(3)="Wednesday"
```

Continued

The system contains the standard VMS bus for board expansion. The units can also be networked with the optional networking board.

Both computers are bundled with powerful software packages, Uniplex II and MIMER. Uniplex II is an integrated package that includes a relational database (with record and file locking), spreadsheet, word processing, menu builder, screen builder, print spooler, and IBM-compatible electronic mail.

MIMER is a relational database manager capable of handling a very large number of records and equipped with an optional program generator for easy application development. Code produced by the program generator is easily portable to large mini-computers and MS-DOS based personal computers.

For traditional program development both computers offer Ohio Scientific BASIC, featuring integration with the MIMER data base manager, ISAM files and record locking. PASCAL, FORTRAN 77, COBOL, APL, C, and Assembler are also available.

The OSI 710 which is scheduled to be shipped by the end of the second quarter, is priced at $10,500, while the OSI 720 shipments of which are expected to begin in the fourth quarter will be $14,000. The next addition to this series will be an entry level Unix/Xenix compatible computer to provide dealers with a full range of price and capability.

As for software transportability, programs written for OS-65U and TurboDOS can be transported by using OSI BASIC, which is bundled with the 710 and 720. This transfer software recompiles commands in the OS-65U programs that are invalid under RTIX.

200 & 300 SERIES HIGHLIGHTS

The Series 200 is a line of computers that utilize a single 2 MHz 6502 microprocessor in a time-sharing mode, useful for up to 4 - 6 terminals. The line is very strong because of the great variety of software packages that have been developed for it. The operating system is the proprietary OS-65U. Typical computers in the Series 200 cost between $5,000 and $10,000.

The most recent update to this line is the Series 235 family which utilize 5 1/4" hard disk technology. The 235/2 series

PROGRAMMER SUBROUTINES CONTINUED:

```
840 REM  The above two lines should be in initialization
845 :
850 X=3: IF MID$(X$,3,1)="/" THEN X=4
860 X=INT(VAL(MID$(X$,X))+M(VAL(X$))+C*VAL(RIGHT$(X$,2)))
870 DAY$=D$(X-7*INT(X/7))
880 RETURN
890 :
895 :
900 REM   ACCEPTS NAME, RETURNS SURNAME (FLG on) OR 2 INITIALS & SURNAME
910 :
920 FOR II=LEN(NAME$)-2 TO 1 STEP -1: IF MID$(NAME$,II,1)<>" " THEN NEXT
930 X$=MID$(NAME$,II+1,2): IF X$="JR" OR X$="SR" OR X$="II" THEN NEXT II
940 K=II: X$=MID$(NAME$,K+1): IF FLG THEN RETURN
950 FOR II=2 TO K
960 IF MID$(NAME$,II,1)<>" " OR MID$(NAME$,II+1,1)=" " THEN NEXT
970 Y$=MID$(NAME$,II+1,1)+". ": IF II>=K THEN Y$="   "
980 X$=LEFT$(NAME$,1)+"."+Y$+X$: RETURN
990 :
995 :
1000 REM   PRIME NUMBER GENERATOR
1001 :
1002 REM   Finds all primes less than 16K, using sieve of Eratosthenes.
1010 :
1020 I=0: K=0: P=0: N=16384: S=SQR(N): DIM F%(N/2): PRINT 2
1030 FOR I=1 TO N/2: IF F%(I) THEN NEXT: END
1040 P=I+I+1: PRINT P: IF P>S THEN NEXT: END
1050 FOR K=(P*P-1)/2 TO N/2 STEP P: F%(K)=1: NEXT: NEXT: END
1060 :
1070 :
2000 REM   ROUTINES FOR HANDLING MONEY ACCURATELY UP TO $32 BILLION
2001 :
2002 REM   These routines split a financial amount X$ into two
2003 REM   components: X% for the millions, and X for the remainder.
2004 REM   This avoids inventing new variable names.  The routines
2005 REM   assume that  M=1000000  has already been defined.
2006 :
2010 REM   ACCEPTS X$, RETURNS X% AND X
2020 :
2030 IF ASC(X$)=32 THEN X$=MID$(X$,2): GOTO 2030
2040 Z=1: IF LEFT$(X$,1)="-" THEN Z=-1: X$=MID$(X$,2)
2050 FOR II=1 TO LEN(X$): IF MID$(X$,II,1)<>"." THEN NEXT
2060 IF II<8 THEN X=Z*VAL(X$): X%=0: RETURN
2070 X=Z*VAL(MID$(X$,II-6)): X%=Z*VAL(LEFT$(X$,II-7)): RETURN
2080 :
2100 REM   ACCEPTS X% AND X,  RETURNS X$, AND PRINTS X$ IF TA<>0
2110 :
2120 Z=INT(ABS(X/M))*SGN(X): IF Z THEN X=X-M*Z: X%=X%+Z
2130 IF X% THEN Z=SGN(X%): IF SGN(X)=-Z THEN X=X+M*Z: X%=X%-Z
2140 X$=STR$(INT(X*100+.5)/100): IF ASC(RIGHT$(X$,2))=46 THEN X$=X$+"0"
2150 IF ASC(RIGHT$(X$,3))<>46 THEN X$=X$+".00"
2160 IF X% THEN X$=STR$(X%)+RIGHT$("000000"+MID$(X$,2),9)
2170 IF ASC(X$)=32 THEN X$=MID$(X$,2)
2180 IF TA THEN PRINT#D,TAB(TA-LEN(X$)) X$;
2190 RETURN
2195 :
2200 REM   PERFORMS  T$=A$+B$
2210 :
2220 X$=A$: GOSUB 2000: A=X: A%=X%
2230 X$=B$: GOSUB 2000: X=X+A: X%=X%+A%: GOSUB 2100: T$=X$
2240 :
2300 REM   PERFORMS  T$=0: FOR I=1 TO N: T$=T$+A$(I): NEXT
2310 :
2320 T=0: T%=0: FOR I=1 TO N: X$=A$(I): GOSUB 2000
2330 T=T+X: Z=INT(ABS(T/M))*SGN(T): T=T-Z*M: T%=T%+X%+Z
2340 NEXT I: X=T: X%=T%: GOSUB 2100: T$=X$
2350 :
2360 :
2370 :
3000 REM   ACCEPTS X (DOLLAR AMOUNT), RETURNS X$ IN WORDS
3010 :
3020 DIM W$(27): GOSUB 3300: REM This should be done at initialization
3030 X=INT(X*100+.5)/100: IF X<=0 OR X>1E7 THEN X$="** VOID **": RETURN
3040 X$="": Y=INT(X/1000000)
3050 IF Y THEN X=X-Y*1000000: GOSUB 3200: X$=X$+"MILLION "
3060 Y=INT(X/1000): IF Y THEN X=X-Y*1000: GOSUB 3200: X$=X$+"THOUSAND "
3070 Y=INT(X): IF Y THEN X=X-Y: GOSUB 3200
3080 IF X$="" THEN X$="ZERO "
3090 IF X$="ONE " THEN X$="ONE DOLLAR ": GOTO 3110
3100 X$=X$+"DOLLARS "
3110 IF X<.005 THEN X$=X$+"AND NO CENTS": RETURN
3120 Y=INT(X*100+.5): GOSUB 3200
3130 X$=X$+"CENT": IF X>.015 THEN X$=X$+"S"
3140 RETURN
3150 :
3200 Z=INT(Y/100): IF Z THEN X$=X$+W$(Z)+" HUNDRED ": Y=Y-Z*100
3210 IF Y=0 THEN RETURN
3220 IF X$<>"" THEN X$=X$+"AND "
3230 IF Y<21 THEN X$=X$+W$(Y)+" ": RETURN
3240 Z=INT(Y/10): X$=X$+W$(18+Z): Y=Y-Z*10: IF Y THEN X$=X$+"-"+W$(Y)
3250 X$=X$+" ": RETURN
3260 :
3300 DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN,ELEVEN
3310 DATA TWELVE,THIRTEEN,FOURTEEN,FIFTEEN,SIXTEEN,SEVENTEEN,EIGHTEEN
3320 DATA NINETEEN,TWENTY,THIRTY,FORTY,FIFTY,SIXTY,SEVENTY,EIGHTY,NINETY
3330 FOR I=1 TO 1000: READ W$(1): IF W$(1)<>"ONE" THEN NEXT: STOP
3340 FOR I=2 TO 27: READ W$(I): NEXT: RETURN
```

★ ★

computers are configured for two users and can be easily expanded for up to four users. Disk capacity is 18.4 megabytes formatted. The 235/4 series is designed for four users with formatted disk capacity of 29 megabytes.

This series contains the 515-System I/O Board that provides functions requiring from five to seven boards in other 200 series computers. In addition to simplifying maintenance, the 515 Boards are more cost effective. One board can be purchased at savings of as much as $2,465 retail over the cost of the normal complement of five to seven boards.

The Series 300, introduced in 1983, utilize multiple Z/80A microprocessors. Each terminal connected to the computer has its own processor, with common resources such as disks and printers shared by all users. With the multi-processor architecture, the Series 300 gives a superior performance in systems with 4 or more simultaneously connected terminals. The operating system is the CP/M compatible TurboDOS.

The most recent addition to the 300 line is the 345 series. This series utilizes the 5 1/4" hard disk technology and is expandable to eight users. THe 345G has a formatted disk capacity of 21.6 megabytes, and the 345I offers 43.2 megabytes formatted.

Isotron markets a number of terminals and printers manufactured by outside suppliers but sold under the Ohio Scientific brand name. Several horizontal software packages such as word processing and data bases are available. In addition, Isotron is now offering vertical packages supplied by outside vendors.

COMPANY MANAGEMENT

THOMAS JABLONSKI, chief executive officer, has extensive previous experience as president of a corporation that markets Ohio Scientific computers in Scandinavia. His background includes M.Sc. in electronics and 13 years experience in the management of companies developing and marketing computer hardware and software. As an individual with both a technical background and long experience in finance and marketing, Mr. Jablonski has overall responsibility for Isotron's operations and long-term strategic planning.

ROBERT V. LEWIS, president, is primarily responsible for the planning and execution of marketing and sales activities as well as corporate issues of a legal and administrative nature. He has had several years experience in top management in research and development, marketing and asset management as well as three years experience in banking and finance. Mr. Lewis holds a degree in economics and a Masters of Business Administration.

ERIC DAVIS, vice president/operations, Aurora, has 13 years experience in the electronics industry encompassing all aspects of manufacturing from printed circuit design to management. Since joining Ohio Scientific in 1976, he has held positions as project technician, manager of technicians, production manager, and director of manufacturing.

★

IMPROVING OSI MINI-FLOPPY DISK RELIABILITY BY PROVIDING HEAD-UNLOAD

By: Bob Ankeney
5740 S.E. 18th Ave.
Portland, OR 97202

Last month Dave Pompea described the 'whole hog' approach that required serious mods and construction. This month, Bob Ankeney gives the more casual hacker an easy way to get at least half of the hog very quickly. Whichever approach you prefer, we do suggest that you implement one or the other.

PURPOSE: Describes a modification to OSI mini-floppy disk units that causes the read-write heads to unload (raise the pressure pad) when the drive is not being accessed. Many users report disks become unreadable with physical track destruction after periods as short as one-half hour. This modification cures this problem.

IMPLEMENTATION: You will need to open the disk drive unit by removing the screws on the bottom of the unit. C4P units will require an Allen wrench to remove these screws. On a flat surface, remove the cover (by pulling it up) and gently lay it to the side. Note that it stays connected to the drive by fairly short wires. You may have to have another person hold the sides of the disk drive apart slightly in order to remove the cover easily without scratching the sides, especially on wood units.

On the right rear corner of the disk drive you will find a socket with a programmable jumper plug in it (SEE FIGURE). OSI programs these jumpers by simply bending out any pins which they want disconnected. These jumpers connect the drive select line (pins 10-13) to either pin 2 (for an A drive) or pin 3 (for the B drive). In addition, a jumper connects pin 14 (head load) pin 1 (HS). OSI keeps pin 1 continually grounded, thus keeping the head down at all times when the drive is selected. However, the computer's head load signal is

Continued

ERRATA

"OS-65U SELECTIVE SEARCH & PRINT PROGRAM" article by Raymond D. Roberts, May 1985.

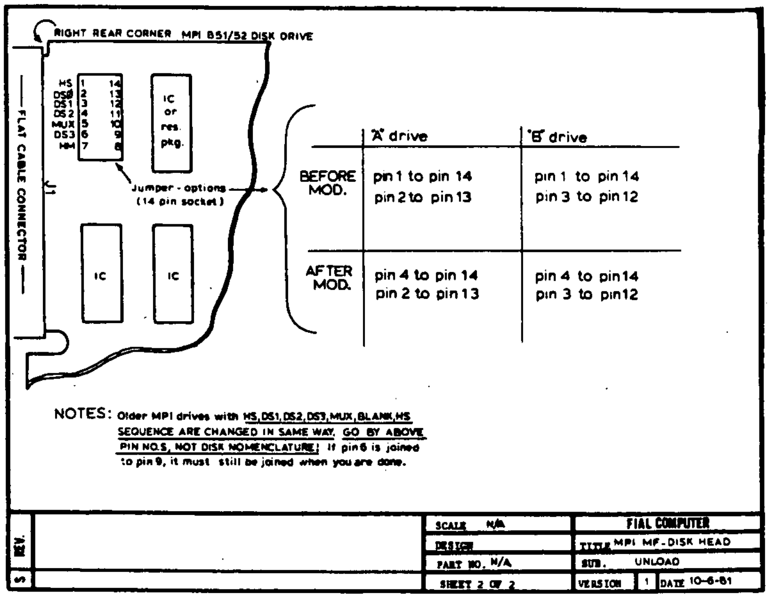
The FROM: TX () INDEX TO: IFINDEX (1) change in the article on page 14 of the May issue (ADS100) somehow got reversed and should really be changed:

```
Line 33035
FROM: IFINDEX  (1)>TX+17  THEN
INDEX <1> = INDEX (1):GOTO3005

TO: IFTX<>INDEX (1) THEN INDEX
<1> = INDEX (1)+1:GOTO 3005:
```

if changing from a 3 digit first field to a longer one.

Raymond D. Roberts
Ferndale, WA 98248



brought to the drive and appears at pin 4 (labeled DS2 on newer drives, DS3 on older drives). Therefore, by disconnecting the head load line (pin 14) from pin 1 and connecting it instead to pin 4, head-load will happen only at the instant the disk is accessed.

There are two ways to accomplish the change:

Method 1: Bend out pin 1 on the jumper (disconnects the always-load-head), bend down pin 4 (connect it) and bend out pin 11 (disconnect it). Then solder a small jumper wire (on the top of the plug) from pin 4 to pin 14 (drive head load). Replace the plug in the correct position.

Method 2: Obtain a 14-pin DIP plug. Remove the OSI 12-pin jumper plug. On the DIP plug, connect a short wire from pin 4 to pin 14. Next, connect a short wire from pin 2 to pin 13 if this is an A drive, OR from pin 3 to pin 12 if this is a B drive. Do not do both or your drive will be an A and a B drive at the same time! Insert the DIP plug into the jumper-options socket. Be sure the two wires are not shorting together and that you have the correct orientation. If you cannot obtain a DIP plug, you may make the connections by plugging the wires directly into the socket (use about 28 ga. stripped about 3/16 from the end. Re-do this with a DIP plug as soon as you get one, as plugged-in wires will work loose eventually.

WARNINGS:

1. The programmable jumper plug used by OSI has only 12 pins, but plugs into a 14 pin socket. Be sure you plug it in so that it is even with the pin 1 end of the socket. Also, all pin numbers given above refer to the SOCKET pins, which are numbered 1 through 14 counter-clockwise when viewed from the top of the drive. Pin 1 is at the right-rear corner of drive.

2. When this modification is installed, the drive will make more noise when the heads load and unload. Also, both heads load and unload, although only one drive is selected. This is a small price to pay for the reduction in disk wear.

3. On older drives, pin 6 is joined to pin 9. If this is the case, be sure they are still connected when you are done. This connection did not seem to do anything on the drives we examined, but may be necessary on some older drives.

Fial Computers has performed the above procedure on many machines with complete success.

Our thanks to Fred Ornellas of Alpine Electronics for discovering this easy fix.

★

SILENCE YOUR DISK DRIVE

By: Dave Pompea
319 Hampton Blvd.
Rochester, NY 14612

CHANGES TO C1P/MF DRIVE FIX FOR USE ON A C4P/MF

The C4P/MF interface (505 board) is the same as the C1P/MF with the exception of a couple of extra pins on the interface board that connects the ribbon cable to the drive. Use the original schematic as published in the July issue of Peek(65) with the following nomenclature changes:

Change U72 (682Ø PIA) to U1A,

Change U75 (7417 driver) to U4A,

Change R33 & R34 (index hole bias resistors) to R17 & R22,

Change R37 (driver pull up resistor) to R7,

Change U5 (spare 74Ø4) to U1F,

Change the reference from U7 pin 6 (not COxx line) to U1F pin 13 or U5J pin 7.

For the spare 3 input nand gate (U7), use the spare 2 input nand gate U6H and change the pin numbers: pin 13 to 1, pin 1 to 2, and pin 12 to 3.

Some Notes: Be sure to check to make sure that pin 4 on the interface board is not grounded. On the schematic, the pin number for the not Q output of the second one shot was omitted, it should be #12 as noted in the write up. To increase the time that the motor stays on, just increase the value of the resistor (44k) or capacitor (1ØØ uf) on the first one shot. Do the same to increase the second one shot to increase the motor spin up time if your drive needs it.

★

DABUG

By: John Whitehead
17 Frudal Crescent
Knoxfield 318Ø
Australia

DABUG is an Australian replacement 2K Monitor written by David Anear.

The first version for the Superboard Series I was available in 198Ø. It contained all the original MONITOR ROM functions plus on screen editing with cursor and destructive backspace, screen clear from keyboard, or in a BASIC program, screen freeze and single key BASIC all for use with ROM BASIC, programs in EPROM or from cassette.

The good point about DABUG is that it is fully compatible

with all cassette and disk programs because there is the option of leaving the extra functions turned off if they are not compatible with a program.

The disk boot and 65V MONITOR entry points are the same as the original OSI SYN600 MONITOR ROM.

David also wrote it for C4P, 65D, 65U, Apple 2 and 2E, and said there are over 5000 in use.

When the Superboard Series II came out, he added the 48 x 12 screen driver and called it DABUG 3.

When I saw a correctly decoded keyboard routine for a C4 by Don Vansyckel in an Aardvark Journal, I typed it into the Assembler and tried it out. With the Shift Lock up, all worked correctly, but with it down, lower case was not available which is little improvement for BASIC.

I altered the new keyboard routine above to enable lower case to be available with Shift Lock down or up.

To retain all control codes and use the Ctrl key in place of Repeat key, I needed more room. After some hints from David Anear, I was able to obtain more room by reducing and improving the 48 x 12 screen driver. The result of all this is DABUG 3J which has been running since 1983.

In addition to the screen editing, etc. above, DABUG 3J has: - correctly decoded keyboard. With the Shift Lock down, it gives upper case as normal. When left or right Shift is down, it gives lower case for use with BASIC or the Assembler. This makes adding lower case text easy.

With the Shift lock up, keys A to Z are inverted in that it gives lower case when the left or right Shift is up and upper case when it is down like a normal typewriter.

The numerical keys are not effected by the Shift Lock and give correct characters as marked.

To get the characters that were originally obtained with Shift K, L, M, N, and O, non alfa keys are used with the Ctrl key.

All non displaying control codes from $01 to $1F are obtainable with Ctrl A to Z and are not affected by Shift or Shift Lock.

The Repeat Key remains undecoded which means that it can be detected with PEEKs and POKEs without it altering the value of any other key pressed.

The keyboard routine is entered at $FD00 and puts the ASCII value of the key pressed into the accumulator. It does not exit until a key has been pressed. The X and Y registers have the same value on Exit as on Entry.

A subroutine at $FD21 will put the ASCII value of a keypress in the accumulator and if no key is pressed, will put zero in the accumulator and then return.

The BASIC warm start OM ERROR has been fixed.

E added to the Break selection to enter EXMON in EPROM at $E800 in the 48 x 12 mode and Shift Lock does not need to be down after a Break. (EXMON in EPROM is not needed for any of the other functions to work.)

As with DABUG 3, I have tried to keep all entry points in DABUG 3J the same as the OSI Monitor. As the keyboard routine is new, any M/CODE program that entered the original keyboard routine at points other than $FCBE, $FCCF, $FD00 or $FEED will need altering.

WP6502 has its own keyboard routine so the keyboard works as before. Two things I did find with WP was that it would not wait for a keypress when saving to tape and the @ was missing from the end of data stored on tape. I could not work out why for sure, it looked like the ACIA was being turned off before the last character had been fully sent. It was soon fixed with a small delay.

DABUG 3J is available from myself in a 2716 EPROM for the Superboard Series II, or for the Series I with the 48 x 12 screen driver bypassed (state which).

The cost is $A10 plus $A3 P & P.

If you order DABUG 3J and already use WP6502 V1.2, send a C15 tape with a copy of

WP65Ø2 V1.2 on it as proof of ownership and for an extra $A2, I will return a copy of WP that will work with DABUG 3J in 24 x 24 or 48 x 12.

★

MORE FORTH

PART II

By: Charles Curley
Courtesy of O.S.I. Users
Independent Newsletter
5595 E. 7th Street, #285
Long Beach, CA 9Ø8Ø4

The response to last issue's article on FORTH was very good, so I shall add to it with some very useful code which is also useful to illustrate the inner workings of both FORTH and Ohio Scientific computers.

On screens 129 and 13Ø, you will find code to deliver a message to 65D's command buffer, and several examples of its use. This code has allowed me to internalize to FORTH a number of useful DOS commands. The only word in any of these listings not found in the fig-FORTH kernel is DOS, which is in the Blue Sky Products version, and is essentially a call to the DOS to execute its command buffer. The word !" delivers a message to that buffer.

The constant DSBUF points to the address in Ø page where 65D looks for the location of its command strings. Since DSBUF is the only pointer to the pointer, the user may easily change the location of the buffer by changing the contents of the cell pointed to by DSBUF.

(!") and !" are almost exactly the same code as (.") and .". The change is the use of CMOVE to deliver the string to the command buffer instead of to the terminal. (!") is the code that is executed in a compiled word. It works in such a way that the code doesn't care where the command buffer is when it is executed, so that the user can, if he wants, move the buffer around after this code is compiled. !" is both the compiler directive for (!") and the interpretive word. If the system is compiling, then the word compiles (!") and the associated string. Otherwise, it delivers the string to the command buffer directly.

Here's how !" works: 22 is the ASCII character " . The value is placed on the stack for later use by WORD. Then, STATE @ brings onto the stack a flag to determine whether we are compiling or not. If we are compiling, then we execute the compile time code. COMPILE (!") forces the compilation of (!"). Then WORD picks up the 22, and searches the input stream until it finds that value. Thus, " is the ending delimiter for the message string, as you can see from the examples. WORD returns with the string at the top of the dictionary, with the count in the first byte of the string. Since the word HERE points to the top of the dictionary, the phrase HERE C@ returns the count. The 1+ is to allow for the count byte, and the word ALLOT extends the dictionary to include the string. Compilation then proceeds as usual.

```
                                                    SCR #    129
     ( .! -- DOS command                        15 JAN 81 CRC )
     BASE @  HEX      E1  CONSTANT DSBUF

     : (!")
         R   COUNT  DUP  1+  R>  + >R  DSBUF  @  SWAP   CMOVE ;

     :  !"  ( Delivers a message to the 65D command buffer )
         22    STATE   @
         IF  COMPILE  (!")  WORD HERE   C@   1+  ALLOT
         ELSE  WORD  HERE  COUNT  DSBUF  @  SWAP  CMOVE ENDIF ;
     IMMEDIATE

     : HOME !" HO " DOS ;      : ECHO  !" IO ,Ø9 " DOS ;
                             ( Enable devs 3 & 1 as outputs )
     : NOECHO !" IO ,Ø1 " DOS ;             -->
```

```
                                                    SCR #    13Ø
  ( DOS commands  SEA SEB                       15 JAN 81 CRC )
    :  FORMAT  S->D   <#  #S  #   #>   DUP  3  =
       IF  DROP 1+  2  THEN  ;   ( format for 65D buffer )

    : SEA  !" SE A " DOS ;           : SEB  !" SE B "  DOS ;
      BASE  !   ;S
```

```
                                                    SCR #     71
  (  BULK INITIALISER                           Ø6 Jun 81 CRC )
    BASE   @    HEX
    ( initialise  disk in drive  1 ( from trk-2 to trk-1  INT  )
    :  INT   SEB   !" INIT          "   1+ 4D  MIN  SWAP  1  MAX
       DECIMAL               ( initialise  disk in drive  1 )
       DO   I  DUP  4  .R  FORMAT
             DSBUF   @   5 +   SWAP
             CMOVE  DOS    LOOP  ;
      ( initialise current disk entirely )
    :  INIT  !" INIT " [ ØD  HERE  1 - C! ]  DOS  SPACE   ;
                      ( install CR this way for readability )
    BASE  !   ;S
```

```
                                                    SCR #    119
  ( Reload code  --  CRC                        23 Aug 81 CRC )
 CREATE  REBOOT   ( reboot the system from disk! )  BASE     @
      HEX  FFEA    HERE 2 -  !   ( Must have kernel in drive.)
           SMUDGE
 : ?BOOT FEØ1 C@ IF ." Video" ELSE ." Serial" THEN SPACE ;
                 CR  ." Reboot code loaded.  "
      BASE !   ;S
```

If we are not in compilation mode, then the word !" is being executed from the terminal. In that case, we simply get the byte count as before and move the message to the command buffer.

Notice the word IMMEDIATE to force execution of !" during compilation. This is what makes a FORTH word a compiler directive, and one of the facilities in FORTH that makes FORTH so easy to expand.

The rest of screen 129 contains simple examples of the use of !" and DOS. For my own use I made the call to the DOS separate from the message function. One could force immediate operation on the command strings by including the word DOS in the definitions of (!") and !" immediately after each CMOVE. However, some of the code shown here shows why this is not a good ideà.

The word FORMAT on screen 130, Line 1, uses FORTH's formatting code to produce a string suitable for the command buffer from a number on the stack. S->D forces a single precision (16 bit) number on the stack to a double (32 bit) precision number, with sign extension, as required by the formatting words. The phrase <# #S # #> does the actual conversion of the string. It forces a string of at least two digits, with a leading 0. It leaves on the stack the count and address of the resulting string. The rest of the word is a fudge to delete the leading 0 in a string representing a two digit value.

FORMAT is used in the word INT on screen 71, which will initialize a specified number of tracks on a disk in 1 drive (B drive in 65D nomenclature). The phrase SEB !" INIT" forces the initialization to take place on 1 drive, and puts part of the message into the command buffer. The rest of line 3 forces the parameters to be within 1 and 76, so that you don't error out. DECIMAL forces the string conversions of FORMAT to be in decimal, which it needs to work correctly. DO starts the loop. I gets the loop index, in this case the current track to be initialized. DUP 4 .R prints out the track number in a right justified field of four digits. FORMAT takes the index and makes a string out of it. DSBUF @ 5 + get the starting address of where the string should be placed in the command buffer (this is the reason for the long string of spaces in the initialization phrase). The CMOVE puts it there, and the DOS forces execution of the command buffer.

INIT, screen 130 line 12, uses a similar approach to initialize an entire disk. However, we do a bit of compile time finagling to get exactly what we want into the command buffer. For the INIT command in 65D to work correctly, the INIT must end with a carriage return. Otherwise, the command interpreter will assume a command to initialize a single track, and will try to interpret a number following in the command buffer. So we have to end the phrase in the command buffer with a carriage return. However, if we simply force the phrase to read !" INIT, <CR>", the resulting source code would have been unreadable. Instead, we compile the message with a space after it. Then we use the word [to leave the compile mode and execute directly. 0D places a carriage return on the stack. HERE points to the byte above the top of the dictionary, so the phrase HERE 1 - points to the last byte in the dictionary, which is where we want the carriage return. The C! puts it there. Then the word] resumes compilation, and DOS and SPACE are compiled as usual. Note that INIT as shown here does not force the initialization to be done on 1 drive.

Screen 119 has some code of interest to any OSI system programmer. The word ?BOOT is useful to me because I have installed a switch to allow my system to boot either as a serial or a video system. This was described in OSUIN #8. ?BOOT simply examines the same byte the boot ROM examines at reset time to determine the console. I have brought the select switch up near my two consoles, and can select the console easily. The word REBOOT eliminates the necessity to lean over and hit the reset button, and the necessity to type D for the reset code. REBOOT jumps into the boot ROM at a point immediately after H/D/M prompt. It causes the system to reboot from whatever disk is in 0 drive, and is handy for switching from FORTH to, say, WP-2 so I can write things like this. (Why doesn't BASIC or WP-2 have this facility?)

How does it work? The phrase CREATE REBOOT creates a head in the dictionary. A head contains the name of the word, and a link back to the previous entry. CREATE then initializes the code field address to point to the next address in the dictionary. In other words, CREATE prepares a head for a code definition. Well, the address we want to execute isn't the next address in the dictionary, it is $FFEA. So we substitute $FFEA for the address left by CREATE. Since HERE points to the next address in the dictionary, HERE 2 - points to the code field of REBOOT. The ! installs the address we want executed.

The next operation, SMUDGE, needs a bit of explanation. In order to compile a dictionary definition, FORTH searches the dictionary for each word to be added to the dictionary. On occasion, it is convenient to re-define a previously existing word with the old definition included as part of the new definition. So it is essential that the dictionary search never find the definition currently being compiled. This is done by setting a bit in the head at create time. SMUDGE is a word to toggle that bit. It is invoked by ; , so it is usually transparent to the user. However, in this definition, we don't have a ; . So the SMUDGE is necessary to make the new word accessible to later dictionary searches.

This definition could have been ended with a ; , but that would have cost two bytes which would never be executed. All we need is the SMUDGE, not the two bytes, so the use of SMUDGE is cheaper, if more esoteric.

★

TAPE TO DISK CONVERSION "DEPTH CHARGE"

By: Jim McConkey
7304 Centennial Rd.
Rockville, MD 20855

Last time (May Peek '85) we saw a few basic rules for conversion of games (or other programs) on tape to run on disk. This time we'll go a little bit deeper into relocation of machine code subroutines and learn a useful animation technique. This article covers the conversion of the game "Depth Charge" from the highly recommended "OSI Greatest Hits" package by Alan Stankiewicz & Bruce Robinson.

Following the program's execution brings us to a subroutine at 30000. Line 30000 itself is the usual screen clear which HEXDOS users can replace with a PRINT CHR$(3), if desired. The USR vector POKEs in 30080 must be changed along with the USR call in 31000. In 32004 we find the classic machine code subroutine load and see that it puts the routine in the usual area between $222 (546) and $2FF (767). As I have mentioned before, this area is used by HEXDOS so the routine must be moved. The corresponding USR vector in 32006 must also be changed.

The relocated version of the machine code subroutine is shown in Figure 2. You should also make a disassembled version at the original location. The subroutine is an example of a classic animation technique. A screen buffer is filled while the computer is thinking about its turn, and then the buffer is copied to the video screen in a flash. This makes the motion appear less jerky. The technique is widely used in animation programs such as Microsoft's Flight Simulator. The subroutine has three parts. The first copies the buffer to the screen. The second clears the buffer and the third draws the background scenery, which appears in the every frame, in the buffer.

Now a few words about disassembling machine code. This is much easier if you have a disassembler program, but assuming not, you will need at least a good 6502 machine code book. Start with the DATA statements containing the subroutine, in this case, line 32008. First we find a 162 (or $A2). The first byte is almost always an instruction or op-code so we look this up and find it to be an LDX# (load X register with a number). The 6502 book should also tell you that LDX# has one argument, which is the subsequent 0. The following 189 (=$BD) is an LDA#, which takes two arguments, etc. This procedure is followed until the whole program is complete. If the buffer needed to be moved (it doesn't in this case), the addresses of the buffer ($1C00, $1D00, $1E00, $1F00) would have to be changed. The references to locations $D000 etc. are the C1P video screen (remember - know your machine!). Also watch out for any absolute jumps (JMP), these will also have to be changed. In this routine, however, the only jumps are relative and do not change if the routine is relocated. The tape version of the game assumes that physical memory ends at 8191 ($1FFF) so 8192 ($2000) is a safe place to relocate the routine.

Continuing, we find a questionable POKE 76Ø,Ø in line 8. As far as I have been able to tell, this doesn't serve any purpose, and since this is the HEXDOS file control area, I deleted line 8. Next in line 15 we find 2 POKEs at 133 and 134. A quick look at our handy memory maps shows that this is the end-of-memory pointer, being changed here to make room for the screen buffer at $1CØØ. Depth Charge has no provision for quitting the game, so you must reboot after a game to restore full memory. I have added a quit option to the program which restores the original end-of-memory pointer, which is saved in the new line 8 before room is reserved for the buffer. This has been changed from line 15 because the optional re-run enters at line 9 meaning that a second run with the vector stored at 15 would set the end-of-memory vector to its present value, namely below the screen buffer. In line 2Ø are POKEs to 6Ø1 and 6Ø2. This lies in the middle of the original machine code subroutine and must be changed. An inspection of the original and relocated routines shows that these new locations are 8243 and 8244 ($2Ø33,2Ø34).

A bit later, in lines 2ØØ-46Ø we find many POKEs. These are a bit tricky to follow, but they all POKE to screen buffer. Next we encounter the unusual statement 49Ø IF ML=. THEN 6ØØ. The "." can be thought of as a decimal point with nothing around it and is the same as saying IF ML=Ø THEN 6ØØ. The main reason for using the "." is that it runs about 25% faster than the same line using a Ø. The POKEs in lines 55Ø and 599 (POKE 53Ø,x) enable and disable the CTRL-C check and should not be changed.

Further down, in line 7Ø5, the USR call will again have to be changed to USR(-7) under HEXDOS. This is also the case in line 5Ø4Ø, where the USR vector must also be changed. This line calls the scan keyboard routine and waits for the player to hit "R" to run again. This is where we add the new quit option. New line 5Ø25 prints the option and line 5Ø44 checks for "Q" being pressed. If "Q" is pressed, the old end-of-memory pointer is restored and the game is ended. One last USR vector needs to be changed in 5Ø5Ø.

The program will seem to work with the mods outlined so far for a while then it will die due to lack of memory. Deleting the useless comments at 18ØØ-18Ø6 will free up about 3ØØ bytes which is enough to let the program function normally.

There you have it, another game converted to disk, and a couple of techniques to speed your programs and smooth out your graphics animation. Write if you have any particular requests for tape to disk conversions and we'll see what we can do.

★

LISTING 1 - HEXDOS version of Depth Charge

```
Delete lines 15 and 1800-1806

Change these lines (these are the changed versions):
8      H1=PEEK(133):H2=PEEK(134*:POKE133,0:POKE134,28
20     J=PEEK(8243)+256*PEEK(8244)
705    X=USR(-7):IFDHTHEN1100
5040   POKE240,0:POKE241,253:X=USR(-7)
5050   POKE240,0:POKE241,32
30000 PRINTCHR$(3)
30080 POKE240,0:POKE241,253
31000 X=USR(-7):RETURN
32004 FORQ=8192TO8252:READM:POKEQ,M:NEXTQ
32006 POKE240,0:POKE241,32

Add these lines:
5025 PRINT"(hit Q to quit)"
5044 IFPEEK(531)=81THENPOKE134,H2:POKE133,H1:END
```

LISTING 2 - Relocated machine code subroutine

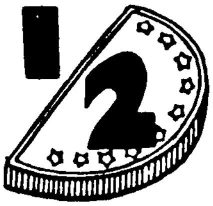
```
10     0000                 .OPTION L 2
                            ;
                            ;DEPTH CHARGE MACHINE CODE SUBROUTINES
                            ;
120    0000                 PC=$2000
                            ;
200    2000 A2 00                  LDX# 0     ;COPY BUFFER TO SCREEN
210    2002 BD 00 1C        LOOP1  LDAX $1C00
220    2005 9D 00 D0               STAX $D000
230    2008 BD 00 1D               LDAX $1D00
240    200B 9D 00 D1               STAX $D100
250    200E BD 00 1E               LDAX $1E00
260    2011 9D 00 D2               STAX $D200
270    2014 BD 00 1F               LDAX $1F00
280    2017 9D 00 D3               STAX $D300
290    201A CA                     DEX
300    201B D0 E5                  BNE  LOOP1
                            ;
400    201D A9 20                  LDA# $20   ;FILL BUFFER WITH BLANKS
410    201F 9D 00 1C        LOOP2  STAX $1C00
420    2022 9D 00 1D               STAX $1D00
430    2025 9D 00 1E               STAX $1E00
440    2028 9D 00 1F               STAX $1F00
450    202B CA                     DEX
460    202C D0 F1                  BNE  LOOP2
                            ;
                                              ;DRAW WATER LINE IN BUFFER
510    202E A9 87                  LDA# $87   ;CHAR FOR WATER
520    2030 A2 1F                  LDX# 31    ;# OF CHARS TO PRINT
530    2032 9D 00 1D        LOOP3  STAX $1D00
540    2035 CA                     DEX
550    2036 D0 FA                  BNE  LOOP3
560    2038 60                     RTS
                            ;
600    2039 60                     .BYTE 96   ;?????
610    203A 60                     .BYTE 96
620    203B 24                     .BYTE 36
630    203C 24                     .BYTE 36
640    203D 24                     .BYTE 36
                            ;
```

LETTERS

ED:

First I would like to thank you for your prompt answers to my letters in the past....it has been a great service. This time maybe I can give some answers!

I wrote some time ago about an intermittent problem I had with my C1P MF blitzing disks on cold boot. You answered that you were not aware of any "race" condition as described by Bill Thompson at OSI, but a letter by T. G. Moore in PEEK(65) March 1985, page 21, describing a fix to WP6502



1/2 PRICE INVENTORY SALE

OUR STOCK ROOM IS OVERFLOWING!

FILL YOUR LIBRARY WITH MISSING MANUALS FOR LESS THAN 1/2 PRICE

All starred items are at 1/2 the marked price
Foreign orders by VISA/MASTER/CHOICE only, plus postage.
Orders must be postmarked not later than August 31, 1985.
Orders **can not** be sent to P.O. Box addresses.

GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

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

( )	**C1P Sams Photo-Facts Manual.** Complete schematics, scope waveforms and board photos. All you need to be a C1P or SII Wizard, just	$7.95	$ ______	★
( )	**C4P Sams Photo-Facts Manual.** Includes pinouts, photos, schematics for the 502, 505, 527, 540 and 542 boards. A bargain at	$15.00	$ ______	★
( )	**C2/C3 Sams Photo-Facts Manual.** The facts you need to repair the larger OSI computers. Fat with useful information, but just	$30.00	$ ______	★
( )	**OSI's Small Systems Journals.** The complete set, July 1977 through April 1978, bound and reproduced by PEEK (65). Full set only	$15.00	$ ______	★
( )	**Terminal Extensions Package** - lets you program like the mini-users do, with direct cursor positioning, mnemonics and a number formatting function much more powerful than a mere "print using." Requires 65U.	$50.00	$ ______	
( )	**RESEQ** - BASIC program resequencer plus much more. Global changes, tables of bad references, **GOSUBs** & GOTOs, variables by line number, resequences parts of programs or entire programs, handles line 50000 trap. Best debug tool I've seen. MACHINE LANGUAGE - VERY FAST! Requires 65U. Manual & samples only, $5.00 Everything for	$50.00	$ ______	
( )	**Sanders Machine Language Sort/Merge** for OS-65U. Complete disk sort and merge, documentation shows you how to call from any BASIC program on any disk and return it or any other BASIC program on any disk, floppy or hard. Most versatile disk sort yet. Will run under LEVEL I, II, or III. It should cost more but Sanders says, "...sell it for just..."	$89.00	$ ______	
( )	**KYUTIL** - The ultimate OS-DMS keyfile utility package. This implementation of Sander's SORT/MERGE creates, loads and sorts multiple-field, conditionally loaded keyfiles. KYUTIL will load and sort a keyfile of over 15000 ZIP codes in under three hours. Never sort another Master File.	$100.00	$ ______	
( )	**Assembler Editor & Extended Monitor Reference Manual** (C1P, C4P & C8P)	$6.95	$ ______	★
( )	**65V Primer.** Introduces machine language programming.	$4.95	$ ______	★
( )	**C1P, C1P MF, C4P, C4P DF, C4P MF, C8P DF Introductory Manuals** ($5.95 each, please specify)	$5.95	$ ______	★
( )	**Basic Reference Manual** — (ROM, 65D and 65U)	$5.95	$ ______	★
( )	**C1P, C4P, C8P Users Manuals** — ($7.95 each, please specify)	$7.95	$ ______	★
( )	**How to program Microcomputers.** The C-3 Series	$7.95	$ ______	★
( )	**Professional Computers Set Up & Operations Manual** — C2-OEM/C2-D/C3-OEM/C3-D/C3-A/C3-B/C3-C/C3-C'	$8.95	$ ______	★

( ) Cash enclosed ( ) Master Charge ( ) VISA

Account No. ______________ Expiration Date ______________

Signature ______________

Name ______________

Street ______________

City ______________ State ______________ Zip ______________

TOTAL	$______
MD Residents add 5% Tax	$______
C.O.D. orders add $1.90	$______
Postage & Handling	$ 3.70
TOTAL DUE	$______

POSTAGE MAY VARY FOR OVERSEAS

gave me an idea. I then added a subset of the program (lines 20 thru 130) he supplied and wrote it to EPROM for the monitor code that is run before the cold boot menu is displayed. So by the time you select the "disk" option, the disk drive PIA has already been initialized (and incidentally selects the B drive if you just powered up). When the normal boot-up begins, the A drive is selected and everything comes up roses!!!! As a result I have not blitzed a disk since I installed that change, which has been about 6 months.

I should point out that I observed by system blitzing disks only after power-up, and I suspect the head load relay delay on the old MPI drives prevents this problem from occurring (my Shugart SA455's don't have head load relays).

I also noted the letter by C. J. Hipsher in the June issue who is having problems with his Cannon drives. I have had this problem myself and there are some things he may have overlooked. He should try removing the "pullup resistor pack" from each of the Cannon drives to see if the problem will go away. These resistors will load the control line drivers too heavily and produce slow control signals. Also, he may have a problem with the speed it takes the drives to come "ready" after selection. The problem he describes seems like it should not matter whether the Cannon is the source or the destination and would probably make it totally unusable on his system (I guess I am puzzled how the double stepmode he describes allows him to use the Cannon drive at all!).

Thanks again for your great service.

Glenn Davis
Endicott, NY 13760

* * * * *

ED:

I'm responding to Robert Dingle's letter in the June issue.

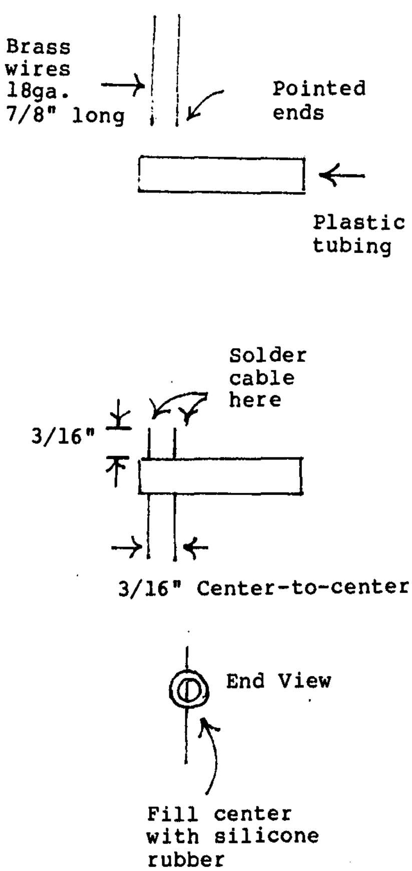
Enclosed is a drawing of a home-made plug for J2, 3, or 4 cn the 600 board. Materials required are some 1/4" o.d. plastic tubing, some 18 ga. brass wire, and some silicone rubber adhesive.

1. Cut a piece of tubing to the desired length - long enough to span the number of pins you wish to use.

2. Cut a piece of brass wire 7/8" long for each pin you need. Shape one end to a chisel point or needle point. Heat this end enough to melt its way through the plastic tubing and stop with 3/16" protruding from the back end (see drawing).

3. Fill body of plastic tubing with silicon adhesive and allow to cure.

4. Solder cable wires onto short end of brass wires.



All of these materials should be obtainable at most hardware and/or hobby stores. The 1/4" plastic tubing is what we use down here for water supply lines to evaporative coolers. It comes in either black or clear, and is flexible (not rigid).

Bruce Showalter
Abilene, TX 79601

* * * * *

ED:

I was interested in the conversion of MINOS to disk article in the May '85 issue of Peek. I noted an error in line 1310?....M=40 should be .... HM=40 (I think). Anyway, I must have done something wrong, but haven't gone back to figure it out.

The reason was that I got side tracked converting ROACH TRAP to disk. (I have 5 1/4" 65D V3.3). The advertising says it only works on the C1P and I hadn't seen anything to the contrary. Although I don't have a standard C2-4 system, I do have its polled keyboard and 32 x 64 display screen.

First, the main problem for ROM or Disk is line 50210 POKE 630,64. It should be POKE 679,64. For Disk, I had to move all the POKEs higher into RAM (any $02XX address is changed to $72XX and any $1XXX address is changed to $7XXX). Also, the screen clear in lines 4 & 6 needs to be changed to clear a 2K area instead of the 1K area. And of course, most DATA lines need to be changed.

So the end result is that a 3 dimension maze is more fun than a 2 dimension one.

Loren Jacobson
Lennox, SD 57039

* * * * *

Loren:

You are correct, there is a typo in LINE 1310, it should read:

```
1310 SR=600:VM=20:HM=20:
IF PEEK(57088)<127THEN HM=40
```

I also found the following omissons:

1. LINES 20,50,60 & 25065, all dealing with the replaced M.L. routine, should be deleted.

2. LINE 30005 should read:

```
30005 PRINT CHR$(3):V=2:
   GOSUB1200:POKE 240,0:
   POKE241,253:X=USR(-7)
```

Sorry for the confusion. I loaded MINOS from tape and performed the mods as listed in the May article with the above changes, and it ran perfectly. Perhaps I should have mentioned that these mods assume more than 8K of RAM. HEXDOS, unlike OS65D will run in 8K with plenty of room to spare.

I am also converting Roach Trap (a 3-D MINOS) to disk. I will be in touch with you, maybe we can get together and write an article for Peek.

Jim McConkey
Rockville, MD 20855

* * * * *

ED:

I was very pleased to see Mr. Jankowski's Wazzat Corner! article on the OSI video ROM in the June '85 issue of PEEK(65). He has done some work that has been on my back burner for many years.

This has also prompted me to report two bugs in the OSI video ROM. One is that two ASCII characters have been transposed in position in the table. The other is that a non-ASCII character has been substituted for an ASCII character.

The substitution is that a divide character (÷) has been substituted for the tilde (~), hex 7E.

The character swap is that the right brace (}) has been swapped with the vertical bar (|). These are hex 7D and 7C, respectively. Neither of these is particularly noticeable to BASIC users, or to most Assembly language programmers. This is especially due to the fact that OSI's polled keyboard routine does not support any of the three characters.

Obviously, the preferred method of handling these bugs is Mr. Jankowski's EPROM substitution method. Failing that, a software solution is to be sought.

As I work exclusively in real-FORTH, the software solution to the switch problem will be shown in Assembly written under real-FORTH.

The solution is quite simple. Outgoing characters are found on the top of FORTH's stack. Each is tested for being either the right brace or the vertical bar, and if the test is true, bit 0 of the character is toggled.

The code might look like this:

```
TOP LDA, FE # AND, 7C # CMP,

IF, BNE, 1 # LDA, TOP EOR, TOP
STA, THEN,
```

This code is inserted into the video version of (EMIT). It leaves the corrected character on the stack for later processing by the normal video output routine. Alternatively, it can be incorporated into a video routine written in real-FORTH, which is what I have done.

If you are contemplating a patch in OS-65D, it should be done after the I/O distributor, in order to avoid sending the "corrected" characters out to a printer. This might involve patching 65D's dispatch table at hex 2301. As I no longer use the I/O dispatcher or any of OSI's console support code, I will leave that patch to others.

Charles Curley
Long Beach, CA 90804

* * * * *

ED:

Is there anyone out there who has developed good graphics capability with hardcopy output for 65U? I don't want to reinvent the wheel.

Richard E. Reed
411 N. Mill Street
Tehachapi, CA 93561

AD$

KEYWORD and CP/M v 2.25. OSI's answer to WORDSTAR. Was sold at $400 each, now reduced to $200 each, or $350 for the pair. Reply PEEK, Box K, c/o PEEK(65), P.O. Box 347, Owings Mills, MD 21117.

* * * * *

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

*** * * OS-65D V3.2 * * ***
DISASSEMBLY MANUAL

Published by Software Consultants, now available through PEEK(65) for $25.95 including postage. Overseas add extra postage (weight 16oz). Make check or money order (in U.S. funds, drawn on a U.S. bank) payable to PEEK(65), P.O. Box 347, Owings Mills, MD 21117.

* * * * *

FOR SALE: C3S1 with almost new Siemens 8" drives: 2 new Siemens 8" drives w/box and power. Best reasonable offer. Tom McGourin (219) 429-4160 days, (219) 489-6001 weekend/ eves.

* * * * *

FOR SALE: Software for non-C1P disk systems:

(1) Term-Plus; smart terminal program for OS-65D V3.3. Captures on disk. Sends BASIC or ASM programs as text. Sends WP-2 or WP-3 files as formatted text. Ten programmable keys for polled keyboards. Error-free file transfers and cursor control on CompuServe. $25.00

(2) Term-32; for OS-65D V3.2. Same as Term-Plus, but lacks cursor control. $25.00

(3) Term-65U; for OS-65U. Captures on disk. Transmits BASIC programs as text. Sends WP-3 files as formatted text. Sends records from OS-DMS files, with sorting. Error-free file transfers on CompuServe. $50.00

(4) ASM-Plus; fast, disk-based assembler for OS-65D V3.3. Allows linked program files so that memory restrictions are reduced. 8 times faster than OSI Assembler. Includes file editor. $50.00

(5) Edit-Plus; almost a word processor for OS-65D V3.3. Great for editing assembler source files and small documents. Supports roughly 90% of WP-3's features. Free form text entry. Right justifies by inserting spaces. $40.00

(6) Data-Plus 65U Mail-Merge; merges selectable fields from OS-DMS files into WP-3 documents. Outputs to printer or to disk files (for printing later or transmission via Term-65U). $50.00

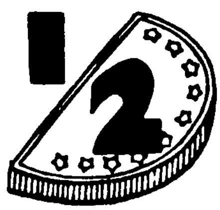
Specify computer and disk size. Richard L. Trethewey, 8 Duran Court, Pacifica, CA 94044.

* * * * *

PEEK [65]
The Unofficial OSI Users Journal

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

DELIVER TO:



PRICE INVENTORY SALE

OUR SHELVES ARE BULGING!

HERE'S YOUR CHANCE TO COMPLETE YOUR LIBRARY AT LESS THAN 1/2 PRICE

Get a one year volume set (12 back issues) for $15.00 and we will pay UPS.
Get one back issue of the OSIO newsletter free with order.
Foreign orders by VISA/MASTER/CHOICE only, plus postage.
Orders must be postmarked not later than August 31, 1985.
Orders can not be sent to P.O. Boxes.

NAME..............................**STREET**..........................

CITY..............................**STATE** ....

ZIP CODE..........................**COUNTRY**........................

Please send me the following volume(s). I enclose:

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	OCT #10	NOV #11	DEC #12