PEEK (65)

The Unofficial OSI Users Journal

FEBRUARY 1985
VOL. 6, NO.2

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

INSIDE

Column One

This being early January, others are still recovering from the Holidays, but PEEK has been busy back in December getting together another packed issue ready for you. Speaking of last year, let's start off with taxes.

In the April 1984 issue, Bob Baldassano announced his new Income Tax package. A bit late, and further delayed due to our publishing lead time, but one creates when one creates. Now Bob has had the rest of the year to spit and polish the program, bring it up to date and make it available to you good tax payers in plenty of time to make use of it. In addition, it is available for just about every Op Sys OSI ever had. Bet you never thought that tax preparation could be fun!

We have received some super articles recently and know that you will enjoy them in the months to come, but one thing has become evident in talking with the authors. Our writers are not necessarily English majors or professors. In fact, over the last year, only a couple of the articles that you have read were prepared by pros.

The point is that you shouldn't be worried about the technicalities of the English language. Just get the thoughts down in an accurate and logical fashion. That shouldn't be too hard for anyone who deals with that accurate and logical language - BASIC. Just let our people worry about spelling and where the commas should go. Just keep the articles and letters coming.

Speaking of letters, frequently we answer your questions by return mail and publish them later. This gets you a speedy answer, but with postal rate increases, it is eating us out of house and home. In the future, if you want a quick answer, and we are only too happy to do so, please include a self addressed stamped envelope.

Housekeeping done, let's move on to what is new out in Denver with the DBI folks. For starters, they are holding a meeting on January 24 and 25 for Distributors and OEMs to bring them up to snuff on all the latest with the DB-1 machine and its new additions in the hard disk line. They are getting bigger and bigger. The list of optional disks has now grown to include an 85 MB and a 190 MB. Just around the corner is a 300 MB disk. And remember that you can put a pair of these in the box and tell it that it's one big disk. Now, a 600 MB is one heck of a lot of storage.

But the storage story doesn't end there. Maybe you don't need anything that big, but it sure would be nice if there was a removable disk. There is! A half-high 5 1/4" removable that holds 10 MB. That's more in my realm of comprehension. And there is yet more. How about half-high cassette streamers with capacities of 25, 40 and 60 MB.

Other things that the attendees will see are a preliminary machine running under the 65C8/16 CPU with a 65U emulator that looks like it will run about 99% of the existing OS-U programs with many of the advantages of 16 bit operation AND they will probably get a few words about the 68000 machine that is reported to be progressing very well, but no dates set as yet.

Somewhere back on the drawing boards is a board that will allow many of these new storage devices to be used with most any OSI 48 pin bus machine. Are you interested in a 60 MB C4P? I'll settle for 10, personally.

I know that things are happening at OSI too, but because the OSI hierarchy travel and are hard to reach and because I am writing this in early January due to an European trip, we just haven't been able to make the connection. We will try to make up for it next time.

Eddie

MAPPING MACHINE LANGUAGE CODE

If you've ever wanted to thoroughly document, explore, and understand your computer's BASIC or Operating Systems - the techniques and programs here are your tools. Written for OSI, these ideas can be modified for other computers.

RESOURCE PART 1

Courtesy of COMPUTE!
By: T. R. Berger
Coon Rapids, MN

Have you ever tried to document your machine software by annotating disassemblies? Have you ever tried to move these programs by reconstructing assembler source listings from disassemblies? If so, you know what a huge investment of time is needed. This article covers a group of BASIC programs which will facilitate regenerating fully documented assembler source listings starting from machine language programs in much less time than the painful direct route.

When I undertook to write these programs, I did not even dream how powerful they would be. I never really anticipated regenerating a source listing of 8K OSI Microsoft Disk BASIC. When I realized that this task could be done, what was one simple program expanded into the four presented here. A much modified and improved version of the single program which started this all off is also included here. If OS65D would allow six buffers to be open at once, these programs could be vastly speeded up and simplified.

These programs are written in BASIC for disk based OSI computers. However, the programs are carefully documented so that those using other 6502 machines with different disassemblers should have no difficulty in copying the idea. The programs accept as input an ASCII file produced by the OSI version of the Apple disassembler (see DR. DOBBS JOURNAL, Sept. 1976, p. 22). The output is a collection of ASCII files which include the following:

1. An assembly source listing of code which will reassemble at the same location without further editing.

2. Equate files necessary to run the assembly source through an assembler.

3. Separate cross reference files for each of the following:

a. Zpage addresses,
b. Jumps and jumps to subroutines,
c. Memory calls, and
d. Branches.

A single pass program RESOURCE S is included for resourcing small programs. On a 48K C8PDF it has no trouble handling OS65D. Since only symbols and cross references are kept in memory, a 32K machine should also have no trouble. Cross reference strings in RESOURCE S are of limited size so that the program will crash in attempting to cover 8K BASIC. The Zpage cross references to $AC overrun about halfway through. Since RESOURCE S is a compressed version of the program package presented, I will comment very little on it. When there are a very large number of cross reference strings, the program slows way down due to garbage collection. In Microsoft BASIC, garbage collection times go up approximately as the square of the number of strings in memory (and not their size).

RUN TIMES APPROACHED 24 HOURS

I have written this package so that hobbyists can understand their most commonly used language: BASIC. A source file for 8K BASIC is colossal. Therefore, many shortcuts are necessary to complete the resourcing task. I originally tried to enlarge RESOURCE S to cope with the job. OS65D has only two disk buffers requiring that a large amount of information be kept in memory for a single program. So many strings were generated and garbage collection time became so great that run times approached 24 hours. Clearly this is not the way to go. I broke the task into small pieces, each being completed in a reasonable amount of time.

On 8" flopppies the BASIC disassembler source ($03A1-$2300) takes 28 tracks (84K). Those using minifloppies must tackle BASIC in three or more passes, using the cross reference tables to properly join the final product.

The final product, scattered through several files, takes up about 36 tracks. There is no hope of assembling these files without a linking assembler. (Leroy Erickson has written such an extension for the OSI Assembler.) However, printout of the source and the cross reference tables greatly simplifies the annotation and documentation process. After one pass of RESOURCE S over OS65D, it was possible to reassemble OS65D at the same location. After about two hours of editing, a file was obtained which assembles anywhere.

Using my maps of OS65D and Jim Butterfield's maps of BASIC, you should be able to obtain fully documented source listings of both BASIC and OS65D. I would hope to see more articles using specific parts of OS65D and BASIC. Namely, what are some subroutines, how do they work, how does one use them, and how does one resource them?

The entire program package presented here is written in BASIC. This sped implementation and modification time. It also makes the programs easier to understand. The price paid is runtime, which is considerable over 8K BASIC. Efforts have been made to optimize runtimes, especially on inner loops. This adds steps to the process, but significantly reduces program running times.

Of course, one must edit the files generated by these programs. I use a group of utilities which constitute a useful BASIC text file editor and processor.

The three most useful utilities are a transfer program to move large text files around, a print program to output large files to a printer, and a fast sorter to sort symbol tables. A further useful addition is a large text file single pass character-oriented line editor.

Copyright • 1985 PEEK (65) Inc. All Rights Reserved.
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

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

HOW IT WORKS

The first program (PASS 1) takes the disassembly listing (which I will call SOURCE) and compresses it into a scratch file (which I call SCRATCH). The main working file is SCRATCH. It is about 25% smaller than SOURCE and serves as input to the other programs. A typical line of SOURCE looks as follows:

```
1A3D BD11B0 LDA $B011,X.
```

In SCRATCH this same line would be:

```
1A3D LDA HHB011,X.
```

The code field has been eliminated and $B011 has been changed to a six letter symbol. All four digit operands $XXXX are changed to six letter symbols HHXXXX, which is the maximum size for symbols in OSI's Assembler. Except for immediate operands, two digit operands $YY are replaced by six letter symbols HHZZYY. Further, the first H in every operand is always aligned as the eleventh letter in a line. BASIC is much too slow to search a line for a symbol. Aligning symbols makes them easy to find when editing. For example,

```
MID$(IN$,11,6)
```

removes a symbol from a line IN$. The 'H' in position eleven distinguishes a symbol. The 'Z' in position thirteen distinguishes a Zpage reference.

A line in SOURCE

```
1A40 FF ???
```

would appear in SCRATCH as

```
1A40 .BYTE $FF.
```

This step makes the resource file assembler ready. Bad disassembly of opcodes must be fixed by editing the final file if a true source file is needed. In particular, tables and text are not resourced correctly, only made assembler-ready.

The first program also builds a table of two byte operands (which I will call SYMBOL). SYMBOL is used in PASS 2 to generate labels and an equate file of two byte operands. Since SYMBOL is searched repeatedly in PASS 2, it must be sorted. Sorting SYMBOL means a fast binary search can be used which is many times faster than a sequential search. (For BASIC, this addition reduced line process times in PASS 2 from about 5 seconds per line to less than 1 second per line.) Since BASIC requires 800 symbols, this search method cuts hours off PASS 2. Accordingly, PASS 1 keeps a sorted symbol table.

PASS 2 generates the resource file (which I call OBJECT). It reads one line of SCRATCH:

```
1A3D LDA HHB011,X.
```

It searches SYMBOL for 1A3D. If 1A3D is found, a numbered line

```
10000 HH1A3D LDA HHB011,X
```

is output to OBJECT. Since 1A3D is now defined by a label, it is marked as 'used' in SYMBOL. If 1A3D is not found, a numbered line

```
10000      LDA HHB011,X
```

is output to OBJECT. After OBJECT is complete, the unmarked symbols in SYMBOL are operands which are not defined by labels in OBJECT. Thus, an equate file (which I call EQUATE) is written using these unmarked terms from SYMBOL. For example, if 1A3D is unmarked, it would be written to EQUATE as a numbered line

```
5000 HH1A3D = $1A3D.
```

Except for Zpage labels, OBJECT and EQUATE are ready for the assembler.

PASS 3 generates the various symbol tables. The symbols are picked out of SCRATCH along with their addresses. A symbol HHXXXX is stored in a string SS$(I) as XXXX. A check is run to see if the symbol already appears in the table. If it does not, the counter SN is incremented and the symbol is added. This list is stored as a sorted table.

Suppose that HHXXXX appears in Line YYYY and that SS$(I) = XXXX. Then UYYYY is appended to the right hand end of SA$(I) where U is chosen to give information about the opcode on line YYYY. Some thought went into the choice of U. In the branch table, the middle letter of a branch instruction ~~comes~~ closest to distinguishing all branches. Thus U is the middle letter of the opcode. Again in the JMP and JSR table, the middle letter distinguishes JMP from JSR. Thus U is M or S in this case. The first letter of the opcode is chosen for the memory table.

In decoding programs, I have found that the most important fact to know about Zpage opcodes is their addressing mode. That is, is an opcode indexed or not? Thus, U is the extreme right hand symbol of the disassembly line. This includes),X, and Y. It is not possible from this to tell whether the Y means indexed or indirect indexed. However, given the simplicity of this approach, it is adequate.

If SA$(I) becomes too long, it is written to a cross reference file and SA$(I) is emptied. (In RESOURCE S this step is not performed, the program bombs when SA$(I) becomes too long.) These "long strings" will appear out of order in the file. (The first few cross references may be out of order.) The symbol table can be resorted by most any sorting program. As it stands, the table is "almost in order."

PASS 4 generates the Zpage equate file which I call ZEQUATE. This is done using the Zpage cross reference file generated in PASS 3. The file resembles the EQUATE file.

In resourcing a large program, there will not be enough room on one disk for all the files generated. SCRATCH, and various other files may be moved using a transfer utility. Symbol and cross reference files may be sorted using a sort utility. Final files may be printed using an output utility.

Example 1 shows the OBJECT file (resourced assembly language) for the beginning of the disassembler in the Extended Monitor. Example 2 gives the two equate files. Example 3 gives the output from the Assembler using these three files. Example 4 gives the four cross reference tables. The first address in each row is the symbol. The other addresses following are the cross references, with some indication as to opcode.

Example 1.

```
10000          .BYTE $17
10010          LDA #$16
10020          STA  HHZZC7
10030 HH18DD   JSR  HH18ED
10040          JSR  HH19D1
10050          STA  HHZZC5
10060          STY  HHZZC6
10070          DEC  HHZZC7
10080          BMI  HH1922
10090          BNE  HH18DD
10100 HH18ED   JSR  HH19BC
10110          LDA (HHZZC5,X)
10120          TAY
10130          LSR A
10140          BCC  HH1901
10150          LSR A
10160          BCS  HH1910
10170          CMP #$22
10180          BEQ  HH1910
```

Example 1. continued

```
10190          AND #$07
10200          ORA #$80
10210 HH1901   LSR A
10220          TAX
10230          LDA  HH17A5,X
10240          BCS  HH190C
10250          LSR A
10260          LSR A
10270          LSR A
10280          LSR A
10290 HH190C   AND #$0F
10300          BNE  HH1914
10310 HH1910   LDY #$80
10320          LDA #$00
10330 HH1914   TAX
10340          LDA  HH17E9,X
10350          STA  HHZZC1
10360          AND #$03
10370          STA  HHZZC2
10380          LDA  HHZZC8
10390          BNE  HH1923
10400 HH1922   RTS
10410 HH1923   TYA
10420          AND #$8F
10430          TAX
10440          TYA
10450          LDY #$03
10460          CPX #$8A
10470          BEQ  HH1939
10480          LSR A
10490          BCC  HH1939
10500          LSR A
10510          LSR A
```

HOW TO USE IT

STEP 1) Creating a SOURCE file.

If you plan to resource BASIC, you must move the Extended Monitor since it overlays part of BASIC. I find it handy to have the Extended Monitor available while BASIC is resident.

After trying several methods, I've decided that the following is the easiest way to generate a SOURCE file. It uses the disk output capability of OS65D. The code you are resourcing should not overlay the disk buffer used. (Video with polled keyboard is assumed; otherwise, re-check the I/O flags.)

a) Initialize a fresh disk.

b) Copy the directory Track D onto this disk using OS65D's copy utility (D is Track 8 on 8" floppies).

c) Create files for all empty tracks except Tracks 0 and D. Delete all directory entries on the new disk.

d) Load the machine language program to be resourced.

e) Load and run the Extended Monitor.

We must now set all the various pointers for a disk buffer. To resource BASIC you need a very large file. Let the first available track be N where the directory is on Track N-1 (N=9 on 8" floppies).

f) Choose a first track number N for your SOURCE file. Let M be the last track number on the disk (M=76 on 8" floppies). Do not choose N so that either N=0 or the directory track is included in the range of N to M.

g) Using the "at" (@) sign command, set the following buffer values. (These are valid for OS65D V3.2, i.e. 8" floppies. The correct values for minifloppies are given in the OS65D User's Guide.)

ADDR($)	ADDRS(D)	VALUE	
2326	8998	7E	BUFFER START ADDRESS
2327	8999	31	
2328	9000	7E	BUFFER END ADDRESS
2329	9001	3D	
232A	9002	N	FIRST TRACK OF FILE
232B	9003	M	LAST TRACK OF FILE
232C	9004	N	CURRENT BUFFER TRACK
232D	9005	0	DIRTY BUFFER FLAG
23C3	9155	7E	ADDRESS DISK OUTPUT
23C4	9156	31	(CURRENT BUFFER ADDR.)

h) Mount the fresh disk.

i) From EM type (i.e. turn on disk output)

!IO ,22 <return>.

The next few steps write directly to the disk without error correction. If you make an error, perform step 1) and restart at step g. Presumably you know the start ($XXXX) and the finish ($YYYY) addresses of the code to be resourced. For BASIC these are $XXXX=$03A1 and $YYYY=$2300. For OS65D these are $XXXX=$2336 and $YYYY=$2E1E. For the ROMs these are $XXXX=$FD00 and $YYYY=$FFFA.

j) Commence disassembly with

QXXXX <return>.

k) Put your finger on LINEFEED. Hold it there until $YYYY has been disassembled. Then hit RETURN.

All but the last track of the SOURCE file is on the disk. The last track is still in the buffer. This last part is also missing an "end of file" marking. The next few steps turn off the disk output long enough to make corrections, then turn it back on to write the final track to the disk.

The next step turns off disk output by creating a syntax error, and puts a mark to help find the end of the SOURCE file.

l) Type

!XIT <return>.

m) Search for the end of the file

W!XIT>317E,3D7F.

If all has gone well, you will receive a message

(*) VVVV/21

where VVVV is the address of ! in the expression !XIT. If you do not receive such a message, it is possible (but unlikely) that a "disk write" occurred in the middle of the word !XIT. Go back to step g) and start again. When you reach step k), hit RETURN five times instead of just once, then proceed. If you do not receive the message (*), something is definitely wrong somewhere. Start a careful search (Beware: some values given only work for 8" floppies).

n) Using quotes and "at" check the following

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

o) Make the following change.

VVVV/21 0D

Now the "end of file" marker is properly installed. Next we write the buffer to the disk.

p) Make the following pointer change.

```
23C3/YY 7E
23C4/WW 3D
```

q) Write down the value TK

232C/TK

r) Type

!IO ,22<Return>

The entire SOURCE file is on the disk. It starts on Track N and ends on Track TK. We must now create a directory entry for this file.

s) Load BASIC and CREATE, but do not run CREATE. (You may need a different disk to do this.)

t) Delete line 20290 (which would erase all the work you have done). It reads:

```
20290 DISK!"IN"+T$:DISK!
  "SA"+T$+",1=317/"+P$
```

u) Run CREATE and name the SOURCE file on your new disk.

STEP 2) Create a SCRATCH and SYMBOL file entry.

These files must be on the

same disk as SOURCE. Be sure to reload CREATE so that line 20290 is not missing. SCRATCH can be about 25% smaller than SOURCE. SYMBOL can be 1-2 tracks. On 8" floppies, 8K BASIC needs two tracks for SYMBOL (4K file size). You may put these files anywhere as long as they do not overlap the directory track, Track 0, or the tracks used by SOURCE.

STEP 3) PASS 1.

Run the first resource program. Prompting will tell you what to do. The new disk must be in the drive throughout the run. The screen will display the current status. On large programs, be prepared for several-minute waits for garbage collection. A five minute wait between screen data lines probably means there has been a system crash. This program will not work with ROM BASIC since the garbage collector is defunct. (See PEEK(65), March 1980, p. 3 for a fix.)

The SOURCE, SYMBOL, and SCRATCH file may fill a disk, so you may have to move some files to other disks. SOURCE is no longer needed, but should be saved in case of trouble. Symbol is needed for PASS two and SCRATCH is needed for PASSes two and three. Using a transfer utility you may move SCRATCH and SYMBOL to a new disk.

STEP 4) PASS 2.

The second resource program generates an EQUATE file and the resourced assembly listing OBJECT. Create such files on a disk containing SCRATCH and SYMBOL, EQUATE need not be large, usually much less than a track. OBJECT should be slightly larger than SOURCE.

The next step creates all the cross reference tables. Each table needs its own file. SCRATCH is the input file. The branch table will probably be the largest file.

STEP 5) PASS 3.

Repeat this step until all cross reference tables are complete. Only Zpage cross references are essential. However, I find the Zpage and JSR tables the most useful. You may wish to sort these tables, even though they are "almost sorted."

STEP 6) PASS 4.

Create the Zpage equate file: ZEQUATE. Input to this program is the Zpage cross reference file. This step is the final one which creates the list of Assembler Zpage equates.

Any of the files generated may be dumped to a printer using a printer utility. The process is much simpler than it sounds. The single pass resource program eliminates most steps if only small programs are being resourced.

MOVING ASCII TEXT FILES TO THE ASSEMBLER

For small programs, the resource can actually be assembled by the OSI Assembler. The three files (OBJECT, EQUATE, and ZEQUATE) must be merged and the program counter location given (10* = $XXXX).

The resourced files are ASCII text files with an end of file (EOF) marker:

```
XIT <return>.
```

Since OSI's Assembler does not keep an ASCII file, more is needed. We must transfer the disk text files into the Assembler/Editor. In OS65D it is easy to reset output flags with:

```
IO ,02.
```

However, only one input is recognized and, if this is not the keyboard, then keyboard input is dead. During disk input, the keyboard is disabled. In particular, OS65D has no way of recognizing the end of a file except by an operating system error. This is a definite deficiency in OS65D. When an operating system error does occur, the IO flags are properly reset to default values.

If a file is on Tracks 2 and 3, inputting these tracks will result in a system error as soon as Track 3 is finished. The trouble is that the actual file may end halfway through Track 3. The rest of Track 3 may contain absolutely destructive information, such as Assembler commands or operating system commands. My favorite is the following. The ASCII character "left bracket" occurs as input opening the Indirect File. This file fills up memory, wiping out everything in the way. It eventually reaches the disk addresses. You hear a thunk and the disk goes dead. If input continues, it next reaches the screen memory filling the screen with jazzy characters. It goes on to the color memory, tone generator, etc. You've probably had this occur and wondered what happened. It's just the Indirect File, filing all the garbage away.

One solution is to remove the destructive information on the track. Another simpler one is to create an operating system error at the end of the file, in this case, midway through Track 3. Input errors to the OSI Assembler do not cause the IO flags to be reset. We must be more subtle than just having an input error. If E<return> is sent to the Assembler, it exits to the operating system. In the operating system command mode, any line which is not a legal command creates a syntax error. For example, another E<return>, will do the job. The following lines in PASS two and PASS four prepare files for entry into the Assembler.

PASS Two

```
680 PRINT #7,"E"
690 PRINT #7,"E"
950 PRINT #7,"E"
960 PRINT #7,"E"
```

PASS Four

```
610 PRINT#7,"E"
620 PRINT#7,"E"
```

There is yet another problem. In their normal positions, the disk buffers occupy the same space as program memory. This problem can be solved by moving the buffers. Use the following steps to load first the file ZEQUATE, second EQUATE, and third OBJECT into the Assembler.

a) Load and run the Extended Monitor.

b) Suppose the file we wish to load starts on Track N and ends on Track M. Perform STEP 1)g) from "HOW TO USE IT." Be sure to use the values given below (or larger values where you have RAM).

ADDR($)	ADDR(D)	VALUE	
2326	8998	00	
2327	8999	50	
2328	9000	00	
2329	9001	5C	
232A	9002	N	
232B	9003	M	
232C	9004	N-1	
232D	9005	BF	
23AC	9132	00	ADDRESS MEMORY
23AD	9133	5C	BUFFERED INPUT

Note that 232C, 232D, 23AC, and 23AD have strange values. These values track the disk into loading the first track of your file into memory. Otherwise, you would have to

do that job separately.

c) If you have already loaded the first file, skip this step. Initialize the Assembler.

d) Re-enter the Assembler.

e) Get input by

!IO 2@ <return>.

f) Repeat a) - e) until all files are loaded.

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

REMARKS,REFINEMENTS,ADDITIONS

Resource will execute on 8K BASIC in a reasonable amount of time. The longest pass (PASS One) will run slightly less than an hour on a 1 MHz machine.

This package of programs is, in a sense, incomplete. Using the cross reference tables, one could give mnemonic names to all of the various labels and equates. These could be entered into a file. Then one extra pass over OBJECT could exchange address labels with mnemonic labels.

A big file line editor utility could be added to edit any one of the files created. If tables are known at disassembly time, they can be edited into SCRATCH. Incorrectly disassembled code could be corrected. These steps could be performed also on SCRATCH or OBJECT.

If table locations are known in advance, disassembly can be cleaned up considerably by replacing all table bytes with $FF (or any other value not equal to a 6502 opcode). Then, all tables will appear in the resource as a sequence of lines: 10000 .BYTE$FF. Using an editor, it then would be a simple task to replace each $FF by its correct value. I used this procedure on BASIC.

OS65D cannot be changed in this way since it will crash. But there is a simple solution. Move OS65D from addresses 2XXX to addresses, say 5XXX. When SCRATCH and SYMBOL are complete, go through them, changing the leading 5's back to 2's. A program to do this is simple to write. SYMBOL must be resorted and repetitions deleted. This way, I was able to use the trick with $FF in tables to obtain an accurate resource of the code in OS65D.

A simple, but useful, utility would be a commenter. Such a utility would allow the user to add comments to the end of each line of the resourced file or to insert lines into the file. I have used this technique to produce the various listings in this article.

Even though I am careful to fully document the machine software I write, I still find it useful to run the resource program over my machine programs. The cross reference files often reveal infelicities and logical inaccuracies.

I am still improving these programs. If you think of a nice enhancement, I'd be glad to hear about it.

Example 2.

```
1000  ;EQUATE FILE
1010 ;
1020  ;ZPAGE
1030 ;
1040  HHZZC1 = $C1
1050  HHZZC2 = $C2
1060  HHZZC5 = $C5
1070  HHZZC6 = $C6
1080  HHZZC7 = $C7
1090  HHZZC8 = $C8
1100 ;
1110 ;
1120  ;TWO BYTE
1130 ;
1140  HH17A5 = $17A5
1150  HH17E9 = $17E9
1160  HH1939 = $1939
1170  HH19BC = $19BC
1180  HH19D1 = $19D1
```

Example 3.

```
   10 18D8                  * = $18D8
 1000                     ;EQUATE FILE
 1010                    ;
 1020                     ;ZPAGE
 1030                    ;
 1040 00C1=               HHZZC1 = $C1
 1050 00C2=               HHZZC2 = $C2
 1060 00C5=               HHZZC5 = $C5
 1070 00C6=               HHZZC6 = $C6
 1080 00C7=               HHZZC7 = $C7
 1090 00C8=               HHZZC8 = $C8
 1100                    ;
 1110                    ;
 1120                     ;TWO BYTE
 1130                    ;
 1140 17A5=               HH17A5 = $17A5
 1150 17E9=               HH17E9 = $17E9
 1160 1939=               HH1939 = $1939
 1170 19BC=               HH19BC = $19BC
 1180 19D1=               HH19D1 = $19D1
10000 18D8 17                    .BYTE $17
10010 18D9 A916                  LDA #$16
10020 18DB 85C7                  STA  HHZZC7
10030 18DD 20ED18  HH18DD JSR  HH18ED
10040 18E0 20D119                JSR  HH19D1
10050 18E3 85C5                  STA  HHZZC5
10060 18E5 84C6                  STY  HHZZC6
10070 18E7 C6C7                  DEC  HHZZC7
10080 18E9 3037                  BMI  HH1922
10090 18EB D0F0                  BNE  HH18DD
10100 18ED 20BC19  HH18ED JSR  HH19BC
10110 18F0 A1C5                  LDA (HHZZC5,X)
10120 18F2 A8                    TAY
10130 18F3 4A                    LSR A
10140 18F4 900B                  BCC  HH1901
10150 18F6 4A                    LSR A
10160 18F7 B017                  BCS  HH1910
10170 18F9 C922                  CMP #$22
10180 18FB F013                  BEQ  HH1910
10190 18FD 2907                  AND #$07
10200 18FF 0980                  ORA #$80
10210 1901 4A      HH1901 LSR A
10220 1902 AA                    TAX
10230 1903 BDA517                LDA  HH17A5,X
10240 1906 B004                  BCS  HH190C
10250 1908 4A                    LSR A
10260 1909 4A                    LSR A
10270 190A 4A                    LSR A
10280 190B 4A                    LSR A
10290 190C 290F    HH190C AND #$0F
10300 190E D004                  BNE  HH1914
10310 1910 A080    HH1910 LDY #$80
10320 1912 A900                  LDA #$00
10330 1914 AA      HH1914 TAX
10340 1915 BDE917                LDA  HH17E9,X
10350 1918 85C1                  STA  HHZZC1
10360 191A 2903                  AND #$03
10370 191C 85C2                  STA  HHZZC2
10380 191E A5C8                  LDA  HHZZC8
10390 1920 D001                  BNE  HH1923
10400 1922 60      HH1922 RTS
10410 1923 98      HH1923 TYA
10420 1924 298F                  AND #$8F
10430 1926 AA                    TAX
10440 1927 98                    TYA
10450 1928 A003                  LDY #$03
10460 192A E08A                  CPX #$8A
10470 192C F00B                  BEQ  HH1939
10480 192E 4A                    LSR A
10490 192F 9008                  BCC  HH1939
10500 1931 4A                    LSR A
10510 1932 4A                    LSR A
```

Example 4.

```
.  CROSS REFERENCES
.
.  ZPAGE
.
C1    1918
C2    191C
C5    18E3 )18F0
C6    18E5
C7    18DB  18E7
C8    191E
.
.
.  JMP & JSR
.
18ED  S18DD
19BC  S18ED
19D1  S18E0
.
.
.  MEMORY
.
17A5  L1903
17E9  L1915
.
.
.  BRANCH
.
18DD  N18EB
1901  C18F4
190C  C1906
1910  C18F7 E18FB
1914  N190E
1922  N18E9
1923  N1920
1939  E192C C192F
```

RESOURCE 1

```
10 REM *** RESOURCE 1 ***
20 REM BY T.R.BERGER (COON RAPIDS,MN.) 11/80
30 PRINT"RESOURCE ** STEP 1 -BUILD SCRATCH AND SYMBOL FILES"
40 POKE 2972,13:POKE 2976,13
50 POKE 8998,00:POKE 8999,128
60 POKE 9000,00:POKE 9001,140
70 POKE 9006,00:POKE 9007,140
80 POKE 9008,00:POKE 9009,152
90 INPUT"SOURCE FILE NAME";SF$
100 INPUT"SCRATCH FILE NAME";JF$
110 INPUT"SYMBOL FILE NAME";SM$
120 PRINT:INPUT"NUMBER OF SYMBOLS";NS
130 REM DIMENSION SYMBOL AND POINTER ARRAYS.
140 DIM SS$(NS),V(NS)
150 REM ** MAIN PROGRAM **
160 DISK OPEN,6,SF$
170 DISK OPEN,7,JF$
180 REM ** LOOP BACK HERE **
190 INPUT #6,IN$
200 IF IN$="XIT"THEN 850
210 IF LEN(IN$)<15 THEN 190
220 REM ** ADJUST SOURCE,PICK UP SYMBOLS **
230 REM A1$=XXXX ADDRESS
240 REM A2$=OPCODE +
250 REM A3$=OPERAND (SYMBOL)
260 REM A4$=ADDRESS MODE
270 REM IN$=INPUT FROM OSI DISASSEMBLER
280 REM CU$=A1$+A2$+A3$+A4$
290 A3$="":A4$=""
```


Continued

```
300 REM ** GET ADDRESS **
310 A1$=LEFT$(IN$,4)
320 REM ** DO ERRORS **
330 IF MID$(IN$,13,1)="?"THEN A2$="  .BYTE $"+MID$(IN$,6,2):GOTO 790
340 REM ** DO REFORMATTING **
350 REM ** ELIMINATE END SPACES **
360 IN$=MID$(IN$,12):L=LEN(IN$)
370 IF MID$(INSS,L,1)=" " THEN L=L-1:GOTO 370
380 IN$=LEFT$(IN$,L)
390 REM ** DO IMPLIED AND ACCUMULATOR ADDRESSING **
400 IF L<7 THEN A2$=IN$:GOTO 790
410 REM ** DO IMMEDIATE ADDRESSING **
420 IF MID$(IN$,6,1)="#" THEN A2$=IN$:GOTO 790
430 REM ** ADJUST OPERAND POSITION **
440 IF MID$(IN$,6,1)="$" THEN K=7:A2$=LEFT$(IN$,5)+" HH":GOTO 470
450 K=8:A2$=LEFT$(IN$,6)+"HH"
460 REM ** Z PAGE CHECK **
470 M=K+2
480 REM ** DO Z PAGE OPERANDS **
490 IF M>L THEN A3$=RIGHT$(IN$,2):A2$=A2$+"ZZ":GOTO 790
500 REM ** Z PAGE, STRIP ADDRESS MODE **
510 IF MID$(IN$,M,1)>"/" THEN 540
520 A3$=MID$(IN$,K,2):A2$=A2$+"ZZ":A4$=MID$(IN$,M):GOTO 790
530 REM ** TWO BYTE OPERAND CHECK **
540 M=K+4
550 REM ** DO TWO BYTE OPERANDS **
560 IF M>L THEN A3$=RIGHT$(IN$,4):GOTO 620
570 REM ** TWO BYTE,STRIP ADDRESS MODE **
580 A3$=MID$(IN$,K,4):A4$=MID$(IN$,M)
590 REM ** PUT SYMBOLS IN TABLE **
600 REM ** SEARCH TABLE FOR SYMBOL **
610 REM THIS IS A BINARY SEARCH **
620 L=0:R=SN
630 REM ** SYMBOL NOT FOUND, INSERT IT **
640 IF L>R THEN 710
650 M=INT((L+R)/2)
660 REM ** SYMBOL IN TABLE SO QUIT **
670 IF A3$=SS$(V(M)) THEN 790
680 IF A3$>SS$(V(M)) THEN L=M+1:GOTO 640
690 R=M-1:GOTO 640
700 REM ** ADD SYMBOL **
710 SN=SN+1:SS$(SN)=A3$
720 REM ** POINT TO ITS PROPER POSITION IN ORDERING **
730 IF L=SN THEN 770
740 FOR I=SN-1 TO L STEP -1
750 V(I+1)=V(I)
760 NEXT I
770 V(L)=SN
780 REM ** GENERATE LINE FOR SCRATCH FILE **
790 CU$=A1$+A2$+A3$+A4$
800 PRINT#7,CU$
810 PRINT CU$
820 GOTO 190
830 REM ** LOOP BACK NOW **
840 REM ** CLOSE SCRATCH AND SOURCE FILES **
850 PRINT #7,IN$
860 DISK CLOSE,6
870 DISK CLOSE,7
880 REM ** END OF MAIN PROGRAM **
890 REM ** WRITE SYMBOL FILE **
900 DISK OPEN,7,SM$
910 REM ** WRITE SYMBOLS IN ORDER **
920 FOR I=0 TO SN
930 PRINT#7,SS$(V(I))
940 PRINTSS$(V(I))
950 NEXT I
960 PRINT#7,"XIT"
970 DISK CLOSE,7
980 REM ** OUTPUT DATA **
990 PRINT:PRINT
1000 PRINTTAB(9) SN" SYMBOLS USED"
1010 PRINT TAB(10) "SCRATCH FILE: "JF$
1020 PRINT TAB(10)"SYMBOL FILE: "SM$
1030 PRINT:PRINT TAB(10) "PASS 1 COMPLETED"
1040 PRINT:PRINT:END
```


★ ★ ★

BEGINNER'S CORNER

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

A BETTER PROGRAM

No program is perfect. People would like programs to run with absolutely no errors, but there is no way of being certain that this will always be so. The most we can hope for is that a program will run correctly with all valid data and recover from all likely erroneous input. One could argue that a program could be correct for one environment but not for another. Even so, it still pays to try and make a program as 'correct' as possible. A program that works 'most of the time', or only if the user has specialized knowledge, is just not good enough.

The 'Otaio Mailing List' (OML - see June 84 issue) is halfway to being a good program. It can be improved in at least three major areas. These are: screen layout, disk access, and user input. The listing shows some examples in the context of OS65D 3.3. Delete from the June listing lines 90 to 120 and lines 1240 and 1250. Type in the changes. When the changes have been made, list the program to the indirect file - (LIST then shift-K, <RETURN> and finish with shift-M). Next, run BEXEC*, select option 7, select one buffer, CTRL-X and PUT the program to disk. Use four tracks. I have found that if this procedure is not followed, programs with buffers will crash after changes have been made. No doubt BASIC pointers are not being reset correctly .... any ideas anyone?

CURSOR CONTROL

OSI DOS 3.3 has three great strengths. They are: 'print using', cursor addressing and 'print at'. The command 'print at' allows text to be printed to any part of the screen, including the center. Centering of text is a great boon - eyeballs no longer have to swivel left-and-up to see what's next! Also, upward screen-scrolling can be totally avoided - scrolling is BAD! The 'print at' syntax takes two forms: PRINT&(X,Y) and PRINT!(17,X,Y). The commands are interchangeable, at least on a video system.

Have a look at lines 1970-1980 - they print a message to the center of the screen. Row and column count is from (0,0), from the top left-hand corner of the screen, so the count in line 1970 is 17 horizontally and 13 down. Maximum values that can be used are 63 and 23.

Cursor addressing is a very powerful feature and is shown at its best in lines 920-1075. In line 920, the cursor is positioned at the start of line 16 - (remember, line 1 has cursor address 0). Line 950 prints the first column of the EDIT menu. Then, in lines 960-980, the 'print at' command really shines - the second column of the menu is printed next to the first! A considerable amount of the screen is now freed for showing several more Record fields, if required. But there is more.

Erasing lines of text from the screen is simplified. Have a look at line 3000. The 'PRINT!(18);' command takes the cursor top-left. Now 'PRINT!(15)' deletes to the end of the line and '!(11)' takes the cursor down one line. What could be simpler?

Editing of Record fields is also simplified. The cursor can be positioned directly on a particular piece of text. This is done in line 1050. The command 'PRINT!(18)' moves the cursor to top-left and then 'PRINT!(11)' moves the cursor down to the required field. Pressing <RETURN> moves the cursor down a line and the previous field is left unchanged. Changes to field contents are made by merely typing over the old text.

DISK CONTROL

OSI BASIC programming suffers a great deal when it comes to disk operations. Formatting of disks can be done from

```
40 GOSUB4010:B$="      OTAID MAILING LIST 12/84 by LZJ"

90 N$="------":L$=CHR$(8):L1$=L$+L$+L$+L$+L$+L$+L$+L$+L$
130 N=200:P=5:Z=0:ST=10:S=64:F$=CHR$(12):R$=" ":S$="$":H$="HELP":T=21
140 P1=P:DIMD$(N,P):C$=CHR$(13):C1$=CHR$(27)+CHR$(28)


190 PRINTC1$:PRINTTAB(11)"$ When in trouble type:-  HELP $":PRINT


220 PRINTTAB(T)"   MAIN MENU":PRINTTAB(T)"   ---------":PRINT
230 PRINTTAB(T)"1> Load a File":PRINTTAB(T)"2> Save a File"
235 PRINTTAB(T)"3> Pack Records":PRINTTAB(T)"4> Search the file"
240 PRINTTAB(T)"5> Edit Records":PRINTTAB(T)"6> Sort the file"
245 PRINTTAB(T)"7> Print or View all Records"
250 PRINTTAB(T)"8> Append to file":PRINTTAB(T)"9> List Erased Record #
255 PRINTTAB(T)"D> Directory":PRINTTAB(T)"R> Reset"
257 PRINTTAB(T)"X> END"
260 PRINT:PRINT"Choice ? ";:GOSUB310:IFY$="X"THEN1970
265 IFY$="R"THENPRINTC$"Reset <-- Sure ? ";:GOSUB310:IFY$="y"THENRUN
267 IFY$="D"THENPRINTC1$:GOSUB4030:D$:GOSUB4010:GOSUB310:GOTO190


340 PRINT"$ Seq. File Name ? "N$L1$;:INPUTY$:IFY$=H$THEN190
345 IFY$=""THENY$=N$


350 PRINT:PRINT"$ Loading from DISK now $":GOSUB4030:Y=Z+1


380 DISK CLOSE,6:GOSUB4010:N$=Y$:GOTO190


410 PRINT"$ Seq. File Name ? "N$L1$;:INPUTY$:IFY$=H$THEN190
415 IFY$=""THENY$=N$
420 PRINT:PRINT"$ Saving to DISK now $":GOSUB4030


450 DISK CLOSE,6:GOSUB4010:N$=Y$:GOTO190


570 PRINTC1$:Q$="?":F=0:K=0:FORC=1TO11:PRINT!(11);:NEXT


920 PRINTC1$!(17,0,15);
930 PRINTTAB(T+3)"EDIT MENU":PRINTTAB(T+3)"---------"
940 PRINT"Change:-";
950 FORC=1TOP:PRINTTAB(10)STR$(C)"> "N$(C):NEXTC:Y=17:X=35
960 PRINT&(28,Y)"or:-   6> Next Record"
970 PRINT&(X,Y+1)"7> Previous Record":PRINT&(X,Y+2)"8> Erase Record"
980 PRINT&(X,Y+3)"9> Random Select":PRINT&(X,Y+4)"X> EXIT":PRINT
990 :
1000 GOSUB1220:PRINT!(17,0,23);
1010 PRINT"Choice ? ";:GOSUB310:PRINTC$;:IFY$="X"THEN190
1020 IFY=0THEN880
1030 IFY>5THENR=-1:Y=Y-5:ONYGOTO1120,1140,1090,1180
1040 :
1050 PRINT!(18):FORC=1TOY:PRINT!(11);:NEXT
1055 INPUTY$:IFY$="X"THEN1000
1060 IFY$<>""THEND$(Q,Y)=Y$
1070 Y=Y+1:IFY=P+1THEN1000
1075 GOTO1055

1220 GOSUB3000:PRINT!(18);"RECORD "+STR$(Q)+" of"+STR$(Z):PRINT
1230 FORC=1TOP:Y$=D$(Q,C):PRINTTAB(2)Y$:NEXTC:RETURN


1400 I=INT(I/2):PRINT"Still Sorting":IFI<1THEN1460


1530 IFA=104THEN190
1540 PRINT:PRINT"Records PACKED ? ";:GOSUB310:IFA=110THEN190
1550 PRINT"Yes":SS=-1:E=ST+1:TB=40
1560 :
1570 PRINT:PRINT"Device # ? ";:GOSUB310:IFY=0THENY=2:PRINTY
1575 PRINT:INPUT"$ # of fields to print is ";P1:PRINT:IFP1>PTHEN1575
1577 IFP1=0THENP1=P
1580 V=Y:PRINT:PRINT"Ready ? ";:GOSUB310:IFA=104ORA=110THEN190
1590 PRINT:FORQ=1TOZ:FORX=1TOL:IFLEFT$(D$(Q,1),2)="ZZ"THEN1670


1680 V=2:P1=P:GOTO190


1740 PRINTC1$:PRINT"$ To return to main menu type:-  $":PRINT

1880 PRINT#V,D$(Q,1)TAB(32)Q:IFP1=1THENRETURN
1885 FORC=2TOP1:PRINT#V,D$(Q,C):NEXTC


1925 IFP1=1THENRETURN
1930 FORC=2TOP1:PRINT#V,D$(Q,C) TAB(TB)D$(Q+ST,C):NEXTC
1940 PRINT#V:RETURN


1970 PRINTC1$!(17,16,12)"To RESTART type:-  GOTO 190"
1980 PRINT!(17,16,14)"Bye !":POKE2073,173:GOSUB4030:END


2010 PRINT:PRINTTAB(18)"$ DISK error - try again! $"


3000 PRINT!(18);:FORC=1TOP+2:PRINT!(15)!(11);:NEXTC:RETURN
3010 :
4000 REM STOP DISK
4010 POKE49152,0:RETURN
4020 REM START DISK
4030 POKE49152,255:FORC=1TO1200:NEXTC:RETURN
```

within BASIC programs, with 'DISK!"INIT"', but not much else. It is true that disk-errors can now be prevented from halting a program with the TRAP command. But some important disk operations have to be done outside of programs - this is BAD! Luckily, HOOKS, (see Dec 83 & Jan, June 84 issues), by Rick Trethewey comes to the rescue. Line 255 provides an excellent example - the ability to read a disk directory from within a BASIC program. Another command worth installing is 'MAKE'. The ability to create disk-files from within the program would now be possible. In fact, the whole list of disk utilities could be provided in their own separate program block. Unfortunately, there is no elegant way to recover from printer being off-line. Some printer control though, could be provided in a separate printer block.

Notice how useful 'Reset' can be, line 255. No need to RUN the program again when switching to work with another file.

OSI software seems to assume that disk drives are to run continuously. Lines 4000-4030 can be used to stop and start all four drives simultaneously. (I only have one drive so I have not been able to test this). If there is more than one drive use 'DISK!"SE X" to select the drive required, where X is one of A,B,C or D. Alternatively, a particular drive can be selected by POKE-ing the right number into 49152.

USER INPUT

A friendly program will minimize the number of keystrokes that a user has to make. A good way of doing this is to provide default input. See this month's WAZZAT! column for a fuller explanation.

The story begins in line 90 with CHR$(8). Printing a CHR$(8) takes the cursor back one space. To input a 6-character file name by default, and allow for the INPUT question mark and space, 8 backspaces are required. They are stored in L1$. To see how all this works, try this short program:

```
5 POKE 2888,0: POKE 8722,0:
   PRINT!(28)
10 N$="------": L$=CHR$(8):
   L1$=L$+L$+L$+L$+L$+L$+L$+L$
```

continued on next page.

```
20 PRINT "File name ? "N$ L1$;
30 INPUT Y$: IF Y$="" THEN Y$=
   N$
40 IF LEN(Y$)<6 OR LEN(Y$)>6
   THEN RUN
50 PRINT "Some operation": N$=
   Y$
60 PRINT: GOTO 20
```

THE POKEs in line 5 allow null inputs in line 20, picked up in line 30.

This idea is applied very effectively in lines 340 and 410. Default input can also be used in DOS 3.2 (L$= CHR$(8)) and in OS65U.

PROGRAM TIPS

A program is always more useful if data can be produced selectively from a Record. With this in mind, the OML can be changed to offer a choice on the number of fields that are printed; see lines 1880 to 1940. But care must be taken when printing mailing labels - the number of fields specified for printing must be five if correct line spacing is to be maintained. Being able to print the first field only is particularly useful when only a list of names is required.

A program can be made a little shorter by storing CHR$ codes in variables. For example, C1$ in line 140. Replace all 'clear-screen' commands in the program with 'PRINTC1$'.

The slowest part of the program is the sort. A speed gain of about 12% (test result) can be gained by declaring first the variables used in the sort. Insert line 25, in which the following variables are all set to zero: C,Y,K,I,R,Q,Z,B,P. This is how it works. For any BASIC program the BASIC Interpreter program makes a variables table. Each time the Interpreter comes across a variable in the program, it searches the table for that variable's name. The further down the table a variable name is, the longer the search. With 50 or so entries in such a table, the micro-seconds add up significantly!

The OML program could be further improved by having the field names read not from within the program but off disk, from a sequential file. The program would then cater for many different types of files, each with its own number of distinctive Record field names.

★

6502 ASSEMBLY LANGUAGE PROGRAMMING CLASS

PART VIII

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

If you've kept up with the lessons to date, you should have a fundamental knowledge of the operations and instructions of the 6502. Now it's time to start putting this knowledge to work in a real program. For the non-OSI visitors to our class, the program we'll be writing is going to be OSI-specific, but the techniques are applicable no matter what machine you use. For the OSI people, we'll be using the OS-65D routines listed in the file LABELS.65D. If you haven't done so already, I suggest that you also read USING.65D.

When you set out to write a program in BASIC, it's easy to continue to add new lines and variables to your program because BASIC does all the nasty housekeeping chores of remembering where everything is stored in memory. You don't have that luxury in Assembly language. You have to keep track of where your program is going to reside in memory and how many bytes it will take up, as well as defining the memory addresses of each label (or variable) that isn't a part of your program. I point this out now because it is a necessary consideration during all stages of program development. However, it should also be noted that the assembler helps you do this in many ways, so don't be too concerned.

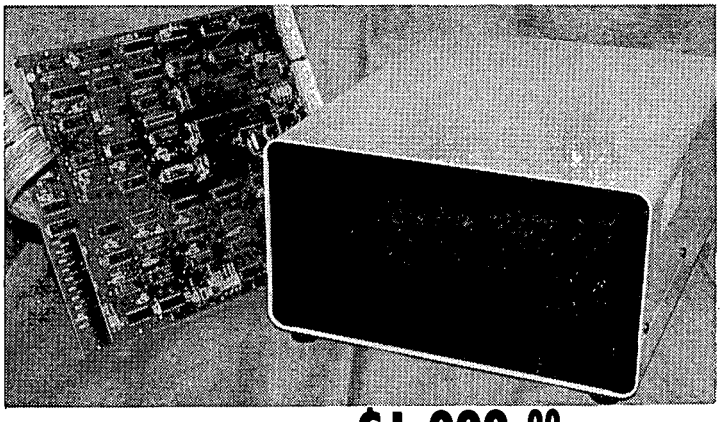
The first task in developing any program is to clearly define what you want the program to do. For this lesson, I've chosen to write a program that will print the disk directory for OS-65D. This program will work for all disk-based OSI's including the C1P-MF.

After defining the purpose of the program, the next step is to break the job down into the individual tasks that make up the whole job. In order to do the directory printer, we first have to know a bit about how OS-65D maintains the directory. The directory is simply a list of the files contained on a disk including the file name, the first track of the file, and the last track of the file. The file name is 6 characters long, the first of which must be an alphabetic character from "A" to "Z". The track numbers are stored in Binary Coded Decimal form (refer to lesson 7 for information on BCD). The directory entries are stored on the disk on a track specifically designated for this purpose. As implied above, each entry in the directory requires 8 characters or bytes. On mini-floppy systems, the directory track is number 12, on 8" systems, it's track number 8. The directory is stored in the first two sectors of the directory track, with each of these sectors being one page (256 bytes) long. At 8 bytes per entry, this gives the directory a capacity of 64 entries. OS-65D reserves a page of memory at $2E79 to hold one sector of the directory for various purposes and

SUPER
HARD DISK Subsystem!

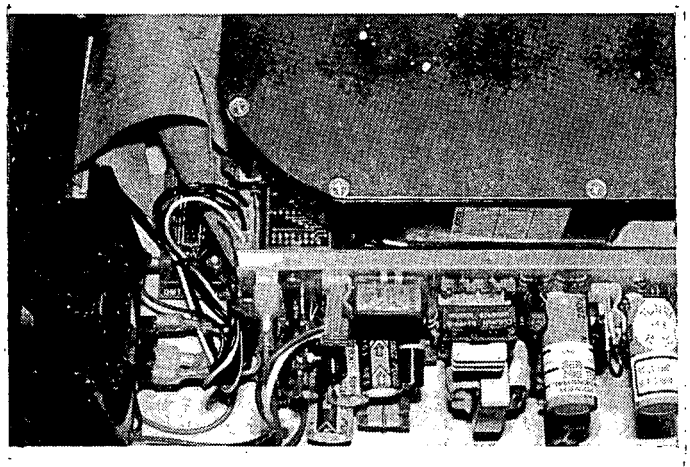
TURNS ANY FLOPPY BASED COMPUTER INTO HARD DISK BASED, INSTANTLY.

- PLUGS INTO ANY OSI TYPE BUS
- ONE RIBBON CABLE CONNECTS TO DRIVE
- COMPLETELY SELF CONTAINED
- 32 BIT ERROR DETECTION AND CORRECTION
- HAS REAL TIME CLOCK *CALENDAR W/BATTERY ON SCSI ADAPTER BOARD
- CAN BOOT DIRECTLY FROM OSI 505/510 CPUs OR DENVER BOARDS W/SCSI PROM
- IDEAL BACK-UP FOR ALL OSI HARD DISK COMPUTERS



FROM $1,999.00

THE SPACE-COM SUPER SUBSYSTEM USES 5¼" INDUSTRY STANDARD HARD DISK DRIVES INTERFACED TO THE OSI BUS BY THE DS-1 SCSI HOST ADAPTER BOARD AT THE COMPUTER END AND THE STATE OF THE ART OMTI 5000 SERIES INTELLIGENT DISK/TAPE CONTROLLERS AT THE DISK END. THE DENVER DS-1 BOARD NOT ONLY PROVIDES THE BUS TRANSLATION, BUT GIVES REAL TIME OF DAY, DAY/WEEK, AM/PM, AND DAY/MO. WITH ON BOARD BATTERY, DATE AND TIME ARE MAINTAINED W/O POWER.



THE CHASSIS IS BEAUTIFULLY ENGINEERED WITH LIGHTED ON/OFF SWITCH, STANDARD A/C CORD, AND INSULATED SPADE TERMINALS FOR EASY SERVICE. A CORCOM EMI FILTER IS INCORPORATED IN THE A/C JACK, AND POWER IS PROVIDED BY AN EXTREMELY EFFICIENT SWITCHING POWER SUPPLY. THE CASE IS ALSO AVAILABLE IN DUAL, SIDE BY SIDE CONFIGURATION AND LOOKS LIKE AN IBM PC BOX. IT INCORPORATES A LARGER POWER SUPPLY AND CAN SUPPORT 2 WINCHESTER DRIVES, OR 1 DRIVE AND TAPE, OR 2 5" FLOPPIES IN PLACE OF ONE OF THE ABOVE.

DRIVES CAN BE ACCESSED FROM ANY SINGLE OR MULTI-USER OSI SYSTEM BY RUNNING AN OVERLAY PROGRAM ON THAT PARTITION, OR CAN BE BOOTED DIRECTLY BY REPLACING CURRENT ROM/PROM WITH OUR SCI 500 PROM, AVAILABLE FOR $49.00 EXTRA.

SINGLE 20 M/B DRIVE (15.7 FORMATTED) SGL CASE	$1,999.00
SINGLE 26 M/B DRIVE (21 FORMATTED) SGL CASE	$2,199.00
DUAL 20 M/B DRIVES (M/B FORMATTED) DUAL CASE	$2,999.00
DUAL 26 M/B DRIVES (42 FORMATTED) DUAL CASE	$3,299.00
SUPER FAST 85 M/B DRIVE (70 FORMATTED) SGL CASE	$3,999.00
DUAL 85 M/B DRIVES (140 M/B FORMATTED) DUAL CASE	$6,699.00

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

we will use this area for our program here as well. Of course, most diskettes will have far fewer than 64 entries and OS-65D uses a pound sign ("#") to mark an unused entry position in a sector of the directory. Now let's get to work.

We'll begin as we should by breaking our job down into steps. The steps we outline here are only to get us started. During the writing of our program, it is likely that we will need to change the order in which the steps are executed and/or add more steps as we discover that our original list overlooked some aspect of the task at hand.

(1) Read the first sector of the directory track into the directory buffer at $2E79.

(2) Define a 16-bit pointer to the directory buffer. This pointer will require two bytes in memory and the address of the start of the directory buffer must be stored in this pointer. We'll call this pointer "DIRPTR" in our program.

(3) Look at the byte pointed to by DIRPTR. If it's a pound sign ("#"), jump to step 9.

(4) Since it's not a "#", display the first 6 characters exactly as we find them.

(5) Get the 7th character. This is the first track of the file in BCD.

(6) Convert this BCD value into two characters, the first for the MSB and the second for the LSB of the track number, and display them.

(7) Get the 8th character.

(8) Repeat step 6 for this, the ending track number.

(9) Add one entry length (8 bytes) to DIRPTR.

(10) Are we done looking at this sector? Or, in other words, does DIRPTR = $2E79 + $100 ? If not, repeat the process beginning at step number 3.

(11) Read the second sector of the directory track into the buffer.

(12) Reset DIRPTR to $2E79.

(13) Repeat steps 3 through 10, quitting when DIRPTR points to $2E79 + $100 (or $2F79).

True confessions time. I seldom write out these lengthy descriptions of each step in a program I'm going to write unless it's a really big job. What I really do is write a skeleton program using significant subroutine names to define the individual steps. This helps me to discover the possibilities of writing loops into my program to handle repetitive chores, as well as pointing up the need for memory locations to store values that my program will need when it is running. For our directory printer, I'd probably write something like this:

```
P0    JSR RDDIR      ;READ DIRECTORY SECTOR
      LDA #$79       ;INIZ
      STA DIRPTR     ;SET DIRPTR LSB
      LDA #$2E       ;INIZ
      STA DIRPTR+1   ;SET DIRPTR MSB
P1    LDY #$00       ;INIZ INDEX TO ENTRY
      LDA (DIRPTR),Y ; FETCH 1ST CHARACTER
      CMP #$23       ; IS IT A "#"?
      BEQ NEXT       ; YES! ==> NEXT
P2    LDA (DIRPTR),Y ; FETCH FILENAME CHARACTER
      JSR OUTCH      ; PRINT IT
      INY            ; BUMP INDEX ONE
      CPY #$06       ; PRINTED 6 CHARACTERS?
      BNE P2         ; NO? ==> P2
      LDA (DIRPTR),Y ; YES! FETCH 1ST TRACK #
      JSR PRINTT     ; PRINT IT
      INY            ; BUMP INDEX ONE MORE
      LDA (DIRPTR),Y ; FETCH ENDING TRACK #
      JSR PRINTT     ; PRINT IT TOO
NEXT  LDA DIRPTR     ; FETCH DIRPTR LSB
      CLC
      ADC #$08       ; ADD 8 TO IT
      STA DIRPTR     ; SAVE IT ACK IN DIRPTR
      LDA DIRPTR+1   ; FETCH MSB
      ADC #$00       ; ADD IN ANY CARRY'S
      STA DIRPTR+1   ; SAVE IT BACK TOO
      LDA DIRPTR     ; FETCH NEW LSB
      CMP #$79       ; IS IT $79 AGAIN?
      BNE P1         ; NO! ==> P1
      LDA SECT       ; YES! GET SECTOR #
      CMP #$02       ; IS IT 2?
      BEQ DONE       ; YES! QUIT! ==> DONE
      INC SECT       ; NO, BUMP IT ONE
      JMP P0         ; AND JUMP TO LOOP TOP!
DONE  RTS            ; EXIT
```

Now certainly a lot of this is intuitive. For example, the code as it stands assumes that RDDIR and PRINTT are already written. But again, this is only a first pass outline, designed not to be the final program but instead to just help you to see what exactly needs to be done, and especially what needs to be done that you didn't think of the first time through the writing process. Just looking at what I wrote above, I see two big problems. The first is that "SECT" needs to be initialized to "1" before step "P0". Secondly, as the program stands, it would print each character one right after the other without any spaces and would be unreadable, so I know I need to add code to print spaces between the end of the file name and the first track number as well as between the first track number and the ending track number. Finally, after the ending track number is printed, I'll insert code to do a carriage return and line feed so that each file will be printed on a separate line.

Okay, there are two major routines in my skeleton program that I have to write. The first is RDDIR. Fortunately, we've already done most of the coding for this in the file USING.65D. The code goes something like this:

```
RDDIR  LDA #$79      ; LOAD LSB OF DIR. BUFFER
       STA ADRLX     ; GIVE LSB TO 65D
       LDA #$2E      ; LOAD MSB OF DIR. BUFFER
       STA ADRHX     ; GIVE MSB TO 65D
       LDA #$08      ; MAKE THIS #$0C FOR MINI'S
       STA TRAKX     ; GIVE TRACK # TO 65D
       JSR SEEKX     ; MOVE DRIVE HEAD TO TRACK
       JSR LOAD      ; LOAD DRIVE HEAD
       JSR CALLX     ; READ DISK SECTOR
       JSR UNLOAD    ; UNLOAD HEAD
       RTS           ; QUIT
```

Next we have to write the routine to print the track numbers. The job here is twofold. First we need to break the number down into two separate digits and then print it out. This is the only use where the Binary Coded Decimal storage of track numbers pays off, because the values are automatically in decimal and we don't have to worry about converting the numbers from hexadecimal. That leaves us with a pretty simple task;

```
PRINTT  PHA       ; SAVE ORIGINAL BYTE ON STACK
        LSR A     ; SHIFT BYTE RIGHT ONE BIT
        LSR A     ; AGAIN
        LSR A     ; AGAIN
        LSR A     ; A TOTAL OF 4
        CLC       ; SET UP FOR ADD
        ADC #$30  ; ADD THE ASCII VALUE OF "0"
        JSR OUTCH ; PRINT IT
        PLA       ; RETRIEVE ORIGINAL FROM STACK
        AND #$0F  ; MASK TO LOW NYBBLE
        CLC       ; SET UP FOR ADD
        ADC #$30  ; ADD "0" AGAIN
        JSR OUTCH ; PRINT IT
        RTS       ; AND QUIT
```

All right, now we can put this all together to form our final program:

```
10         *=$5D00          70 TRAKX  =$2662
20 DIRPTR  =$E1             80 SEEKX  =$26A6
30 OUTCH   =$2343           90 LOAD   =$2754
40 SECT    =$265E          100 UNLOAD =$2761
50 ADRLX   =$2660          110 CALLX  =$295D
60 ADRHX   =$2661          120;
```

continued on next page.

```
130         LDA #$01        ; INIZ
140         STA SECT        ; SET 65D TO SECTOR 1
150 P0      JSR RDDIR       ; READ DIRECTORY SECTOR
160         LDA #$79        ; INIZ
170         STA DIRPTR      ; SET DIRPTR LSB
180         LDA #$2E        ; INIZ
190         STA DIRPTR+1    ; SET DIRPTR MSB
200 P1      LDY #$00        ; INIZ INDEX TO ENTRY
210         LDA (DIRPTR),Y  ; FETCH 1ST CHARACTER
220         CMP #$23        ; IS IT A "#"?
230         BEQ NEXT        ; YES! ==> NEXT
240 P2      LDA (DIRPTR),Y  ; FETCH FILENAME CHARACTER
250         JSR OUTCH       ; PRINT IT
260         INY             ; BUMP INDEX ONE
270         CPY #$06        ; PRINTED 6 CHARACTERS?
280         BNE P2          ; NO? ==> P2
290         LDA #$20        ; LOAD AN ASCII <SPACE>
300         JSR OUTCH       ; PRINT IT
310         LDA (DIRPTR),Y  ; YES! FETCH 1ST TRACK #
320         JSR PRINTT      ; PRINT IT
330         LDA #$2D        ; LOAD A "-"
340         JSR OUTCH       ; PRINT IT
350         INY             ; BUMP INDEX ONE MORE
360         LDA (DIRPTR),Y  ; FETCH ENDING TRACK #
370         JSR PRINTT      ; PRINT IT TOO
380 NEXT    LDA DIRPTR      ; FETCH DIRPTR LSB
390         CLC             ;
400         ADC #$08        ; ADD 8 TO IT
410         STA DIRPTR      ; SAVE IT BACK IN DIRPTR
420         LDA DIRPTR+1    ; FETCH MSB
430         ADC #$00        ; ADD IN ANY CARRY'S
440         STA DIRPTR+1    ; SAVE IT BACK TOO
450         LDA DIRPTR      ; FETCH NEW LSB
460         CMP #$79        ; IS IT $79 AGAIN?
470         BNE P1          ; NO! ==> P1
480         LDA SECT        ; YES! GET SECTOR #
490         CMP #$02        ; IS IT 2?
500         BEQ DONE        ; YES! QUIT! ==> DONE
510         INC SECT        ; NO, BUMP IT ONE
520         JMP P0          ; AND JUMP TO LOOP TOP!
530 DONE    RTS             ; EXIT
540 ;
550 RDDIR   LDA #$79        ; LOAD LSB OF DIR. BUFFER
560         STA ADRLX       ; GIVE LSB TO 65D
570         LDA #$2E        ; LOAD MSB OF DIR. BUFFER
580         STA ADRHX       ; GIVE MSB TO 65D
590         LDA #$08        ; MAKE THIS #$0C FOR MINI'S
600         STA TRAKX       ; GIVE TRACK # TO 65D
610         JSR SEEKX       ; MOVE DRIVE HEAD TO TRACK
620         JSR LOAD        ; LOAD DRIVE HEAD
630         JSR CALLX       ; READ DISK SECTOR
640         JSR UNLOAD      ; UNLOAD HEAD
650         RTS             ; QUIT
660 ;
670 PRINTT  PHA             ; SAVE ORIGINAL BYTE ON STACK
680         LSR A           ; SHIFT BYTE RIGHT ONE BIT
690         LSR A           ; AGAIN
700         LSR A           ; AGAIN
710         LSR A           ; A TOTAL OF 4
720         CLC             ; SET UP FOR ADD
730         ADC #$30        ; ADD THE ASCII VALUE OF "0"
740         JSR OUTCH       ; PRINT IT
750         PLA             ; RETRIEVE ORIGINAL FROM STACK
760         AND #$0F        ; MASK TO LOW NYBBLE
770         CLC             ; SET UP FOR ADD
780         ADC #$30        ; ADD "0" AGAIN
790         JSR OUTCH       ; PRINT IT
800         RTS             ; AND QUIT
810;
820         .END
```

To run this program, create a file to hold the program. Three tracks should do it. Next, get to the OS-65D "A*" prompt and enter "ASM" to run the OSI Assembler/Editor. Type in the program as it exists in the listing above. Save the program in the file you created with the command;

```
!PU FILNAM
```

where "FILNAM" is the name of the file. Next, enter the command;

```
H5CFF
```

This tells the assembler not to use any memory that might conflict with where our program will reside in memory. Next, enter the command;

```
A3
```

This tells the Assembler to go ahead and assemble the programming memory. Now all that's left is to execute the program. To do this, enter;

```
!GO 5D00
```

This should display the directory of the currently selected disk drive.

In looking at this program, you'll note that I did not use the OS-65D routine called "SWAP" at $2CF7, to set up the DOS context of OS-65D. The reason this was unnecessary is that the use of the "!" command in the assembler automatically puts the system in the DOS context. The page zero locations $E1 and $E2, labeled DIRPTR in the program, are utility locations in OS-65D that are used to hold other pointers. Since these pointers are only temporary, we don't have to worry about destroying anything 65D needs by using them in our own programs. My admonitions about using page zero locations in USING.65D are still valid, however.

★

LABELS AND ADDRESSES

"LABELS.65D"

(See PEEK(65) Dec. 84, page 7.)

OS-65D EXTERNALS

TEMP	=$E0	SELECT	=$29C6
PNTL	=$E1	ERROR	=$2A4B
PNTH	=$E2	OS65D3	=$2A51
MAXVAL	=$E5	ERRSU	=$2A7D
TMP2	=$FB	CSI	=$2A84
TMP1	=$FC	ERR7	=$2AC0
TMP	=$FD	DEFAUL	=$2AC5
ADRL	=$FE	ASMTK	=$2ADF
ADRH	=$FF	BASR	=$2AE6
MAXMEM	=$2300	LOADER	=$2BA7
INFLAG	=$2321	PUTR	=$2BDD
OUFLAG	=$2322	SRCSIZ	=$2BE9
INCH	=$2340	REASM	=$2C04
OUTCH	=$2343	REBAS	=$2C0B
MEMIN	=$2389	REEM	=$2C12
DISC	=$265C	TXTLNE	=$2C9B
SECT	=$265E	TIN0	=$2CEC
PAGES	=$265F	SWAP	=$2CF7
ADRLX	=$2660	CRLF	=$2D6A
ADRHX	=$2661	STROUT	=$2D73
TRAKX	=$2662	PRBYTE	=$2D92
HOME0	=$2663	FNDNUM	=$2DA6
SEEKX	=$26A6	DIRTRK	=$2DC4
LOAD	=$2754	TXTBUF	=$2E1E
UNLOAD	=$2761	DIRBUF	=$2E79
INTRC	=$277D	CRSCLR	=$32E3
SAVEX	=$27D7	WIDTH	=$32EA
FIND	=$28C4	HZLPRT	=$33C0
CALLX	=$295D	KEYIN	=$3590
CALL	=$2967	CASECK	=$3A5F
DUMRED	=$2998	SRCSTR	=$3A79

★

WAZZAT CORNER!

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

CURSOR-BACK

For a long time the only useful cursor control available to the OSI disk user was 'cursor-back' - with CHR$(8). The command is surprisingly powerful when writing programs designed to provide default input. Examine the three programs labeled INPUT ONE TWO and THREE.

The first two programs illustrate the same idea - using CHR$(8) to position the cursor inside the line instead of at its end. In the first program a halting gey-key routine is

used and the cursor is positioned over the 'Y' in 'Yes'. In the second program the technique is illustrated with 'INPUT', with the cursor positioned over 'N' in 'NAME'. Notice that with INPUT the required number of CHR$(8)s is always,'length of default message plus 2'. Why 'plus 2' ? Because a question mark and a space are printed when INPUT is used. The idea should work with DOS 3.2 and OS 65U.

CURSOR-UP

With OS 65D 3.3 a whole new world of cursor addressing opened. Program three does exactly the same as programs one and two but this time the cursor is tabbed across, then taken up one line with '!(12)'. The cursor will be positioned over the 'N' in 'NAME' and is preceded by the INPUT space and question mark. The POKEs in line 25 are required to permit null input.

In program four, line 40 illustrates how 'A to z' key presses can be selected from input from the keyboard. 'RUN' it and see what happens!

BASIC forces programs to work with 14 character fields in a width of 112+14. The fifth program is a good example where this mandatory figure of '14' is a nuisance. Aha! Why not change '14' to something else? Change line 20 to

20 POKE 2720,12.

Other useful POKEs to know are POKE 24,X (normally 112) for line-width for fields, and POKE 23,X (normally 132) for terminal width.

EXTENDED INPUT

All these ideas can be thrown together to provide Extended Input under DOS 3.3 and 3.2 - see the final program. DOS 3.3 arithmetic works to 9 significant figures, so I=9 in line 30.

This is how it works. Somewhere in BASIC or DOS is the number 80 against which the number of characters input from the keyboard is counted. Once 80 characters have been counted no more can be input from the keyboard.

It would be good to know where '80' (79?) is stored. The program would then be trivial - merely POKE the appropriate value. Can anyone help?

Anyway, M$ in line 60 has length 21. Print 47 spaces in line 70, giving a total of 68 characters printed. Now backspace 48 characters leaving 11 .....68+11=79. (Yes, eleven - 0 is now the first character count; 0 to 79=80!.) Subtract two, one for the space and one for the question mark as printed by 'INPUT'. And that leaves 9 - as in 'I'. Wazzat!

Input from the keyboard can be limited to only 'I' characters. Complicated and incredible, but it works!

Finally, change line fifty to:

50 POKE 2797,ASC("$")

and spot the difference in output!

★

```
10 REM INPUT ONE by LZJ
20 :
30 PRINT!(28): L$=CHR$(8): DV=2
40 PRINT"Send to Printer ? Yes" L$ L$ L$;: GOSUB 500
50 PRINT: IF Y$="y" THEN 100
60 PRINT"Output to screen"
70 GOTO 300
80 :
100 PRINT #DV, "Output to printer"
300 END
400 :
500 DISK!"go 2336": Y$=CHR$(PEEK(9059)): PRINT: RETURN
```

```
10 REM INPUT TWO by LZJ
20 :
25 POKE 2888,0: POKE 8722,0
30 PRINT!(28): L$=CHR$(8)
40 PRINT "File name   NAME" L$ L$ L$ L$ L$ L$;
50 INPUT Y$: IF Y$="" THEN Y$="NAME"
80 :
100 PRINT: PRINT"Loading file " Y$
300 END
400 :
```

```
10 REM INPUT THREE by LZJ
20 :
25 POKE 2888,0: POKE 8722,0
30 PRINT!(28): L$=CHR$(8)
40 PRINT "File name   NAME"
50 PRINT TAB(10)!(12);:INPUT Y$: IF Y$="" THEN Y$="NAME"
80 :
100 PRINT: PRINT"Loading file " Y$
300 END
```

```
10 REM INPUT FOUR by LZJ
20 :
30 PRINT!(28): INPUT "press a key ('A to z'), and <RETURN> "; Y$
40 IF (ASC(Y$) OR 32) >96 THEN 100
50 PRINT: PRINT"Rhubarb rhubarb"
60 END
70 :
100 PRINT!(28): PRINT"Y$ is " Y$: PRINT
110 PRINT"and CHR$((ASC(Y$) OR 32)) is " CHR$(ASC(Y$)OR32)
```

```
10 REM Formatting by LZJ
20 :
30 REM
40 PRINT"X", "X$X", "X$X$X", "X$X$X$X", "sq root"
60 FOR c=1 TO 10: X=C
70 :  PRINT X, X$X, X$X$X, X$X$X$X, SQR(X)
80 NEXT C
```

```
10 REM Extended Input Idea by LZJ
20 :
30 PRINT!(28): L$=CHR$(8): I=9
50 REM
55 :
60 M$="Type in the amount   ":L=LEN(M$): X=77-L-I: Y=X+1
70 PRINT M$;: FOR C=1 TO X: PRINT " ";: NEXT
80 FOR C=1 TO Y: PRINT L$;: NEXT
85 :
90 INPUT A$: PRINT: PRINT "Amount = $";VAL(A$)
100 REM
```

TAX AIDS II
A REVIEW

By: Richard L. Trethewey
8 Duran Court
Pacifica, CA 94044

TAX AIDS II written by Robert Baldassano (RSB Enterprises) is a package designed to help you prepare your taxes for 1984 and beyond. The program allows you to make entries for the various tax forms and have the results printed for you.

Not a program that merely lets you "fill in the blanks", TAX AIDS II allows you to play with the numbers before you talk to Uncle Sam. The versatility of this package is in the way it allows you to create your own tax tables and forms for future needs and to have those forms incorporated

automatically in the final result. It's like having your own spreadsheet program especially designed for your taxes, and yet accounting for future needs. In addition, the program helps you compute your depreciation allowances, using various methods including the ACRS.

The documentation is a bit skimpy, although that judgement is likely to be due to my inexperience with the more esoteric aspects of tax forms. Still, I have to say that I would have felt more comfortable with some extra handholding. The instructions can be displayed on your screen or routed to a printer. Since they are incoporated into a BASIC program for output, this probably largely accounts for the brevity. You will want to have a hardcopy of the instructions printed out before using the program.

Even with the brief instructions, the programs are easy to use. Much of the time you will be simply copying numbers from your records into the program. The tax preparation utility supports form 1040, with schedules A, B, C, and G, as well as form 6251, and you will be entering information exactly as required by those forms.

TAX AIDS II will greatly speed up your preparation of your taxes, as well as helping you plan your finances for the coming years. The author also makes an optional utility program available that will take the information generated and print them directly onto the forms for you. The "PRINT ON FORMS" option is only $10.00.

★

WHAT IS THE OSI-SIG ON COMPUSERVE?

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

The Ohio Scientific, Inc. Special Interest Group (OSI SIG) is a service of CompuServe that is dedicated to OSI owners. Much like most local BBS's, OSI SIG provides a sophisticated messaging system and a Data Library for users to exchange programs and articles. There is currently (2500*640) K of files available to OSI SIG members. In addition, OSI SIG provides an on-line Conference area that lets users talk (or type) to each other in real time. We hold a conference each week on Thursday evenings at 10 PM Eastern time.

A subscription to CompuServe is available in several ways. Radio Shack used to (and may still) sell a subscription for $19.95 that included one hour of Standard connect time and a terminal program for TRS-80s. Most computer stores sell a generic package for $39.95 that includes five hours of Standard connect time and a CompuServe User Guide, which is by far the best deal. CompuServe is a local phone call in most cities and Standard connect time will cost you between $6.25 and $8.25, depending on whether or not you have to sign on through a supplemental network such as TYMNET or TELENET. Prime time and 1200 baud access runs between $12.50 and $15.00.

Don't forget that in addition to access to OSI SIG, a CompuServe subscription will let you use all of their other services as well - including airline schedules and reservations, stock quotes and analysis, weather forecasts, news, other Special Interest Groups, multi-user games, Scott Adams' Adventures, discount purchases from companies like Sears, florists, 47th Street Photo, and much more.

For information on getting a subscription, call 800-848-8199.

★

TIPS FROM OSI

Text Processor TP-2 for
Series 200 and Challenger

A. Configuration of the TP-2 for terminals and printers not included in the standard software.

TP-2 is a compiled package. It requires very efficient input/output routines in order to function satisfactorily on time-shared systems. Field addition of new terminal or printer drivers is, therefore, not possible. Isotron has a policy of providing free configuration service under conditions that the customer lends us his peripheral and manual for it, pays for its transport to and from our facility in Fairfield and buys at least 3 copies of the reconfigured TP-2.

Terminals must have following functions to be usable: erase all, erase to end of screen, erase to end of line, insert line, delete line, and at least one video attribute <u>not</u> occupying a space on the screen (ADDS will not work).

Following terminals are currently supported: Hazeltine 1420, Televideo 920/925, ACT-4, OSI T-6310, MicroTerm 4420 in an enhanced ADM-3A mode and Tandberg 2215.

Please note: Hazeltine 1500 has a different keyboard layout (no arrow keys, forcing the user to press 4 keys to move the cursor). This, together with some other minor problems, makes it unsuitable for screen-oriented word processing.

B. A bug has been discovered in the NEC printer driver included in the initial release. It has been corrected, and all copies delivered after the end of September are okay. Please note that there is no reason to request an update if you do not use a NEC printer. Even if you do, you may very well get all performance you need by using the BACKSPACE driver instead. Otherwise, updated copies may be obtained from the service dept. in Aurora. Simply return your <u>original TP-2 diskette</u> for recopying. The charge for this, as with all other software recopying, is $25 per diskette. Be sure to contact customer service, Aurora, for your return authorization number and to remember that the orignal copy must be returned.

KeyWord Word Processor
for Series 300

In spite of the relatively high quality of the KeyWord manual and extensive on-line help utilities, a few points are frequently misunderstood and are explained below:

A. Automatic pagination does not seem to work.

Automatic paging works fine, but requires that you give a "paging" command. Try pressing SET P (not SET PAGE). See also page 8-4 of the KeyWord manual.

B. Underlining does not work on SPACE and Hyphens.

Use HARD SPACE (SET SPACE) and HARD HYPHEN (SET -1) instead of the usual ones.

C. Video attributes don't always show as expected on the screen.

This happens with screens

which do not allow for overlapping of several attribute commands. That means that the second attribute extinguishes the first one and can be observed on, among others, the OSI T-6310 terminal.

Initial Print/Spool Allocation in TurboDOS

Changing QUEPRT in SLAVE.PAR does not change initial queue assignment. The queue assignment is instead set automatically to whatever has been predetermined for this node in the network.

Solution: GEN a new SLAVE.SYS using ;S option. Check address of QUEPRT. Enter MONITOR and load in SLAVE.SYS. Search using function "W" for following string (hex):

32,LB,HB,C9,06 (LB and HB are QUEPRT address).

If more than one location is given, select the lower one (should be somewhere between 0300 and 0400). Replace found 32 with a C9. Save back SLAVE.SYS and copy it to OSSLAVEX.SYS.

This procedure enables you to tailor your initial queue assignment for individual users connected to the system.

★

VARIABLE LISTER FOR OSI ROM BASIC AND 65D 3.2 BASIC

By: E. D. Morris
3200 Washington
Midland, MI 48640

When debugging your own programs or trying to understand someone else's program, it is often useful to know all the places that a particular variable is used. The following program will search a BASIC program and pick out all the variables. The variables are then sorted alphabetically and printed with the line number where each appears. To use the program, first load your program, then the variable lister. Then "RUN 60000". As each variable is found, it will be printed on the screen together with the line number. On the second pass the variables are sorted. If you do not have a line printer, you can stop the listing with control "C" and "CONT" to resume the listing.

The program works by picking out all ASCII letters within the BASIC line and assuming these are variables. Remember BASIC commands such as "PRINT" "GO TO" "NEXT" are stored as a single byte token and not ASCII characters. The variable lister ignores letters in REM and DATA statements, and everything between quotes in PRINT statements. All other letters are variable names.

The BASIC text starts at location 769 (dec). The first two bytes of every line are pointers to the next line. The next two bytes are the current line number in hex. Following is the BASIC text with a hex 00 at the end of each line. The variable lister makes use of this line structure to step through each line of BASIC. Line 60470 of the sort routine prints T$(L) and then replaces it with the "up arrow" character so it will not be printed again.

Line 60060 dimensions T$ for the maximum number of variables you may have. This may need to be adjusted up or down depending on the size of your program and the amount of free memory you have. The variable lister does use subscripted strings. If you are searching a large program and have only 8K of memory, you may trigger the garbage collector bug. If this happens, you can either use this as an excuse to buy more memory or search your program in several sections.

```
60000 PRINT"VARIABLE LISTER"
60010 PRINT"EXTRACTS AND LISTS BASIC VARIABLES":PRINT
60020 PRINT"AFTER THE FIRST PASS THRU"
60030 PRINT"PROGRAM-SORTS-THEN LISTS"
60040 PRINT"ALPHABETICALLY"
60050 LL=59999:REM HIGHEST LINE TO EXAMINE
60060 DIMT$(100)
60070 REM ***FIND FIRST LINE***
60080 NL=769:P=769:REM FOR 65D3.2   NL=12671  P=12671
60090 CL=NL:F=0
60100 NL=PEEK(P)+256*PEEK(P+1)
60110 IF NL=0 THEN60420
60120 P=P+2
60130 LN=PEEK(P)+256*PEEK(P+1)
60140 IF LN>LL THEN60420
60150 P=P+1:CH=PEEK(P):REM GET NEXT ALPHA
60160 IFP=NLTHEN60090
60170 IF CH=34THENGOSUB60380:IFFTHEN60090
60180 IFCH=142THENP=NL:GOTO60090: SKIP REM
60190 IFCH=131THENP=NL:GOTO60090: SKIP DATA
60200 IFCH<65 THEN60150
60210 IFCH>90 THEN60150
60220 LA$=CHR$(CH):GOTO60240
60230 LA$=LA$+CHR$(CH)
60240 P=P+1:CH=PEEK(P):REM GET NEXT CHAR
60250 IFP=NLTHEN60090
60260 IFCH>64 THENIF CH<91 THEN60230
60270 IFCH>47 THENIF CH<58THEN60230
60280 IF CH=36 THEN60230
60290 REM STORE LABEL
60300 PRINTLA$;LN;"   ";
60310 X=X+1
60320 T$(X)=LA$+STR$(LN)
60330 IFPOS(0)>53THENPRINT
60340 IFT$(X)=T$(X-1)THENX=X-1:GOTO60150
60350 IFX>1THENIFT$(X)=T$(X-2) THENX=X-1
60360 GOTO60150
60370 REM SKIP PRINT STATEMENTS
60380 P=P+1:CH=PEEK(P):IFP=NLTHENF=-1:RETURN
60390 IF CH<>34 THEN60380
60400 P=P+1:CH=PEEK(P):IFP=NLTHENF=-1
60410 RETURN
60420 REM ***SORT RTN ***
60430 PRINT:PRINT:PRINT"SORTING"
60440 FORJ=1TOX:L=1
60450 FORI=1TOX:IFT$(I)<T$(L)THENL=I
60460 NEXT
60470 PRINTT$(L):T$(L)="^":NEXT
60480 PRINT:PRINT"SORTED";X;" VARIABLES"
```

The variable lister was inspired by a similar program written for the Apple by Ray Cadmus.

The program will work for 65D3.2 disk BASIC by making the changes noted on line 60080. This is a pointer to the start of the BASIC code. To run the variable lister with disk BASIC, the variable lister program must be appended to your BASIC program. This can be done with the control X procedure.

```
DISK!"LOAD LIST"

LIST[
Program will list out

]
Enter after listing done-
   another ] will appear

DISK!"LOAD YOURS"

control X
List will be appended

RUN 60000
To start Lister

LIST-99

 5 REM DEMONSTRATION RUN
 10 FOR X=1 TO 10: NEXT X
 20 Z=X+Y
 30 Q$="HELLO"

OK
```

Continued

```
RUN 60000
VARIABLE LISTER
EXTRACTS AND LISTS BASIC VARI-
ABLES

AFTER THE FIRST PASS THRU
PROGRAM-SORTS-THEN LISTS
ALPHABETICALLY
X 10 X 10 Z 20 X 20 Y 20 Q$ 30

SORTING
Q$ 30
X  10
X  20
Y  20
Z  20

SORTED 5  VARIABLES

OK
```

LETTERS

ED:

I don't know how to begin. So...Hobbyists and Hackers: why are you selling your machines? The back pages of PEEK are full of ads for ClPs, C4Ps, etc. Are you upgrading? To what? Do you miss the game software that's available for other computers? Did you ever consider getting an Atari video game, instead? They're only about $50-$60 now and the game cartridges are available in copious and dirt cheap quantities. Not only that, but the graphics are reasonably good and you won't have to do any programming. Or is it speed? Well, I've run a little test (a FOR/NEXT loop with "X"; printed 100 times) on Commodores, TIs, Apples, and anything else I've been able to get my hands on and my ClP seems as fast or faster than any of them. If you're selling for this reason, good luck. Be careful about the 16-bit computer trap, too. If its got an 8-bit data bus, it may be slower than you expect for the money you'll have to pay.

If you're selling for educational and technical software, though, maybe you've got a point. But be careful here, too. X-brand computer may have a quadtrillion software packages available for it but how many do you really intend to buy, anyway? (Don't forget to check the OSI SIG on CompuServe - they give software away). Here's a story: A friend of mine recently purchased a Commodore 64 and a disc drive to "educate" his children with a $100.00 SAT program. Well, he's donating the program to the local high school so he can claim it on his taxes. The computer's not being used, either. Just how long do you think a $100.00 SAT program will hold a kid's interest?

Business users, here's one for you. Our company ($40 million in sales) uses a Hewlett-Packard HP 3000 mini-computer. We've got terminals located everywhere. What's the problem? It's bogged down! I don't know how they did it, but the COBOL programmers seem to have the computer checking mostly empty files and spending most of its time inputting and outputting to Winchester drives and spoolers (we seem to have more drives than NASA - maybe it's due to inefficient file storage). What's the point? From an engineering department point of view, I was asked to stop calling the computer room unless it took longer than 1 minute for the prompt to come back onto the monitor. I've watched in awe as our $10,000 HP multi-pen color graphics plotter sat still for as long as 5 minutes without making a mark. I now use my cassette-based ClP at home for most of my engineering calculations. Why? You just try entering 300 data values into a computer that can make you wait for as long as 60 seconds (during "Prime Time") before you can enter the next value. Frustration multiplied by frustration! Engineers, <u>resist!</u> Get your own system. Don't be bullied into riding "free" on your company's mainframe. You don't want it and the computer lords up front will discover that they don't want it (face it, guys, your programs are microprocessor intensive). Also, don't forget that once you've been duped into a terminal, it's tough getting off. Isotron, if you're listening, you're surely missing out if you don't compete in these markets.

So much for the "meanderings," at least I feel better!

Steve McGinnis
Ridgeway, PA 15853

* * * * *

ED:

I am the proud owner of an OSI C4P-MF with 52K of RAM, a single mini-floppy, and a ADS 2000 parallel printer. I have been using this system for many years and been having great fun with it. I have written a variety of software programs mostly in Assembly Language with my notable ones being:

AEXEC* - An Assembly Language boot routine that can create, delete, rename and zero files. It uses 1 track and does not require BASIC.

MODEM - An Assembly Language 300/1200 BAUD terminal emulator. Requires the interrupt on the 6850 be connected to an IRQ line. It uses interrupts when a character is received, provides a menu to select BAUD rates, duplex, # of bits and will send and receive disk files.

I have gotten spoiled with the great software that is available for the DEC Rainbow 100 I use at work. Its software influenced the design of my modem program and is influencing more and more of my software design. I am interested in this type of software for my OSI. I have written to many software sellers, found from the ad section in your magazine, with little or no luck. Most don't bother to return my letters. I know that at one time a lot of software was available for the C4P and am wondering what happened to it all! Also, if there is software available for "Public Domain" how does an average user like me get access to the information. I know that some information is available on CompuServe but have no idea of what. So my questions to CompuServe are:

OSI-SIG on CompuServe

- What does it do?
- How can it help?
- What does it provide?
- How do you join?

Richard P. Bernard
Gulfport, MS 39503

Richard:

First of all, how about sharing some of those programs for the "fun" of other subscribers. But please send them on disk if possible, the disk will be returned if requested.

Regarding your problems with software sellers, let us know more precisely who and what and we will try to go to bat for you as we have for others.

For software, check out the Oct and Nov issues of both '83 and '84. There are about 16 pages of available software. Also check the first classified ad in Jan '85.

Public Domain software comes from folks like yourself. Naturally, we encourage users to send it to us.

As to OSI SIG, see the article in this issue.

Eddie

* * * * *

ED:

Regarding Mr. Gary Florence's letter in the December issue of PEEK(65), I would like to be of assistance.

I suspect the problem that Mr. Florence is having with "Fantastic Copy", is the disk switch which he has installed. Fantastic Copy was not designed with the switch in mind. However, we have just signed an agreement with Mike Putnam to market "Fantastic Copy II", which incorporates the following changes:

1. It works with drives that have head unload or motor shut-off modifications.

2. The copy protect mechanism has been removed so the end user is able to make backup copies for his own use.

3. It is available in two versions; C1P-MF and C4P-MF. It should also work on the C2P-MF.

It is available from Thomas Technical Service (see classified Jan. issue ad) immediately. For those of your readers that want their present copy updated, they should send us their disk and we will update it at a reduced rate.

Since Mike has removed the copy protect mechanism, maybe you could get him to tell your readers how he did it. To the best of my knowledge, no one ever broke the code.

We at Thomas Technical Service would like to commend PEEK(65) for continuing to publish a 'quality' publication for the OSI user. We would like to suggest to the OSI community, at large, that they patronize your magazine and the firms that advertise in it, which after all, is all they have left to protect their investment. We all need to do all we can to convince Isotron that they are making a serious mistake if they are, in fact, dropping the entire lower end of their line. There are fewer and fewer OSI dealers around that an end user, with anything from a Superboard to the C3 Series, can turn to for assistance, parts or software. The growing concern for the future of OSI's involvement in the 'Personal Computer line' becomes more bleak when we see them selling out their existing inventories of computers, boards, and parts, and a lack of any meaningful advertising for the personal computers.

We recently called Bill Thompson at Isotron Support in Aurora for a mini-disk data cable, which used to sell for $7 to $11. We were told that it was no longer a stock item, but COULD be ordered for around $100.00.

We believe that Isotron should make a clear cut policy statement concerning their plans for the future of the OSI personal computer and not leave everyone in the dark. After all, any company who has sold millions of personal computers to the public should, at the very least, advise those owners of corporate decisions that will affect their investment. We still believe the OSI to be the very best computer in its price range and we believe it could have a bright future, but it will take a very serious commitment from Isotron and a complete overhaul of past practices. If hardware houses like D&N and Space-Com could make major strides in upgrading the OSI personal computers by producing things like the D&N 80 board which allows CP/M to be run on OSI and Space-Com could reduce the board count by producing a superior Hard Disk Controller and Generic Computer Products could produce superior memory boards not to mention the famous "Denver Boards". This should tell somebody something. It is our hope that PEEK(65) will be able to ascertain Isotron's plans and not some broad, all encompassing policy statement that is less than meaningful to the average OSI home computer owner, and it is also our hope that the revelation will give encouragement to these same owners.

Walt Thomas
Linden, PA 17744

* * * * *

ED:

The enclosed program 'Pretty' pretty-prints and has features such as: Long line split, 'FOR' loop trace, right justification of 'REMS', Split on 'Then', 'Or', 'And', Align 'Data', Left justify Print statement, String/line table, variable/line table.

It is printed with my new Leading Edge GX100 printer. It is a Gorilla Banana. The price here is $150, which is not bad for a 80 column graphic printer. The line to pin 35 (test) has to be cut before it can be used with OSI, otherwise, it will stay in the test line routine. The user manual is not too informative, but can be used.

Here are a few tips for users who have different printer specifications. Line length of the printout is determined by variable "LL" in line 60480. The value "80" in line 60030 should be changed to about half of your new line length and the value "60" in line 60400 (IFK>60) should be decreased. You may also want to check the CHR$(14) and CHR$(15) statements that produce double high and wide print on the Banana.

Here is a tip using OSI indi-

rect memory as block delete. Assume your program has lines 10 to 1000, and we want to delete 50 to 80.

type LIST 10-40 Shift K return

type LIST 90-1000 Shift K return, Shift M Shift P return

lines 10-40 and 90-1000 are now in upper memory.

type NEW and Control X to bring the program back down.

I would like to ask if anyone has a 'BASE2' printer that I could get for parts?

Henry Jata
1424 Newbridge Rd.
Bellmore, NY 11710

```
                    PAGE 1
                   PRETTY
        H. JATA, 1424 NEWBRIDGE ROAD, BELLMORE, NY 11710
 THIS PROGRAM 'PRETTY-PRINTS', CREATES A STRING AND A VARIABLE TABLE.
 IT WILL TRACE 'FOR' LOOPS BY LEVEL, FORMAT 'REM' TO RIGHT OF PAGE.
  THE WHOLE FIRST PAGE COULD BE USED FOR COMMENTS SUCH AS THESE.
     THESE COMMENTS WILL ONLY APPEAR ON THE FIRST PAGE.
 TYPE THE PROGRAM WITHOUT REMS, PU TO DISK. THE ROUTINE TO USE IT IS:
  DISK!"LO PRETTY": MOVE IT TO INDIRECT MEMORY, DISK!"LO YOUR PROG"
    MOVE 'PRETTY' BACK DOWN WITH A CTRL-X. MAKE SURE THAT THE STRING
   ' 63999' (LINE=60660) & THE NUMBER 63999 (LINE=60200) ARE CHANGED TO
   ' 60000' AND 60000, TYPE 'RUN 60000'. THIS PROGRAM WAS SET UP
    FOR A GX100 PRINTER (GORILLA BANANA). THESE COMMENTS ARE
            TERMINATED BY A '.' STRING.

 60000  PRINT  CHR$ (28)"DID YOU CHANGE 63999 TO 60000" !
          GOTO 60480 $
-------------------------------------------------$ SPACES TO END OF PAGE $
 60010  SP = 66 - LC $
I60020  FOR I = 1 TO SP !
I         LC = LC + 1 !
I       PRINT #DV,"." !
I         NEXT I !
            RETURN  $
-------------------------------------------$ SPLITS AND PRINT LONG LINES $
I60030  FOR I = 40 TO K !
I         IF  MID$ (G$,I,1) =  CHR$ (1) THEN
I           G1$ =  LEFT$ (G$,I) !
I            GOTO 60050 $
I60040  NEXT I !
            GOTO 60400 $
 60050  PRINT #DV,TF$;N$;G1$ !
          G$ =  RIGHT$ (G$,K - I) !
          N$ = " cont> " $
 60060  LC = LC + 1 !
            GOTO 60380 $
-------------------------------------------------$ PEEKS NEXT POSITION $
 60070  P = P + 1 !
          B =  PEEK (P) !
           RETURN  $
-------------------------------------------$ CHECKS FOR NEAR END OF PAGE $
 60080  IF LC < LP THEN
           RETURN  $
-------------------------------------------$ PRINTS END OF PAGE MESSAGE $
 60090   GOSUB  60010 !
          PC = PC + 1 !
        PRINT #DV, TAB( 20);H$(3)" ON PAGE"PC" LINE"DK $
 60100  IF PC <  > 1 THEN
          H = 3 $
---------------$ ELIMINATES ALL BUT PROGRAM NAME AND PAGE ON PAGE TWO ON $
 60110  LC = 0 !
           H$(2) = "PAGE " +  STR$ (PC) !
           FOR I = 1 TO H !
I            E = LL -  INT ( LEN (H$(I))) $
I60120  PRINT #DV, TAB( E / 4);H$(I) !
I         NEXT  I !
            LC = H + 1 $
 60130  N$ =  STR$ (DK) !
           N$ = N$ + " " $
---------------$ ELIMINATES ALL BUT PROGRAM NAME AND PAGE ON PAGE TWO ON $
 60140  IF  LEN (N$) < 7 THEN
```

PRETTY ON PAGE 2 LINE 60140

```
                    PAGE 2
                   PRETTY
 60140      N$ = " " + N$ !
             GOTO 60140 $
 60150  RETURN  $
 60160   GOSUB  60070 !
           D = B !
            GOSUB  60070 !
            D = 256 * B + D !
             GOTO 60200 $
---------------------$ PRINTS END MSGS AND GOES TO STRING/VARIABLE ROUTIN $
 60170  LC = LC + 1 !
            GOSUB  60010 !
        PRINT #DV, TAB( DV / 2 * 15); CHR$ (14)"END OF "H$(3) $
 60180  PRINT #DV, CHR$ (14)"LEN=" INT ((P - 14975) / 1024)"K"TN"LINES"; $
 60190  PRINT #DV,M$;TS - TR"NON-REMS"TR"REMS" CHR$ (15) !
          GOTO 60020 $
-------------------$ GETS LINE NUMBER AND CHECKS FOR END OF PROGRAM $
 60200   GOSUB  60070 !
          D = B !
           GOSUB  60070 !
           DK = B * 256 + D !
            IF DK = 63999 THEN
             GOTO 60170 $
-------------------------------------------------$ ENTRY INTO MAIN LOOP $
 60210   GOSUB  60130 $
 60220  D = 0 !
          SK = 1 !
           GOSUB  60070 $
-------------------$ CHECKS FOR END OF LINE TN=TOT LINES TS=TOT STATEMENTS $
 60230  IF B = 0 THEN
          G$ = G$ + " " +  CHR$ (187) !
           TN = TN + 1 !
            TS = TS + 1 !
             GOTO 60350 $
-------------------------------------------------$ CHECKS FOR TOKENS $
 60240  IF B > 127 THEN
          B = B - 127 !
          GOTO 60280 $
-------------------------------------------$ CHECK FOR END OF STATEMENT $
 60250  IF B = 58 THEN
          G$ = G$ +  CHR$ (14) +  CHR$ (124)
 cont> +  CHR$ (15) !
           TS = TS + 1 !
            GOTO 60350 $
-------------------------------------------------$ ADDS CHARACTER TO G$ $
 60260  G$ = G$ +  CHR$ (B) !
          IF  LEN (G$) < 2 THEN
          G$ = " " + G$ $
 60270   GOSUB  60070 !
          GOTO 60230 $
-------------------------------------------------$ CHECK FOR 'REM' $
 60280  IF B = 15 THEN
          N$ = N$ !
           TR = TR + 1 !
            RF = 1 !
             GOTO 60340 $
---------------------$ CHECK FOR 'FOR' AND SETS FLAG FOR LEVEL TRACE $
 60290  IF B = 2 THEN
          F2 = 1 $
---------------------$ CHECKS FOR 'DATA' AND SET DD FLAG SO DATA LINES UP $
```

PRETTY ON PAGE 3 LINE 60300

```
                    PAGE 3
                   PRETTY
 60300  IF B = 42 OR
           B = 33 OR
            B = 41 THEN
             G$ = G$ + T$(B) !
              GOTO 60350 $
 60310  IF B = 4 THEN
           DD = 1 $
-------------------------------------------------$ CHECKS FOR 'THEN' AND DECREMENTS TF FLAG $
 60320  IF B = 3 THEN
           F3 = 1 $
-------------------$  CHECKS FOR 'PRINT' SET FLAG PL TO PRINT ON ONE LINE $
 60330  IF B = 24 THEN
           PL = 24 $
-------------------------------------$ TRANS. TOTEN ANDPUTS IN G$ AND GOES TO MAIN LOOP $
 60340  G$ = G$ + T$(B) !
          GOSUB  60070 !
           GOTO 60230 $
-------------------------------------------------$ ADDS SPACES TO LEFT G$ $
 60350  D$ =  LEFT$ (BB$,1 + SK) $
-------------------------------------------------$ INC. TF FLAG FOR EACH 'FOR' $
 60360  IF F2 = 1 THEN
           TF = TF + 1 !
            F2 = 0 $
-------------------------------------------$ DONT SPACE FOR REM PRINT OR DATA $
 60370  IF SK > 1 AND
           DD <  > 1 AND
            RF <  > 1 AND
             PL <  > 24 THEN
               G$ = G$ + G$ !
               N$ = LB$ $
 60380   GOSUB  60080 $
-------------------------------------$ TRANS FOR LEVEL INTO CHAR. $
 60390  TF$ =  CHR$ (TF + 72) !
           IF TF = 0 THEN
            TF$ = " " $
-------------------------------------$ CHECK FOR LONG LINES FOR SPLIT $
 60400  K =  LEN (G$) !
           IF K > 60 AND
            DD <  > 1 AND
             PL <  > 24 THEN
              GOTO 60030 $
---------------------$ PUT DASHES IN FRONT OF REM STATEMENTS FOR PRINTER $
 60410  IF RF = 1 AND
           LEN (G$) < DV / 2 * 38 THEN
            G$ = "-" + G$ !
             GOTO 60410 $
-------------------------------------------$ PRINT DATA STATEMENT $
 60420  IF DD = 1 THEN
           DD = 0 !
        PRINT #DV,TF$N$G$; !
           GOTO 60450 $
-------------------------------------------$ MAIN PRINT ROUTINE $
 60430  PRINT #DV,TF$N$G$; !
           DD = 0 !
            I =  FRE (I) $
-------------------------------------$ DEC TF FLAG FOR EACH 'NEXT' $
 60440  IF F3 = 1 THEN
           TF = TF - 1 !
            F3 = 0 $
```

PRETTY ON PAGE 4 LINE 60450

```
                    PAGE 4
                   PRETTY
 60450  PRINT #DV, CHR$ (15) !
           SK = SK + 1 !
            RF = 0 !
             G$ = "" !
              LC = LC + 1 !
               PL = 0 $
 60460  IF B = 0 THEN
           GOTO 60160 $
-------------------------------------------$ CLEAN UP AND GOTO MAIN $
 60470   GOSUB  60070 !
          GOTO 60230 $
-------------------------------------$ SET  UP HEADERS AND COMMENTS  $
 60480  DIM T$(70),H$(60) !
           P = 14974 !
            QF =  - 1 !
             LC = 0 !
              B = 0 !
               LP = 63 !
                LL = 80 !
                 H = 7 $
 60490  POKE 23,250 !
           E = TN = TS = TR = 0 !
            H = 4 !
             TF = 0 !
              GOSUB  60590 $
 60500  G$ = "" !
           M$ = "" !
            LB$ = "          " !
             BB$ = LB$ + LB$ + LB$ + LB$ + LB$ $
 60510  INPUT "DV";DV !
 60510    PRINT #DV, CHR$ (15)"Line up Printer NOW" CHR$ (13) $
 60520  INPUT "PROG NAME";H$(3) !
           IF H$(3) = "TEST" THEN
            H$(4) = "JATA" !
             GOTO 60560 $
 60530  INPUT "PROGRAMMER";H$(4) !
           H = 4 $
 60540  H$(3) =  CHR$ (14) + H$(3) +  CHR$ (15) $
-------------------------------------$ CHECK FOR '.' TO END HEADERS $
 60550  H = H + 1 !
           H$(H) = BB$ !
            INPUT "COMMENTS";H$(H) !
             IF H$(H) <  > "." THEN
              GOTO 60550 $
---------------------$ PRINT DASHES TO LINE UP PRINTER ANS N FOR TEST LINE $
 60560  H$(1) = "----------" !
I           FOR I = 1 TO 3 !
I            H$(1) = H$(1) + H$(1) !
I             NEXT I $
 60570  INPUT "YES";Y$ !
           IF  ASC (Y$) <  > 89 THEN
        PRINT #DV,H$(1) CHR$ (13); !
           GOTO 60570 $
-------------------------------------------------$ GOTO MAIN $
 60580  PRINT  (28) !
          PC = 1 !
           GOSUB  60110 !
           GOTO 60160 $
-------------------------------$ EXTRACT FUNCTION AND PUT IN T$(I) $
```

PRETTY ON PAGE 5 LINE 60590

```
------------------------------------------------------------------------
                PAGE  5
                PRETTY
I 60590  FOR I = 644 TO 870 !
I             NAME$ = NAME$ +  CHR$ ( PEEK (I) AND
I              127) $
I 60600  IF` PEEK (I) > 127 THEN
I             W = W + 1 !
I              T$(W) = NAME$ + " " !
I               NAME$ =  CHR$ (1) + " " $
I 60610  NEXT I !
              T$(15) = "*" !
               T$(13) =  CHR$ (14) + T$(13) +  CHR$ (15) !
                RETURN  $
 ---------------------------------* START OF STRING VARIABLE TABLE ROUTINE $
 60620  AD = 14975 !
             DIM V$(100) !
              QF = 0 !
               VR$ = "-STRING/VARIABLE TABLE" $
 60630  PRINT #DV,H$(1) !
 60630    PRINT #DV, CHR$ (13) TAB( 20)H$(3)VR$ CHR$ (13) $
 60640  ST =  PEEK (AD) + ( PEEK (AD + 1) * 256) $
 60650  LN$ =  STR$ ( PEEK (AD + 2) + ( PEEK (AD
 cont>  + 3) * 256)) !
             AD = AD + 3 $
 60660  IF LN$ = " 63999" OR
             CN = 100 THEN
              60860 !
 ---------------------------------------------* CHANGE " 63999"TO " 60000" $
 60670  A$ = "" !
             AD = AD + 1 !
              IF AD = ST THEN
                60640 $
 60680  IF QF = 3 THEN
             AD = AD - 1 !
              QF = 0 $
 60690  IF  PEEK (AD) <  > 34 THEN
             60740 $
 60700  IF  PEEK (AD) = 34 THEN
             QF = QF + 1 !
              QQ = 0 $
 60710  IF QF =  > 2 THEN
             AD = AD + 1 !
         PRINT #DV, CHR$ (34) !
              GOTO 60740 $
 60720  IF QF = 1 AND
             QQ <  > 1 THEN
              QQ = 1 !
         PRINT #DV,LN$; TAB( 7)""; $
 60730  PRINT #DV, CHR$ ( PEEK (AD)); !
              AD = AD + 1 !
                IF QF = 1 THEN
                 GOTO 60700 $
 60740  QF = 0 !
               IF  PEEK (AD) = 131 OR
              PEEK (AD) = 142 THEN
               AD = ST !
                GOTO 60640 $
 60750  IF  PEEK (AD) = 131 OR
             PEEK (AD) = 142 THEN
              AD = ST !
               GOTO 60640 $
                PRETTY ON PAGE 6  LINE 60760
------------------------------------------------------------------------
                PAGE  6
                PRETTY
 60760  IF  PEEK (AD) > 90 OR
             PEEK (AD) < 65 THEN
              60670 $
 60770  A$ = A$ +  CHR$ ( PEEK (AD)) !
             AD = AD + 1 !
              IF AD = ST THEN
               60820 $
 60780  P =  PEEK (AD) !
             IF P > 47 AND
              P < 91 AND
               P <  > 61 AND
                P <  > 58 THEN
                 60770 $
 60790  IF P = 34 THEN
             QF = 3 $
 60800  IF  PEEK (AD) = 36 THEN
             A$ = A$ + "$" $
 60810  IF  MID$ (A$,3,1) <  > "$" THEN
             A$ =  LEFT$ (A$,2) $
 60820  IF  LEN (A$) < 3 THEN
             A$ = A$ + " " !
              GOTO 60820 $
 60830  A$ =  LEFT$ (A$,3) !
             IF A$ = "REM" THEN
              AD = ST !
               GOTO 60640 $
I 60840  FOR I = 1 TO CN !
I             IF  LEFT$ (V$(I),3) = A$ THEN
I              60900 $
I 60850  NEXT  !
             CN = CN + 1 !
              V$(CN) = A$ + LN$ !
               GOTO 60670 $
I 60860  FOR I = 1 TO CN !
I 60860    PRINT #DV, TAB( 10) CHR$ (14) LEFT$ (V$(I),3) CHR$ (15) $
I 60870  PRINT #DV," " RIGHT$ (V$(I), LEN (V$(I)) - 3) $
I 60880  NEXT  !
 60880  PRINT #DV,"" !
              INPUT "RESTART";G$ !
               IF  ASC (G$) = 89 THEN
                60860 $
 60890  PRINT "DO NOT S A V E THIS PROGRAM" !
             STOP  $
 60900  IF LN$ =  RIGHT$ (V$(I), LEN (LN$)) THEN
             60670 $
 60910  IF  LEN (V$(I) + LN$) < 200 THEN
             V$(I) = V$(I) + LN$ !
              GOTO 60670 $
 60920  GOTO 60850 $
.
.
.
.
.
.
.
.
.
.
.
.
                        END OF PRETTY
LEN= 4 K 129 LINES 226 NON-REMS 37 REMS
------------------------------------------------------------------------
                PRETTY-STRING/VARIABLE TABLE
 60000 "DID YOU CHANGE 63999 TO 60000"
 60020 "."
 60050 " cont> "
 60090 " ON PAGE"
 60090 " LINE"
 60110 "PAGE "
 60130 " "
 60140 " "
 60170 "END OF "
 60180 "LEN="
 60180 "K"
 60180 "LINES"
 60190 "NON-REMS"
 60190 "REMS"
 60230 " "
 60260 " "
 60390 " "
 60410 "-"
 60450 ""
 60500 ""
 60500 ""
 60500 "      "
 60510 "DV"
 60510 "Line up Printer NOW"
 60520 "PROG NAME"
 60520 "TEST"
 60520 "JATA"
 60530 "PROGRAMMER"
 60550 "COMMENTS"
 60550 "."
 60560 "----------"
 60570 "YES"
 60600 " "
 60600 " "
 60610 "*"
 60620 "-STRING/VARIABLE TABLE"
 60660 " 63999"
 60670 ""
 60720 ""
 60800 "$"
 60810 "$"
 60820 " "
 60830 "REM"
 60870 " "
 60880 ""
 60880 "RESTART"
 60890 "DO NOT S A V E THIS PROGRAM"
        SP
  60010 60020
        LC
  60010 60020 60060 60080 60110 60120 60170 60450 60480
        I
  60020 60030 60040 60050 60110 60120 60430 60560 60590 60600 60610 60840 60860
 60870 60900 60910
        DV
  60020 60050 60090 60120 60170 60180 60190 60410 60420 60430 60450 60510 60570
 60630 60710 60720 60730 60860 60870 60880
        K
  60030 60050 60400
        G$
  60030 60050 60230 60250 60260 60300 60340 60370 60400 60410 60420 60430 60450
 60500 60880
        G1$
  60030 60050
        TF$
  60050 60390 60420 60430
        N$
  60050 60130 60140 60280 60370 60420 60430
        P
  60070 60180 60480 60780 60790
        B
  60070 60160 60200 60230 60240 60250 60260 60280 60290 60300 60310 60320 60330
 60340 60460 60480
        LP
  60080 60480
        PC
  60090 60100 60110 60580
        H$
  60090 60110 60120 60170 60480 60520 60530 60540 60550 60560 60570 60630
        DK
  60090 60130 60200
        H
  60100 60110 60120 60480 60490 60530 60550
        E
  60110 60120 60490
        LL
  60110 60480
        D
  60160 60200 60220
        TN
  60180 60230 60490
        M$
  60190 60280 60500
        TS
  60190 60230 60250 60490
        TR
  60190 60280 60450
        SK
  60220 60350 60370 60450
        RF
  60280 60370 60410 60450
        F2
  60290 60360
        T$
  60300 60340 60480 60600 60610
        DD
  60310 60370 60400 60420 60430
        F3
  60320 60440
        PL
  60330 60370 60400 60450
        Q$
  60350 60370
        BB$
  60350 60500 60550
        TF
  60360 60390 60440 60490
        LB$
  60370 60500
        QF
  60480 60620 60680 60700 60710 60720 60730 60740 60790
        Y$
  60570
        NA
  60590 60600
        W
  60600
        AD
  60620 60640 60650 60670 60680 60690 60700 60710 60730 60740 60750 60760 60770
 60780 60800 60830
        V$
  60620 60340 60850 60860 60870 60900 60910
        VR$
  60620 60630
        ST
  60640 60670 60740 60750 60770 60830
        LN$
  60650 60660 60720 60850 60900 60910
        CN
  60660 60840 60850 60860
        A$
  60670 60770 60800 60810 60820 60830 60840 60850
        QQ
  60700 60720
```

★ ★

ED:

I am very much interested in the things that Jan Synek mentioned in the letter in the October issue. In particular, I am interested in any information Jan has about the UCSD PASCAL SBIOS, and I am interested in a shorter/simpler boot sequence for DOS/65. I have the PASCAL system for my C4PMF, but do not have the source for SBIOS. Jan, did you disassemble the SBIOS or do you have the source? Could we get Jan to write an article or two for PEEK(65) further describing the above?

I have DOS/65 version 2 and I am quite pleased with it. I have the DOS/65 LOADER loaded by BEXEC* using 65D but would be very interested in a more direct booting sequence. I have modified the SIM to support formfeed, home, and clear-to-end-of-screen for the C4P; it also has been modified so that the beginning and end of the screen can be on a line (64 byte) boundary rather than a page (256 byte) boundary. The latter modification was necessary so that I could see the bottom line of the screen without having to give up three additional lines. Also, I modified the FORMAT program to support drive selection. I forwarded these changes to Richard Leary, the author of DOS/65; he said he would use the revised FORMAT, but did not mention the modified SIM. I assume this means he will not use the modified SIM. If anyone is interested in the revised SIM for DOS/65 for the C4PMF, send me a signed statement that you own a legal copy of DOS/65 and three dollars to cover the cost of a cheap diskette, mailer, plus postage, and I will forward the revised SIM to you. Add two more dollars if you want a good quality diskette.

I purchased a CAT modem from Isotron but cannot get it to work with my C4PMF. The cable I fabricated used lines 2, 3, 7, and 8, and I am using the MODEM program from OSI. Please tell me how to use a modem with the C4P.

Bill Beshures
294 Milford St., Apt. 32
Rochester, NY 14615

Bill:

I will probably disappoint you, but I only bought the UCSD PASCAL in my search for a better operating system. I did play with it a little, but not enough for me to do an in depth report on the subject. I have version II.Ø and, despite a few bugs in some utility programs, it works quite well. I expect, however, to use only the editor and assembler occasionally.

I do most of my programming in Assembly language, and so I wanted an easy and complete interface to the operating system functions. One of the best operating systems in this regard is FLEX (for 68ØØ family of processors) by Technical Systems Consultants.

The PASCAL operating system is in p-code, which makes it slow and very inefficient to use as a general purpose operating system. It also requires a large amount of memory for the p-system in addition to 4Ø pages for the interpreter and 14 pages for SBIOS.

Yes, I have disassembled the SBIOS, but luckily DOS/65 came along before I got too deeply involved in the p-system. I will send a copy of the disassembly to anyone interested for $3.ØØ to cover postage and copying cost. Please note that there are about 3Ø pages in the listing with only a very few hand written comments.

In my last letter I have mentioned the SBIOS keyboard routine. I have since upgraded to DOS/65 version 2.Ø and was glad to see that Richard Leary used it in the new SIM.

To load DOS/65 directly, a modification is required to BOOT, SIM, and to SYSGEN.COM to generate the self loading system. SYSGEN is not in the public domain, so if I get a permission from Richard Leary, I will be glad to write an article for PEEK(65) describing the modifications.

Jan Synek
Chicago, IL 6Ø651

Bill:

Peek's answer to your last question. Not always, but usually pin 8 (DCD) needs to be held low by a jumper to pin 7 (ground).

Eddie

* * * * *

ED:

I wonder if anyone is aware of the fact that there are many users like myself that have purchased used OSI machines and are not into "programming" by desire but by necessity, don't know a USR call from a duck call, don't have the time or background to learn how to program in machine code, change BIOS, I/O registers or solder jumper wires from pin 7 to pin 12, or address FCØØ and FDD3, whatever that might be.

We would like to see articles on business applications programs, communications programs, "turnkey" modification systems that upgrade our equipment without us having to be an expert programmer. I realize that the prevailing train of thought is that a "serious" user will learn and become an "expert" in source code, machine code, Machine/Assembly "language" programming, and while I AM learning these things, and PEEK provides excellent information on them, (I really do appreciate all the articles) we have businesses to run and work to get done NOW! My secretary does not have to be an electronic engineer to use the typewriter or office copier; why does she have to be one to use a computer?

The reviews that are done on programs are very helpful and welcome.

What I am trying to say, I suppose, is the majority of articles are very technical and far beyond a lot of us "beginners". But, that may be the audience you are addressing, and we will just have to plod along and by bits and pieces, catch up.

Thanks for letting me have my say, and I guarantee I will send in a program we have developed that some may have use for.

PEEK(65) is the ONLY source of info for us OSI users and consider each issue to be very valuable. Keep it up!

Raymond Roberts
Ferndale, WA 98248

Raymond:

Hang in there! You <u>are</u> winning! You already have one of the fastest machines going.

For our part, we are making a concerted effort to look at things from your point of view. That's why we have Beginner's Corner, Assembly Language course, and more reviews on software, not to mention our annual software issues. For starters, get yourself a word processor, a Data Base Manager, and maybe Planner Plus. With those in hand, you will find yourself using

the machine a lot more.

If you tell us what you expect of your machine, maybe we can give you some more help.

Eddie

* * * * *

ED:

I discovered another quirk which occurs both in OSI ROM and disk BASIC. Question: How can line 30 be executed without using RUN 30 ??

```
10 PRINT"THIS IS LINE 10"
20 END
30 PRINT"THIS IS LINE 30"
```

The answer is enter RUN and when the program stops in line 20, a CONT or continue statement will execute line 30. This little effect burned me since I usually write programs with subroutines tacked on after the mainline program ends. The CONT statement caused the execution to fall through the normal END and the computer started executing the next subroutine. I guess a better plan would be to always have the END statement of the physical end of the program and have GOTOs pointing to it.

Earl Morris
Midland, MI 48640
* * * * *

HUMOR!

Real programmers don't write COBOL. COBOL is for wimpy Application programmers.

Real programmers never work 9 to 5. If they are around at 9 a.m., it's because they were up all night.

*** * * * * GIVE AWAY * * * * ***
Multi-Strike Printer Ribbons

What do you currently pay for a multi-strike ribbon cartridge? About $4.00 each in lots of 6?

We have found a solution that may cause you never to use a fabric ribbon again. 1) Did you know that most all multi-strike ribbon cartridges use the same ribbon bobbin? It is just pressed on a different size hub and put in your cartridge type. 2) We have found a source of recently outdated (yes, many are dated) Diablo Hi-Type I cartridges. We took the oldest one we could find, put it in our NEC cartridge and printed this ad. Now, honestly, do you see any difference? We can't either. So we are offering those of you who use Hi-Type I, or are willing to pry open whatever cartridge you are using and replace the bobbin, a deal you can't refuse.

Buy one box of 6 cartridges for $8.00 and we will give you a second box FREE. That's 66.66 cents a piece or 83% off. At that rate, how can you lose? Add $3.00 for postage and handling. 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. Order NOW, supply limited!

AD$

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

* * * * *

MUST SELL. Still in original wrappings, KEYWORD CP/M Word Processor, CP/M v 2.25. Cost was $400.00 each. Will sacrifice $250.00 each, or $400.00 for set. Reply PEEK, Box K, c/o PEEK(65), P.O. Box 347, Owings Mills, MD 21117.

* * * * *

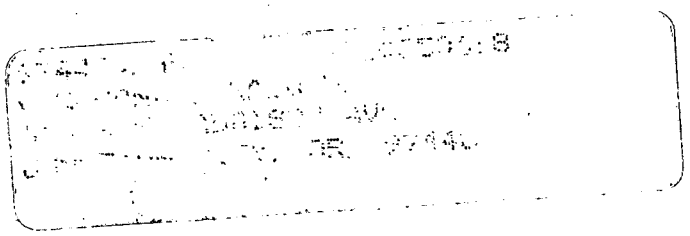
WANTED: Cipher 1/2" tape drive with OSI interface board, in good working condition. Also OKI 2350 printer. Call collect Ting Barrow (212) 989-1945.

PEEK (65)
The Unofficial OSI Users Journal

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

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

DELIVER TO:



GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

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

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

( ) Cash enclosed ( ) Master Charge ( ) VISA

Account No. _____ Expiration Date _____

Signature _____

Name _____

Street _____

City _____ State _____ Zip _____

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

POSTAGE MAY VARY FOR OVERSEAS