PEEK [65]

The Unofficial OSI Journal

Summer 1986
Vol. 7 Nos. 5, 6, 7, & 8

Column One

Despite appearances, things haven't totally fallen apart here at PEEK[65]. Quite the contrary. If the size and cover of this issue doesn't make it obvious, this issue is covering a multitude of sins. The next issue will have a cover month of September, and at that time we will go back to our regular monthly schedule.

The reasons for this Summer issue are many and varied. First of all, I wasn't able to recover the original schedule. Between all of my commitments and other reasons, there just haven't been enough hours in the week to make much progress. But one of the most important reasons for the delay of this issue in particular has been the flurry of activity that directly relates to our discussions of new hardware for OSI systems. Several new product announcements were imminent and I was hoping to include them in this issue. Those announcements have not been made public as yet. What I can say is that all OSI owners will soon have major upgrade paths available within the next 30 to 60 days at very reasonable prices. Everybody - from Superboards to serial systems.

Even though this issue is more than twice as large as normal, I realize that it doesn't make up for the intervening issues that would normally have been published. Therefore, I have extended the subscriptions of everyone who was current through June by two months. I know this won't satisfy everyone, but it is as fair as I can make it. Note that the mailing labels on this issue DO NOT reflect the extension to your subscription, nor did the recent renewal forms I recently sent out to many of you.

Inside This Month:

Back on the news front, this issue contains a lot of articles that have been in the PEEK library for some time, but that we simply didn't have room to publish before. It has been, and remains my policy where possible to not break up articles over more than one month. If I present the article, then the entire program listing should be printed as well. In addition, and again where possible, articles are printed contiguously within each issue, so you don't have to page back and forth between the articles and the listings. This has led to some of the editor's curse known as "white space", but I think it makes PEEK eminently more useful.

Cleaning out the library in this fashion means that I am in desparate need of new material. The April issue contains several topics that I hope you'll consider. The library has a couple of articles that are incomplete. I hope those of you among those authors will complete your work and send it in. It will be most appreciated.

Special thanks goes out this month to Larry Hinsley of Software Consultants and Ed Richardson of the Autralian group KAOS for their contributions to this issue. Matt Holcomb shows us how to list out the symbol table in the OSI Assembler/Editor. Daniel McDonald provides us with a nifty random number generator. Doug Johansen demonstrates a way to display over-sized characters on video systems.

Your humble editor has been busy as well. I have included several articles in this issue including instructions for using the Data Library in OSI SIG, a program for getting true random access files under OS-65D, the start of a series of articles on the innards of OS-65U, and a few other things I hope you'll find interesting.

Thanks to all of you for your help and patience over the past few months. It's been a pleasure dealing with all of you and writing in this forum. With your continued support, the future looks brighter for all of us than it has in many years.

User Survey Final Results

The User Survey was a huge success as far as I'm concerned. It really helped me to get a good idea of what PEEK[65] readers wanted and how willing you are to part with your hard-earned cash to get it.

40 people mailed in responses. That's about the number I expected considering the number of subscribers and the summer computer doldrums. Of those, some 16 entries listed multiple systems owned by the submittor. The breakdown by model went as follows:

8" Serial: 18
8" Video: 17
C4P-MF (or equivilent): 17
C1P-MF: 4

38 respondents had printers, and 25 owned modems. The vast majority listed ownership of OS-65U V1.44 and OS-65D V3.3.

OS-DMS was far and away the most frequently mentioned commercial software package, with 11 people naming it as their most often used software. Close behind was DQFLS' WP-6502 word processor at 8. OSI's WP-2 and WP-3 came in third with 6 respondents. Fourth place went to my own Term-Plus program. 9 people mentioned various accounting packages from other sources, but none gained any significant following in our survey.

Amazingly, interest in both new CPU and graphics hardware waned in the final weeks. Much of that can be attributed to the influx of a lot of serial system owners. The final tabulations went as follows:

New CPU
$0: 8
$15: 3
$100-$200: 15
$200-$500: 7
$500-$1000: 1

New Graphics
$0: 18
$50-$100: 13
$100-$200: 3
$200-$500: 5

Not all entries voted in the above figures and many people made ambiguous comments that made it hard to put their vote in any catagory. The main reason for the confusion was that a lot of people weren't sure why they would want any new hardware. Hopefully the rest of this issue will clear up any such mysteries. I found it interesting that within the above tallies, some 18 people were willing to commit to upgrading both their CPU and graphics capabilities. The heavy NO voting was almost all attributable to serial system owners, which is more than understandable in the OSI world.

Toward the end of tallying up all of the figures, it became clear that people's software wish lists and their suggestions for topics for articles in PEEK were closely related. It is abundantly clear that owners of all OSI systems are clamoring for new word processing software. Many included specific features they wanted to see, such as disk-based software, interchangeable fonts/type styles, and the ability to do superscripts and subscripts. 17 people mentioned a desire for a new WP. Second place went to a desire for an assembler that would be compatible with the new CPU chips we're all discussing. A good number of people also wanted better terminal software.

Hardware articles dominated the desires of the respondents. Some wanted articles about interfacing various peripherals, but a significant number expressed an interest in ways to add new and better disk drives to their systems. I think the past 3 issues of PEEK demonstrate that these desires have been heard for a long time and something is being done to help. On the software side, there was a lot of interest in assembly language, which I was pleased to see. There was roughly a 50-50 mix of people mentioning assembly language information on the new CPUs and interests in modifying either 65D or 65U. You can count on PEEK to be a steady source of such information.

One area in which PEEK has been weak is in the area of OS-65U articles that deal with hard disk management, Level 3 operations, and specific OS-DMS applications. While over the years there have been a slew of patches to EDMAFL, we haven't really gone very deep. That situation is also being addressed. It is clear to me that as the PEEK[65] community matures, they are becoming ever more dominated by business users. There is no doubt this trend will continue since OSI no longer manufactures video systems.

I was disappointed by the number of people who expressed a desire for software that is not only available, but is advertised here in PEEK. I'm the first to admit that the software sold here could be improved, and it will be, but what is available often met the specifications mentioned. So take a closer look at those ads, folks!

Overall, I think the survey shows that both PEEK[65] and the commercial vendors are on the right track. There are people addressing all of the desires expressed and that tells me that we have an exciting autumn to look forward to. Thanks once again to all of you who responded to the survey.

Copyright 1986 PEEK[65] All rights reserved
published monthly
Editor: Richard L. Trethewey

Subscription Rates	Air	Surface
US		$22
Canada & Mexico (1st class)		$38
Europe	$42	$48
Other Foreign	$47	$48

All subscriptions are for one year and are payable in advance in US dollars. For back issues, subscriptions, or other information, write to:
PEEK[65]
P.O. Box 586
Pacifica, CA 94044 415-359-5708

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

16 Bits: The New Horizon

by Richard L. Trethewey

The discussions of new 16-bit versions of the 6502 microprocessor have been brewing for several years now. As I write this, many projects both inside and outside the OSI community are coming to fruition at a most opportune time.

Of the enhanced versions of the 6502, the 65C02 has been the most popular to date. When Apple Computer chose this chip to power its IIc, the chip began to be available in quantity and at a price that was affordable. The 65C02 added a number of instructions to the original 6502 that made it attractive to the software buffs, and since it was pin-compatible with its predecessor, many OSI owners adopted it and have been using it for some time.

However, the 65C02 remains an 8-bit microprocessor and the world has been demanding more power than the 65C02 has been able to provide. The Western Design Center of Mesa, Arizona has designed two microprocessors that bridge the 8 and 16 bit worlds, namely the W65C816 and the W65C802 (which I'll refer to as simply the 65816 and 65802, respectively).

The 65816 and 65802 are true 16-bit microprocessors with full 16-bit registers that correspond to their 65xx predecessors. The 65816 is capable of 24-bit external addressing for a range of 16 megabytes of memory, and the 65802 is capable of 16-bit external addressing for a range of 64 kilobytes. Both of these chips have an emulation mode that make them fully software compatible with the 6502. The 65802 is pin compatible with the 6502, making it a natural replacement in our systems. In addition, the two chips are compatible with each other, save for the hardware differences. Rather than get too deep into a technical discussion of the chips, I thought it would be better to answer some of the questions that people asked in the User Survey.

When we speak of a 16-bit microprocessor, we mean that the chip is capable of dealing with data in 16-bit chunks for all of its normal operations including addition, subtraction, and bit manipulation. It's hard to generalize about what kinds of speed gains the 16-bit chips offer over their 8-bit counterparts, but a quick look at some typical assembly language code should be enlightening.

Consider the code to add two 16-bit values. The 6502 code would look like this:

Code	Cycles
LDA $B000	4
CLC	2
ADC $B002	4
STA $B002	4
LDA $B001	4
ADC $B003	4
STA $B003	4
Total	26

Now for the 65816 in the 16-bit mode, the code looks like this:

Code	Cycles
LDA $B000	6
CLC	2
ADC $B002	6
STA $B002	6
Total	20

Just in terms of raw speed, you're getting a 23% increase. But in addition to that, the 16-bit code saves 9 bytes (10 vs. 19)! These savings are not always going to apply, especially when the software has to deal with 8-bit hardware. However, if we were to stay conservative and estimate a general speed increase of 15% and a size decrease of 30%, there are some clear advantages worth investigating.

Of course, in the near term we still have to deal with our regular 6502-based software that cannot take advantage of these features without modifying the hardware to use a higher system clock speed. But the advent of these two chips allows us to make incremental improvements in our hardware and software to suit our needs and pocketbooks. The size of the leap you make is very much under your control.

As mentioned at the start of this article, the 65802 is a pin-compatible replacement for the 6502. Pop the old one out and the new one in and you're in business. Your current software wouldn't know the difference, although your hardware would breathe a tad easier due to the CMOS power savings.

In the near term, I would expect to see patches to BASIC and the various operating systems, much like my Hooks into BASIC, which will take advantage of the 16-bit capabilities of these chips. It is the longer term that is really thrilling to me as a programmer.

Two key elements of the OSI system architecture have hindered development of sophisticated software. First among these is the system memory map. The hardware is scattered all over the top of the memory map limiting it to only 48K of contiguous memory. The second problem is the ancient OSI disk interface. By attacking the first obstacle, we can do wonders for making up for the second.

When you have the ability to address a lot of memory - contiguous memory, many doors open up. Database software can hold linked lists in RAM so that sorting, searching and other operations are made significantly faster. Spreadsheets can be huge and entirely RAM-resident for speed and versatility. Word processing will no longer be limited to 5 to 10 pages. Those are real benefits and they're just around the corner. The key is moving to the 65816 and it's ability to address memory beyond our traditional base 64K. Many of these programs will also be useful to those who choose the 65802.

If it isn't obvious by now, this article is written with some specific hardware in mind. The Toronto user group TOSIE, who have given us so many treasures in the past, is working on a 65816-based CPU board. Other hardware announcements are most certainly in the offing from many sources. There is no doubt in my mind that the 658xx family will be the hot topic in PEEK[65] for a long time and I'm looking forward to it.

ADVENTURES AND THE OSI

By: Ed Richardson
Courtesy of SUPERBOARD
Newsletter of the Ohio Superboard User Group
146 York Street, Nundah 4012
Queensland, Australia

AN INTRODUCTION TO ADVENTURE

Adventure games have been played on computers of all types for many years, and are one of the most difficult games to play, and certainly the hardest to create. Essentially, the player is in a universe of the writer's imagination, questing for a goal which is often obscure, and having to solve problems which should have logical solutions, but sometimes don't. Usually, the objectives are to survive, and find some sort of treasure. The location can be caves, castles, outer space, or even in open surroundings.

The first adventure was simply titled "Adventure" and was written in Fortran, to run on a DEC PDP-10 computer with 300k of memory. Of course, the introduction of the microprocessor meant that adventures had to be crammed into much smaller memory, usually 16k. Much of the magnificent wording which described rooms in the original Adventure had to be left out. An example of such wording follows: -

"You're at a low window overlooking a huge pit, which extends up out of sight. A floor is indistinctly visible over 50 feet below. Traces of white mist cover the floor of the pit, becoming thicker to the left. Marks in the dust around the window would seem to indicate that someone has been here recently. Directly across the pit from you and 25 feet away, there is a similar window looking into a lighted room. A shadowy figure can be seen there peering back at you. What now?"

This is nowhere near the longest room description in Adventure, but such descriptions could not possibly be used in even a 64k machine. The IBM, of course, offers such possibilities. Other machines could possibly call in the description of the rooms from disk, however, most adventures for home computers merely truncate the description drastically to only the most essential details.

Probably, the most advanced and complex adventure game is ZORK, written entirely in compiled code. While ZORK does not have enormous room descriptions, it does accept almost any answer. ZORK was also written on a PDP-10, and is usually supplied on 2 to 3 disks, which says something of its size. ZORK has its own interpreter, just like a BASIC interpreter, which makes it easier to adapt to different processors. With ZORK, you can say "Take the bomb and put it at the foot of the door". Almost all other adventures would require "Take bomb","Put bomb", WHERE?,"Door".

Of course, ZORK has already been eclipsed by graphical adventures and also role playing games typified by Dungeons and Dragons. The ultimate adventures will come when the Laser video disk is coupled to home computers. You will then see the rooms through your character's eyes. You will also be able to select your character's traits and so the adventure can be different every time you play it, the final outcome depending on the role you have adopted. With varying strengths of physical and intellectual capacity, several million different characters would be possible. A strong heart would also be recommended for the player. To see yourself about to be destroyed would provide quite a shock. The psychiatrists might do well out of it!

However, we will have to wait for this. For the moment, we will be limited to simple 8 or 16k adventures for the OSI. Although several quite good 8k adventures have been written, (even 5k ones!) I really think 16k is more appropriate. A really good adventure should have perhaps 40 or more rooms, and this is simply not possible with 8k.

SOLVING ADVENTURES

There are two cardinal rules to observe when setting out on a new adventure. The first one is to look at everything, and the second is to draw a map as you travel. Most objects you come across will have some role to play, and most will have only one role, though this is never certain. With the OSI adventures, you won't find many red herrings or dead ends, simply because the 8K memory doesn't allow any space for it. However, in 16K games, you will find routes which lead absolutely nowhere, and objects which have not the slightest use except to annoy you and delay the solving of the puzzle.

Drawing up a map will always enable the adventure to be solved much faster, as it prevents random wanderings over the same ground. On your map, you should name each room and mark the contents as you first find them, and also note the exits. Wherever you start drawing your map on the paper will almost certainly be the wrong place, so to avoid crunching up the last part into some obscure corner, have a second sheet ready to stick on. Some adventures have one-way movement which is rather hard to represent on a map. Perhaps a different colour pen might help there. If your adventure contains anything which suggests a maze, you should most carefully document your journey. This will save much wandering in a later game when you meet with that inevitable nasty fate in early games.

Some games have random distribution of objects as in our Treasure Quest game which will follow, however, most real adventures have a fixed and logical method for solving the puzzle. If you encounter a problem, you will not be able to solve it without the correct object. Sometimes you will not be able to return to get it, and have to replay the game over. Some adventures have a "save the game" feature, though I haven't seen one for OSI. This enables you to recall a partly completed game, and is a very useful thing to do before some heroic but risky venture, such as attacking a dragon!

ATTENTION: DEALERS!

PEEK[65] needs new subscribers and you need new customers, and together we can make it happen with our own Co-op advertising program. This program pays dealers for signing up new subscribers with free ad space in PEEK[65]. Just five paid subscriptions will earn a 1/9th page advertising credit in PEEK[65].

Call or write today for details and your free promotional materials. Making a PEEK[65] subscription a part of every sale is painless and profitable. This time, "Co-op" pays you.

540 Video Driver with Color Controls

by Software Consultants
6435 Summer Avenue
Memphis, TN 38134

(Editor's Note: We are again indebted to Software Consultants for making this code available. The software and accompanying article were originally written some time ago and I have made changes to the article to reflect the current state of the OSI community. Ergo, any errors or inconsistancies are my fault and not Software Consultants'.)

This routine was written to provide the users of OSI video based systems most of the features found in the standard terminals in use on microcomputers. In addition, it gives you several options not available on any terminal. The program consists of a machine code routine tied into OS-65D and as such may be used with any of the languages presently supported by OSI. The routine takes up 1.25K of memory and loads in the top portion of the available memory.

The routine was designed to be as easy to use as possible while still allowing the utmost in end-user flexibility. This is done by providing a carefully chosed set of command codes that give you complete control over all parameters associated with the 540 video board. In addition, other control functions can be easily added and linked to the video system.

One concept that is used extensively in this set of routines is that of windows and windowing. This concept will be familiar to users of OS-65D V3.3, but may still be new to some of you. A window is the area on the display that is recognized and used by the video driver software. The video routine supplied with OS-65D V3.2 and earlier used all but the bottom few lines of the 540's display area as its "window" and all printing and scrolling was done within this area. This new code allows you to define any rectangular area on the display as your "window" and then save and enable these "windows" as you wish.

Most of the command codes operate relative to the present window. This enables you to print something at one place on the screen and then by carefully choosing your window parameters you can print, clear, or do anything else you like to other portions of the screen without affecting what you originally printed. While some of these concepts may seem difficult at first, after a little use you will wonder how you ever did without it.

We will now take each command code and explain its function and use. Any questions you have can probably be answered by sitting down at your computer and experimenting. The ASCII number of the command code is shown along with its function and any special instructions for its use. From BASIC, you use these command codes by simply printing the command code with the CHR$ function of BASIC.

Command Codes and Function

(1) Set Master Window - This initializes the 540 video, setting it to 64 characters per line with 25 lines available. This is a "special" window and is not considered part of the define/set window routines. Any time this command code is printed, a window starting at $D100 and ending at $D7C0 with a line length of 64 characters will be set. This command code does not affect the color or the sound.

(2) Set 64 - sets the video to the 64 characters per line format. Does not affect the color or sound.

(3) Set 32 - sets the video to 32 characters per line. Also, does not affect the color or sound.

(4) Clear 540 - clears the entire video display without moving the cursor.

(5) Vertical Plot - used to plot a vertical line from the present cursor position. To use, print the command character followed by the number of positions to plot, and then the character to draw while plotting. For example, in a BASIC program the statement:

```
PRINT CHR$(5);CHR$(20);CHR$(161)
```

will print a vertical line from the present cursor position with a length of 20. The character printed will be a solid block. The cursor will be at the end of the line.

(6) Horizontal Plot - same as the vertical plot except that the line is horizontal.

(7) Bell - This control code is not implimented, but is reserved for the bell function.

(8) Backspace - this is a non-destructive backspace.

(9) Set window to color - this sets the present window to a certain color. To use this feature from BASIC, you;

```
PRINT CHR$(9);CHR$(x);
```

where "x" is the desired color code. This also sets the individual character color to the window color (see number 11).

(10) Line feed - advances the video display down by one line. Will scroll if at the bottom line of the window.

(11) Set character color - sets the character color. Used from BASIC by;

```
PRINT CHR$(11);CHR$(x);
```

where "x" is the desired color. From this point on, anything you print will be printed in this color (provided the color is enabled).

(12) Clear window - clears only the present window without affecting the rest of the video display. Also, homes the cursor in the present window.

(13) Carriage Return - positions the cursor at the front of the present line, but does not print the cursor. This is useful in some graphics applications where you do not want the cursor showing on the screen.

(14) Define as Home - uses the present cursor position as the "home" position or the upper left hand corner of the present window.

(15) Set lower right hand corner of window - to use this command, position the cursor and print the command. Using this command in conjunction with the Define Home command allows the programmer to easily define a window anywhere on the 540 display. The 2 command codes when used together define a box (window) giving starting and ending address and the line length. Remember that all cursor movement is relative to the present "home" position.

(16) Define Window - The video system allows you to save up to 6 windows for instant recall. To use this from BASIC, you;

```
PRINT CHR$(16);CHR$(x);
```

where "x" is a number between 0 and 5. This saves all current window parameters (starting line, ending line, color, and line length) in a table for later recall. Window 0 is already defined to be the entire 540 video display and window 5 is used internally by the set window to color command. You may use window 0 for your own use, but you should know that once that window's parameters are changed you have no way to access parts of the video screen outside of this "master window" (command code 1). If you are not using color or the set color controls, then you may also use window 5. If you are using color, don't use window 5.

(17) Set Window - This is the command that allows you to recall saved window parameters. To recall a window from BASIC, you;

```
PRINT CHR$(17);CHR$(x);
```

where "x" is a number between 0 and 5. This will set the window to the saved parameters and home the cursor in that window.

(18) Video Control - This command code is used to control the video board's color and sound. To use this command from BASIC, you;

```
PRINT CHR$(18);CHR$(x);
```

where "x" is the desired function number. Refer to the manual that came with your system for the desired function number. This command also stores the last command function entered at $259E (decimal 9630) so that the present video/sound/color attributes can be read. For proper operation of the 540 Video Routine, you should no longer POKE the color/sound/video control function, but use this command instead.

(19) Output Character - This command allows you to print any of the graphics characters, including control characters. To use this command from BASIC, you;

```
PRINT CHR$(19);CHR$(x);
```

where "x" is the ASCII value from 0 to 255 of the character you wish to print.

(20) Direct Cursor Position - This command is used to position the cursor anywhere within the present window. It is used by;

```
PRINT CHR$(20);CHR$(x);CHR$(y);
```

where "x" is the desired column and "y" is the desired row. This routine does range checking and will now allow the cursor to move outside of the presently defined window. All movement is relative to the "home" position.

(21) Cursor Up - This command moves the cursor non-destructively up by one line.

(22) - (23) Unused.

(24) Cursor Right - This command moves the cursor non-destructively 1 position to the right.

(25)-(28) Unused.

(29) Home - Homes the cursor in the present window.

(30) Clear the rest of line - clear from present cursor position to the end of the line without affecting the cursor position.

(31) Clear rest of window - clears from the present cursor position to the bottom of the window without affecting the cursor position.

Installation

The first step is to make a new OS-65D (version 3.2 or earlier) diskette. On that disk create three files: a two-track file named "BEXEC*", a one-track file named "VIDEO*", and a large file (10 tracks for 8", 15 for mini's) named "VIDASM". Write down the track number where the file "VIDEO*" resides on your disk. You'll need it later.

Boot the Assembler/Editor and enter the assembly language program given in Listing 2. Change the origin address on line *730 to reflect your system's memory size. On 24K systems, it should remain at $5B00, on 32K systems set it to $7B00, and on 48K systems use $BB00. Save this program in the file named "VIDASM". Use the "H" command in the assembler to protect the high end of memory (ie. "H5A00", "H7A00", or "HBA00") and assemble the file to memory with the command "A3". If the assembly proceeds without error, save the machine code to disk with the command;

```
!SA tt,1=xB00/5
```

where "tt" is the track number where "VIDEO*" resides and "xB00" is the origin address of the code (ie. "5B00", "7B00", or "BB00").

Now, leave the Assembler/Editor and invoke BASIC. Enter the BEXEC* program given in Listing 1. Note that you'll have to also insert the track number for "VIDEO*" in lines 10280 through 10300 as you did in the above command. Finally, save it in the file named "BEXEC*" (clever, eh?). Run this program and the new video driver will be installed and ready for use. When you want to install the video driver on other diskettes, just transfer the files "BEXEC*" and "VIDEO*".

Programming Tips

Most of the command codes are easy to understand and use. However, several things need to be pointed out. Defining and setting windows is very easy once you understand the step by step procedure.

First, set the video parameters to the master window using command code 1. Using direct cursor positioning, CHR$(20), move the cursor to your desired "home" location. Then print the Define As Home command, CHR$(14). Position the cursor to the desired lower right hand corner position of the window you wish to define. Remember that all cursor positioning is relative to the current "home" position. Thus, if you want your new window to be 10 lines by 20 characters long, print the cursor position command followed by the width and height you want and finally print the Set Lower Right Hand Corner Command as in;

```
PRINT CHR$(20);CHR$(10);CHR$(15);
```

You have just defined a new window on your video display. Try LISTing a BASIC program, cursor positioning, set window color, etc. and you will see that you can do anything without affecting the rest of the video display.

If you wish to save this window definition for later use, print the Define Window command followed by the number you wish to assign to this window. Refer to the Define Window command (code 16). To recall this window, print the Set Window command (code 17) followed by the window number you chose.

Listing 1

```
10 REM   BEXEC* : BASIC EXECUTIVE
12 REM   OS65D V3.2
15 REM   LAST MODIFIED: 07/02/86 BY RICHARD L. TRETHEWEY
16 REM   WRITTEN BY SHOF BEAVERS  01/06/81
17 REM
18 REM   SOFTWARE CONSULTANTS
19 REM      7053 ROSE TRAIL
20 REM     MEMPHIS, TN 38134
21 REM      (901)  377-3503
22 REM
24 REM   SET UP INFLAG AND OUFLAG FROM DEFALT
25 X = PEEK(10950): POKE 8993,X: POKE 8994,X
26 IF PEEK(57088)=223 THEN POKE 9794,37
30 GOSUB 10000: PRINT CHR$(12);: END
10000 REM   ROUTINE TO UNLOCK AND MODIFY BASIC OS
10010 REM
10020 REM   ENABLE <CTRL>'C'
10030 POKE 2073,173
10040 REM  ALLOW NULL INPUT TO STRINGS AND NUMERICS
10050 POKE 2888,0: POKE 8722,0
10060 REM   CHANGE "REDO FROM START?" MESSAGE
10070 REM   TO "MUST BE NUMERIC?"
10080 FOR I = 3129 TO 3143: READ U: POKE I,U: NEXT I
10090 DATA 77,85,83,84,32,66,69,32
10100 DATA 78,85,77,69,82,73,67
10110 REM   ALLOW COMMA AND COLON IN INPUTS
10120 POKE 2972,13: POKE 2976,13
10130 REM   ALLOW "NEW" AND "LIST"
10140 POKE 741,76: POKE 750,78
10150 REM   DELETE "?" INPUT STATEMENT PROMPT
10160 FOR I = 2895 TO 2898: POKE I,234: NEXT I
10170 POKE 2899,160: POKE 2900,0
10180 POKE 2948,234: POKE 2949,234: POKE 2950,234
10190 REM   KILL AUTO CR/LF FROM PRINT
10200 POKE 2813,234: POKE 2814,234: POKE 2815,234
10210 POKE 2658,234: POKE 2659,234: POKE 2660,234
10220 POKE 23,63: POKE 24,49
10230 REM   CHANGE INDIRECT FILE LOAD COMMAND TO <CTRL>'Z'
10240 POKE 9594,26: POKE 9554,110: POKE 9368,110: REM  MOVE TO $6E00
10250 X=PEEK(8960): IF X=>95 THEN TA=90
10260 IF X=>127 THEN TA=122
10270 IF X=>191 THEN TA=186
10280 IF TA=90 THEN DISK!"CA 5B00=TT,1"
10290 IF TA=122 THEN DISK!"CA 7B00=TT,1"
10300 IF TA=186 THEN DISK!"CA BB00=TT,1"
10310 POKE 9628,32: REM  CLEAR CHARACTER
10320 POKE 9629,15: REM  COLOR CHARACTER
10330 POKE 9630,1 : REM  SET VIDEO TO 64 CHAR/LINE
10340 POKE 56900,1: POKE 9643,32: POKE 9646,0: POKE 9647,0
10350 POKE 9645,161: REM  CURSOR CHARACTER
10360 REM   POINT OS-65D TO NEW VIDEO DRIVER
10370 POKE 8979,255: POKE 8980,TA
10380 REM   PROTECT VIDEO DRIVER FROM BASIC
10390 POKE 132,255: POKE 133,TA: POKE 8960,TA
10400 PRINT CHR$(1);CHR$(4);: RETURN
```

```
                        .PAGE ' 540 VIDEO ROUTINE WITH COLOR '
                ; ---------------------------------
                ; 540 VIDEO DRIVER WITH
                ; COLOR CONTROLS FOR OS65D V 3.X
                ; REVISION 1.1
                ;
                ; WRITTEN BY SHOF BEAVERS
                ; SOFTWARE CONSULTANTS
                ; 7053 ROSE TRAIL
                ; MEMPHIS, TN. 38134
                ; (901)-377-3503
                ; ---------------------------------
                ;
                ; ZERO PAGE USED
                ; --------------
                ;
0050                    *=$0050
0050=                   ZPAGE=*
0050=                   CLAL=*
0051=                   CLAH=*+1
                ;
                ; CONSTANTS
                ; ---------
                ;
0014=                   CPOS=$14        ; CURSOR POSITION CHARACTER
0005=                   PLOTV=$05       ; VERTICAL PLOT CHARACTER
0006=                   PLOTH=$06       ; HORIZONTAL PLOT CHARACTER
0010=                   DEFW=$10        ; DEFINE WINDOW CHARACTER
0011=                   SETW=$11        ; SET WINDOW CHARACTER
0012=                   VCNTRL=18       ; VIDEO CONTROL CODE
0009=                   BCOLOR=9        ; BACKGROUND COLOR CODE
000B=                   FCOLOR=11       ; CHARACTER COLOR CODE
0013=                   CHROUT=19       ; OUTPUT CHARACTER (X)
0006=                   MAXWIN=6        ; MAXIMUM NUMBER OF WINDOWS-1
                ;
                ; SYSTEM ADDRESSES AND SUBROUTINES
                ; --------------------------------
                ;
DF00=                   KPORT=$DF00     ; POLLED KEYBOARD PORT
DE44=                   VSIZE=$DE44     ; VIDEO CONTROL (32/64)
                        .PAGE
                ;
                ; OTHER ADDRESSES USED BY VIDEO DRIVER
                ; $2599 UP TO $2643 USED BY STANDARD VIDEO
                ; ----------------------------------------
                ;
2599=                   STOR1=$2599     ; STORAGE FOR ZERO PAGE
259B=                   CNTRLC=$259B    ; CONTROL CHARACTER SAVE
259C=                   CLEARC=$259C    ; CLEAR SCREEN/WINDOW CHARACTER
259D=                   COLORC=$259D    ; COLOR CHARACTER
259E=                   VREG=$259E      ; R/W VIDEO REGISTER
259F=                   WCOLOR=$259F    ; WINDOW COLOR
25A4=                   VPARM=$25A4     ; VIDEO PARAMETERS SAVE
25A4=                   CRLINE=VPARM    ; CURRENT LINE
25A4                    *=VPARM
25A6=                   HAL=*+2         ; HOME ADDRESS LOW
25A7=                   HAH=*+3         ; HOME ADDRESS HIGH
25A8=                   ELAL=*+4        ; ENDING ADDRESS LOW
25A9=                   ELAH=*+5        ; ENDING ADDRESS HIGH
25AA=                   LEN=*+6         ; LINE LENGTH
25AB=                   CSAV=*+7        ; CHARACTER UNDER CURSOR
25AC=                   CURSOR=*+8      ; CURSOR POSITION IN LINE
25AD=                   CCHAR=*+9       ; CURSOR CHARACTER
25AE=                   TEMP=*+10       ; TEMPORARY
25AF=                   CCOUNT=*+11     ; COUNT FOR GET PARM
25B0=                   COLM=*+12       ; COLUMN FOR XY POSITIONING
25B1=                   ROW=*+13        ; ROW FOR XY POSITIONING
25B0=                   CHAR1=COLM      ; FIRST CHARACTER FROM GET PARM
25B1=                   CHAR2=ROW       ; SECOND CHARACTER FROM GET PARM
                ;
                ; THERE IS NOW OPEN MEMORY FROM $25B2 TO $2643
                ;
5B00                    *=$5B00
                ;
                ; START OF VIDEO DRIVER
                ; ---------------------
                ;
5B00 8DAE25   WRITE   STA TEMP        ;VIDEO OUTPUT ROUTINE
5B03 A002             LDY #$02        ;SWAP OUT 2 BYTES FROM ZERO PAGE
5B05 B94F00   SWAPIN  LDA ZPAGE-1,Y
5B08 999825           STA STOR1-1,Y
```

```
 820 5B0B B9A325          LDA VPARM-1,Y
 830 5B0E 994F00          STA ZPAGE-1,Y
 840 5B11 88              DEY
 850 5B12 D0F1            BNE SWAPIN
 860 5B14 20295B          JSR WRT        ;USE THE OUTPUT BYTE
 870 5B17 A002            LDY #$02       ;RESTORE ZERO PAGE
 880 5B19 B94F00  SWAPOT  LDA ZPAGE-1,Y
 890 5B1C 99A325          STA VPARM-1,Y
 900 5B1F B99825          LDA STOR1-1,Y
 910 5B22 994F00          STA ZPAGE-1,Y
 920 5B25 88              DEY
 930 5B26 D0F1            BNE SWAPOT
 940 5B28 60              RTS
 950                ;
 960 5B29 ADAF25  WRT     LDA CCOUNT     ;IS THE BYTE A PARAMETER
 970 5B2C D06C            BNE GPARM      ;YES, SAVE IT
 980 5B2E ADAE25          LDA TEMP
 990 5B31 F049            BEQ RETURN     ;IF NULL GO BACK
1000 5B33 C920            CMP #$20       ;IS IT A CONTROL CODE
1010 5B35 9053            BCC CNTL       ;YES, DO IT
1020                ;
1030 5B37 ACAC25  DISPLY LDY CURSOR      ;GET INDEX INTO LINE
1040 5B3A 9150           STA (CLAL),Y   ;OUTPUT THE CHARACTER
1050 5B3C 20605D         JSR COLADJ     ;CHANGE $DX TO $EX
1060 5B3F AD9D25         LDA COLORC     ;GET COLOR
1070 5B42 9150           STA (CLAL),Y   ;OUTPUT TO COLOR MEMORY
1080 5B44 20605D         JSR COLADJ     ;CHANGE $EX TO $DX
1090 5B47 C8             INY            ;BUMP THE INDEX
1100 5B48 CCAA25         CPY LEN        ;END OF LINE
1110 5B4B D008           BNE BACK       ;NO, GO BACK
1120 5B4D A000           LDY #$00       ;SET INDEX INTO LINE=0
1130 5B4F 8CAC25         STY CURSOR
1140 5B52 4CCF5B         JMP LF         ;DO LINE FEED
1150 5B55 8CAC25  BACK   STY CURSOR     ;SAVE THE INDEX
1160                ;
1170                ; CHECK FOR CNTRL S AND CNTRL Q
1180                ; ------------------------------
1190                ;
1200 5B58 A901           LDA #$1        ;CHECK FOR THE 'CNTRL' KEY
1210 5B5A 8D00DF         STA KPORT      ;LATCH THE PORT
1220 5B5D AD00DF         LDA KPORT      ;READ THE CHARACTER
1230 5B60 C941           CMP #$41       ;IS IT THE CONTROL
1240 5B62 D018           BNE RETURN     ;NO, GO BACK
1250 5B64 A908           LDA #$8        ;YES, CHECK FOR 'S'
1260 5B66 8D00DF         STA KPORT      ;LATCH THE PORT
1270 5B69 AD00DF         LDA KPORT      ;READ THE KEYBOARD
1280 5B6C C980           CMP #$80       ;IS IT THE 'S' KEY
1290 5B6E D00C           BNE RETURN     ;NO, GO BACK
1300 5B70 A902           LDA #$2        ;CHECK FOR THE 'Q'
1310 5B72 8D00DF         STA KPORT      ;LATCH THE PORT
1320 5B75 AD00DF  STOP   LDA KPORT      ;READ THE KEYBOARD
1330 5B78 C980           CMP #$80       ;IS IT THE 'Q'
1340 5B7A D0F9           BNE STOP       ;NO, KEEP LOOPING
1350                ;
1360                ; RETURN: NORMAL EXIT, OUTPUTS CURSOR
1370                ; -----------------------------------
1380                ;
1390 5B7C ACAC25  RETURN LDY CURSOR     ;GET INDEX TO LINE
1400 5B7F B150           LDA (CLAL),Y   ;GET CHARACTER UNDER CURSOR
1410 5B81 8DAB25         STA CSAV       ;SAVE THE CHARACTER
1420 5B84 ADAD25         LDA CCHAR      ;GET THE CURSOR CHARACTER
1430 5B87 9150           STA (CLAL),Y   ;OUTPUT IT
1440 5B89 60             RTS            ;GO BACK FROM OUTPUT ROUTINE
1450                ;
1460                ; CNTL: GET CONTROL CODE ROUTINE ADDRESS FROM
1470                ; TABLE AND EXECUTE
1480                ; ------------------------------------------
1490                ;
1500 5B8A 8D9B25  CNTL   STA CNTRLC     ;SAVE THE CONTROL CODE
1510 5B8D 0A             ASL A
1520 5B8E A8             TAY            ;SET TO INDEX TABLE
1530 5B8F 88             DEY
1540 5B90 88             DEY
1550 5B91 B9F45E         LDA CNTLTB+1,Y ;GET HIGH BYTE
1560 5B94 48             PHA            ;PUSH ON STACK
1570 5B95 B9F35E         LDA CNTLTB,Y   ;GET LOW BYTE
1580 5B98 48             PHA            ;PUSH
1590 5B99 60             RTS            ;EXECUTE THE ROUTINE
1600                ;
1610                ; GPARM: GET PARAMETERS FOR CURSOR POSITIONING,WINDOWS
1620                ; ----------------------------------------------------
```

```
1630                ;
1640 5B9A ADAF25  GPARM  LDA CCOUNT     ;GET CHARACTER COUNT
1650 5B9D C901           CMP #$01      ;IS THIS THE SECOND CHARACTER
1660 5B9F F00A           BEQ SPARM     ;YES
1670 5BA1 ADAE25         LDA TEMP      ;GET THE BYTE
1680 5BA4 8DB025         STA CHAR1     ;SAVE IT AT CHAR1 (COLM)
1690 5BA7 CEAF25         DEC CCOUNT    ;ADJUST COUNT
1700 5BAA 60             RTS
1710 5BAB ADAE25  SPARM  LDA TEMP      ;GET THE BYTE
1720 5BAE 8DB125         STA CHAR2     ;SAVE IT (ROW)
1730 5BB1 CEAF25         DEC CCOUNT    ;SET CCOUNT = 0
1740 5BB4 4C675D         JMP WHICH1    ;GO DO THE CONTROL FUNCTION
1750                ;
1760                ; BSPACE: BACKSPACE ROUTINE
1770                ; ------------------------
1780                ;
1790 5BB7 20125D  BSPACE JSR DELCUR    ;DELETE CURSOR
1800 5BBA ACAC25         LDY CURSOR    ;GET CURSOR POSITION
1810 5BBD F0BD           BEQ RETURN    ;IF AT FRONT OF LINE RETURN
1820 5BBF 88             DEY           ;DECREMENT INDEX
1830 5BC0 8CAC25         STY CURSOR    ;AND SAVE
1840 5BC3 4C7C5B         JMP RETURN    ;GO BACK AND PRINT NEW CURSOR
1850                ;
1860                ; CR: CARRIAGE RETURN ROUTINE
1870                ; ---------------------------
1880                ;
1890 5BC6 20125D  CR     JSR DELCUR    ;DELETE CURSOR
1900 5BC9 A000           LDY #$00      ;RESET INDEX
1910 5BCB 8CAC25         STY CURSOR
1920 5BCE 60             RTS
1930                ;
1940                ; LF: LINE FEED ROUTINE ( SCROLLS IF NEEDED )
1950                ; -------------------------------------------
1960                ;
1970 5BCF 20125D  LF     JSR DELCUR    ;DELETE CURSOR
1980 5BD2 20F65C         JSR INCL      ;INCREMENT THE LINE COUNT
1990 5BD5 90A5           BCC RETURN    ;IF NOT AT END RETURN
2000 5BD7 ADA625  SCROLL LDA HAL       ;LAST LINE, DO SCROLL
2010 5BDA 8550           STA CLAL      ;RESET CURRENT LINE ADDRESS
2020 5BDC ADA725         LDA HAH
2030 5BDF 8551           STA CLAH
2040 5BE1 ADAA25  LINE   LDA LEN       ;A= LINE LENGTH
2050 5BE4 8DAE25         STA TEMP      ;SET TEMP TO LINE LENGTH
2060 5BE7 18             CLC           ;SET TO ADD
2070 5BE8 6940    OFFSET ADC #$40      ;A=LINE LENGTH + $40
2080 5BEA AA             TAX
2090 5BEB CEAE25  COPY   DEC TEMP      ;DECREMENT LINE COUNT
2100 5BEE CA             DEX           ;DECREMENT INDEX INTO LINE
2110 5BEF 8A             TXA           ;MOVE X TO Y THROUGH A
2120 5BF0 A8             TAY
2130 5BF1 B150           LDA (CLAL),Y ;GET CHARACTER FROM LINE + $40
2140 5BF3 ACAE25         LDY TEMP      ;GET LINE INDEX
2150 5BF6 9150           STA (CLAL),Y ;STORE CHARACTER (MOVE BY $40)
2160 5BF8 D0F1           BNE COPY      ;NOT DONE WITH THIS LINE SO LOOP
2170 5BFA 20F65C         JSR INCL      ;INCREMENT CURRENT LINE
2180 5BFD 90E2           BCC LINE      ;NOT DONE YET SO LOOP BACK
2190 5BFF ACAA25         LDY LEN       ;RESET INDEX TO LINE
2200 5C02 A920           LDA #$20      ;SET A TO CLEAR LAST LINE
2210 5C04 88      SPLOOP DEY           ;DECREMENT INDEX
2220 5C05 9150           STA (CLAL),Y ;OUTPUT THE SPACE
2230 5C07 D0FB           BNE SPLOOP    ;NOT DONE KEEP LOOPING
2240 5C09 4C7C5B         JMP RETURN    ;GO BACK AND PRINT CURSOR
2250                ;
2260                ; HOME: HOME CURSOR IN WINDOW
2270                ; ---------------------------
2280                ;
2290 5C0C 20C65B  HOME   JSR CR        ;DO CARRIAGE RETURN
2300 5C0F ADA625         LDA HAL       ;RESET CURRENT LINE TO HOME LINE
2310 5C12 8550           STA CLAL
2320 5C14 ADA725         LDA HAH
2330 5C17 8551           STA CLAH
2340 5C19 60             RTS
2350                ;
2360                ; CLEAR: CLEAR PRESENT WINDOW AND HOME CURSOR
2370                ; --------------------------------------------
2380                ;
2390 5C1A 200C5C  CLEAR  JSR HOME      ;HOME CURSOR
2400 5C1D AD9C25  NXTLIN LDA CLEARC    ;GET CLEAR CHARACTER
2410 5C20 9150    NXTSP  STA (CLAL),Y ;AND OUTPUT IT
2420 5C22 C8             INY           ;BUMP THE INDEX
2430 5C23 CCAA25         CPY LEN       ;Y=LINE LENGTH?
```

```
2440 5C26 D0F8           BNE NXTSP     ;NO LOOP BACK
2450 5C28 A000           LDY #$00      ;YES, RESET Y
2460 5C2A 20F65C         JSR INCL      ;INCREMENT THE CURRENT LINE
2470 5C2D 90EE           BCC NXTLIN    ;NOT DONE, LOOP BACK
2480 5C2F 8CAC25         STY CURSOR    ;RESET INDEX TO LINE
2490 5C32 200C5C         JSR HOME      ;HOME CURSOR
2500 5C35 60      RET    RTS
2510              ;
2520              ; FORWRD: MOVE CURSOR RIGHT 1 POSITION
2530              ; ------------------------------------
2540              ;
2550 5C36 20125D  FORWRD JSR DELCUR     ;DELETE CURSOR
2560 5C39 C8             INY            ;BUMP LINE INDEX
2570 5C3A CCAA25         CPY LEN        ;AT END OF LINE
2580 5C3D B003           BCS RET3       ;YES, GO BACK
2590 5C3F 8CAC25         STY CURSOR     ;SAVE NEW INDEX
2600 5C42 4C7C5B  RET3   JMP RETURN     ;GO BACK AND PRINT CURSOR
2610              ;
2620              ; CSCRN: CLEAR 540 VIDEO DISPLAY
2630              ; -------------------------------
2640              ;
2650 5C45 A000    CSCRN  LDY #$00       ;SET INDEX TO 0
2660 5C47 AD9C25         LDA CLEARC     ;GET CLEAR CHARACTER
2670 5C4A 9900D7  CSLOP  STA $D700,Y    ;OUTPUT IT TO ALL LINES
2680 5C4D 9900D6         STA $D600,Y
2690 5C50 9900D5         STA $D500,Y
2700 5C53 9900D4         STA $D400,Y
2710 5C56 9900D3         STA $D300,Y
2720 5C59 9900D2         STA $D200,Y
2730 5C5C 9900D1         STA $D100,Y
2740 5C5F 9900D0         STA $D000,Y
2750 5C62 C8             INY            ;BUMP THE INDEX
2760 5C63 D0E5           BNE CSLOP      ;LOOP IF NOT DONE
2770 5C65 60             RTS
2780              ;
2790              ; CLINE: CLEAR REST OF LINE
2800              ; --------------------------
2810              ;
2820 5C66 ACAC25  CLINE  LDY CURSOR     ;GET INDEX IN LINE
2830 5C69 A920           LDA #$20       ;A='SPACE'
2840 5C6B 9150    CLOOP  STA (CLAL),Y   ;OUTPUT SPACE
2850 5C6D C8             INY            ;BUMP THE INDEX
2860 5C6E CCAA25         CPY LEN        ;AT END OF LINE?
2870 5C71 D0F8           BNE CLOOP      ;NO, KEEP LOOPING
2880 5C73 4C7C5B         JMP RETURN     ;GO BACK AND PRINT CURSOR
2890              ;
2900              ; CURPOS,PLOT: SAVE CONTROL CODE AND
2910              ; SET CCOUNT FOR 2 PARAMETERS
2920              ; ----------------------------------
2930              ;
2940              PLOT
2950 5C76 A902    CURPOS LDA #$02       ;SET CCOUNT TO 2 FOR GETPARM
2960 5C78 8DAF25         STA CCOUNT
2970 5C7B 60             RTS
2980              ;
2990              ; CRWIN: CLEAR REST OF WINDOW
3000              ; ----------------------------
3010              ;
3020 5C7C ACAC25  CRWIN  LDY CURSOR     ;GET INDEX IN LINE
3030 5C7F 8CB025         STY COLM       ;SAVE CURRENT LINE PARAMETERS
3040 5C82 A550           LDA CLAL
3050 5C84 8DB125         STA ROW
3060 5C87 A551           LDA CLAH
3070 5C89 8DAE25         STA TEMP
3080 5C8C 201D5C         JSR NXTLIN    ;JUMP TO MIDDLE OF CLEAR WINDOW
3090 5C8F ADB025         LDA COLM      ;RESTORE LINE PARAMETERS
3100 5C92 8DAC25         STA CURSOR
3110 5C95 ADAE25         LDA TEMP
3120 5C98 8551           STA CLAH
3130 5C9A ADB125         LDA ROW
3140 5C9D 8550           STA CLAL
3150 5C9F 4C7C5B         JMP RETURN    ;GO BACK AND PRINT CURSOR
3160              ;
3170              ; SET64: SET VIDEO TO 64 CHARACTERS PER LINE
3180              ; --------------------------------------------
3190              ;
3200 5CA2 AD9E25  SET64  LDA VREG      ;SET 540 FOR 64 CHARACTER/LINE
3210 5CA5 0901           ORA #$01
3220 5CA7 8D44DE         STA VSIZE
3230 5CAA 8D9E25         STA VREG
3240 5CAD A940           LDA #$40      ;SET LINE LENGTH
```

```
3250 5CAF 8DAA25          STA LEN
3260 5CB2 60              RTS
3270              ;
3280              ; SET32: SET TO 32 CHARACTERS/LINE
3290              ; --------------------------------
3300              ;
3310 5CB3 AD9E25  SET32   LDA VREG    ;SET TO 32 CHARACTERS PER LINE
3320 5CB6 29FE            AND #$FE    ;TURN OFF 1 BIT
3330 5CB8 8D44DE          STA VSIZE
3340 5CBB 8D9E25          STA VREG
3350 5CBE A920            LDA #$20    ;SET LINE LENGTH
3360 5CC0 8DAA25          STA LEN
3370 5CC3 60              RTS
3380              ;
3390              ; CUP: MOVE CURSOR UP
3400              ; --------------------
3410              ;
3420 5CC4 20125D  CUP     JSR DELCUR  ;DELETE CURSOR
3430 5CC7 20445D          JSR SUB     ;SUBTRACT $40 FROM CL ADDRESS
3440 5CCA 4C7C5B          JMP RETURN  ;GO BACK AND PRINT CURSOR
3450              ;
3460              ; DHOME: DEFINE PRESENT CURSOR POSITON AS HOME
3470              ; ---------------------------------------------
3480              ;
3490 5CCD A550    DHOME   LDA CLAL    ;SET HOME TO CURRENT LINE
3500 5CCF 18              CLC
3510 5CD0 6DAC25          ADC CURSOR  ;ADD LINE INDEX
3520 5CD3 8DA625          STA HAL     ;AND SAVE LOW BYTE
3530 5CD6 A551            LDA CLAH    ;SET HIGH BYTE
3540 5CD8 8DA725          STA HAH
3550 5CDB 200C5C          JSR HOME    ;HOME CURSOR TO SET PARAMETERS
3560 5CDE 60              RTS
3570              ;
3580              ; DLRCW: SET LOWER RIGHT CORNER OF WINDOW
3590              ; ----------------------------------------
3600              ;
3610 5CDF ADAC25  DLRCW   LDA CURSOR  ;GET INDEX IN LINE
3620 5CE2 8DAA25          STA LEN     ;PRESENT INDEX=NEW LINE LENGTH
3630 5CE5 A550            LDA CLAL    ;CURRENT LINE=NEW LAST LINE
3640 5CE7 8DA825          STA ELAL
3650 5CEA A551            LDA CLAH
3660 5CEC 8DA925          STA ELAH
3670 5CEF 60              RTS
3680              ;
3690              ; WINDOW: SET PARAMETERS FOR WINDOW CONTROLS
3700              ; ALSO USED FOR ANY COMMAND WITH 1 PARAMETER
3710              ; -------------------------------------------
3720              ;
3730 5CF0 A901    WINDOW  LDA #$01    ;SET GET PARM FOR 1 PARAMETER
3740 5CF2 8DAF25          STA CCOUNT
3750 5CF5 60              RTS
3760              ;
3770              ; INCL: INCREMENT CURRENT LINE. CARRY SET IF AT LAST
3780              ; ---------------------------------------------------
3790              ;
3800 5CF6 18      INCL    CLC         ;GET SET TO ADD
3810 5CF7 A551            LDA CLAH
3820 5CF9 CDA925          CMP ELAH    ;LESS THAN ENDING ADDRESS
3830 5CFC 9008            BCC INCH    ;YES, DO INCREMENT
3840 5CFE A550            LDA CLAL    ;CHECK THE LOW BYTES
3850 5D00 CDA825          CMP ELAL
3860 5D03 9001            BCC INCH    ;DO INCREMENT
3870 5D05 60              RTS         ;RETURN WITH CARRY SET
3880 5D06 A550    INCH    LDA CLAL    ;INCREMENT CURRENT LINE
3890 5D08 6940            ADC #$40
3900 5D0A 8550            STA CLAL
3910 5D0C 9003            BCC INCEND
3920 5D0E E651            INC CLAH
3930 5D10 18              CLC
3940 5D11 60      INCEND  RTS
3950              ;
3960              ; DELCUR: DELETE CURSOR, RESTORE CHARACTER UNDER CURSOR
3970              ; -----------------------------------------------------
3980              ;
3990 5D12 ACAC25  DELCUR  LDY CURSOR    ;GET INDEX IN LINE
4000 5D15 B150            LDA (CLAL),Y  ;GET CHARACTER
4010 5D17 CDAD25          CMP CCHAR     ;IS IT THE CURSOR
4020 5D1A D00A            BNE RET2      ;NO, GO BACK
4030 5D1C ADAB25          LDA CSAV      ;YES, GET CHAR UNDER CURSOR
4040 5D1F 9150            STA (CLAL),Y  ;AND RESTORE
4050 5D21 A920            LDA #$20      ;CLEAR CSAV
```

```
4060 5D23 8DAB25          STA CSAV
4070 5D26 60       RET2   RTS
4080              ;
4090              ; MASWIN: SET VIDEO PARAMETERS TO 64 CHAR/LINE
4100              ; WITH 25 LINES TO THE SCREEN
4110              ; TURNS OFF COLOR AND SOUND
4120              ; ----------------------------------------
4130              ;
4140 5D27 A900     MASWIN LDA #$00      ;RESET VIDEO PARAMETERS
4150 5D29 8550            STA CLAL
4160 5D2B 8DA625          STA HAL
4170 5D2E 8DA825          STA ELAL
4180 5D31 8DAC25          STA CURSOR
4190 5D34 A9D1            LDA #$D1
4200 5D36 8551            STA CLAH
4210 5D38 8DA725          STA HAH
4220 5D3B A9D7            LDA #$D7
4230 5D3D 8DA925          STA ELAH
4240 5D40 20A25C          JSR SET64    ;SET FOR 64 CHAR/LINE
4250 5D43 60              RTS
4260              ;
4270              ; SUB: ADJUST PRESENT CURSOR POSITION
4280              ; UP BY 1 LINE
4290              ; -------------------------
4300              ;
4310 5D44 A551     SUB    LDA CLAH     ;CHECK TO SEE IF AT TOP
4320 5D46 CDA725          CMP HAH
4330 5D49 D007            BNE DOIT     ;NO, ADJUST POSITION
4340 5D4B A550            LDA CLAL
4350 5D4D CDA625          CMP HAL
4360 5D50 F00D            BEQ RET4
4370 5D52 A550     DOIT   LDA CLAL
4380 5D54 38              SEC
4390 5D55 E940            SBC #$40
4400 5D57 8550            STA CLAL
4410 5D59 A551            LDA CLAH
4420 5D5B E900            SBC #$00
4430 5D5D 8551            STA CLAH
4440 5D5F 60       RET4   RTS
4450              ;
4460              ; COLADJ: INTERNAL SUBROUTINE TO ADJUST ADDRESS FOR
4470              ; COLOR CONTROLS. CHANGES $DX TO $EX OR $EX TO $DX.
4480              ; ------------------------------------------------
4490              ;
4500 5D60 A551     COLADJ LDA CLAH      ;GET PRESENT ADDRESS
4510 5D62 4930            EOR #$30      ;CHANGE HIGH BYTE
4520 5D64 8551            STA CLAH      ;AND SAVE
4530 5D66 60              RTS
4540              ;
4550              ; WHICH1: DETERMINE WHICH CONTROL CODE
4560              ; SET CCOUNT FOR GET PARM AND EXECUTE
4570              ; THE PROPER ROUTINE
4580              ; ------------------------------------
4590              ;
4600 5D67 AD9B25  WHICH1 LDA CNTRLC
4610 5D6A C914            CMP #CPOS
4620 5D6C D003            BNE W1
4630 5D6E 4CAC5D          JMP POSCUR
4640 5D71 C905    W1      CMP #PLOTV
4650 5D73 D003            BNE W2
4660 5D75 4CD25D          JMP VLINE
4670 5D78 C906    W2      CMP #PLOTH
4680 5D7A D003            BNE W3
4690 5D7C 4CEF5D          JMP HLINE
4700 5D7F C910    W3      CMP #DEFW
4710 5D81 D003            BNE W4
4720 5D83 4C0A5E          JMP DEFW1
4730 5D86 C911    W4      CMP #SETW
4740 5D88 D003            BNE W5
4750 5D8A 4C365E          JMP SETW1
4760 5D8D C909    W5      CMP #BCOLOR
4770 5D8F D003            BNE W6
4780 5D91 4C745E          JMP COLOR
4790 5D94 C90B    W6      CMP #FCOLOR
4800 5D96 D003            BNE W7
4810 5D98 4CB55E          JMP SCOLOR
4820 5D9B C913    W7      CMP #CHROUT
4830 5D9D D003            BNE W8
4840 5D9F 4CBC5E          JMP COUT
4850 5DA2 C912    W8      CMP #VCNTRL
4860 5DA4 D003            BNE W9
```

```
4870 5DA6 4CC55E          JMP SETVID
4880 5DA9 4C7C5B  W9      JMP RETURN
4890              ;
4900              ; POSCUR: DIRECT CURSOR POSITIONING
4910              ; --------------------------------
4920              ;
4930 5DAC 20125D  POSCUR JSR DELCUR     ;DELETE CURSOR
4940 5DAF 20B55D         JSR POS1       ;FIND CURSOR POSITION
4950 5DB2 4C7C5B         JMP RETURN     ;GO BACK AND OUTPUT CURSOR
4960              ;
4970              ; POS1: FIND POSITION ON SCREEN
4980              ; ----------------------------
4990              ;
5000 5DB5 200C5C  POS1   JSR HOME       ;HOME CURSOR POSITION
5010 5DB8 AEB125         LDX ROW        ;GET THE ROW INFORMATION
5020 5DBB F006           BEQ SETCOL     ;IF 0 THEN SET THE COLUMN
5030 5DBD 20F65C  ROWLP  JSR INCL       ;INCREMENT LINE
5040 5DC0 CA             DEX            ;ADJUST ROW COUNT
5050 5DC1 D0FA           BNE ROWLP      ;NOT DONE, KEEP LOOPING
5060 5DC3 ADB025  SETCOL LDA COLM       ;GET COLUMN INFORMATION
5070 5DC6 CDAA25         CMP LEN        ;AT END OF LINE
5080 5DC9 9003           BCC SETC1      ;NO, SET COLUMN
5090 5DCB ADAA25         LDA LEN        ;GET LINE LENGTH
5100 5DCE 8DAC25  SETC1  STA CURSOR     ;SAVE INDEX IN LINE
5110 5DD1 60             RTS
5120              ;
5130              ; VLINE: PLOT VERTICAL LINE
5140              ; --------------------------
5150              ;
5160 5DD2 ACAC25  VLINE  LDY CURSOR     ;GET INDEX IN LINE
5170 5DD5 AEB025         LDX CHAR1      ;GET NUMBER OF BLOCKS
5180 5DD8 8EAF25         STX CCOUNT     ;AND SAVE
5190 5DDB ADB125  VLINE1 LDA CHAR2      ;GET OUTPUT CHARACTER
5200 5DDE 9150           STA (CLAL),Y   ;OUTPUT IT
5210 5DE0 CEAF25         DEC CCOUNT     ;ADJUST COUNT
5220 5DE3 F006           BEQ DPLOT      ;EXIT IF DONE
5230 5DE5 20445D         JSR SUB        ;MOVE UP BY 1 LINE
5240 5DE8 4CDB5D         JMP VLINE1     ;LOOP BACK FOR NEXT CHARACTER
5250 5DEB 8CAC25  DPLOT  STY CURSOR     ;SAVE INDEX IN LINE
5260 5DEE 60             RTS
5270              ;
5280              ; HLINE: PLOT HORIZONTAL LINE
5290              ; ----------------------------
5300              ;
5310 5DEF ACAC25  HLINE  LDY CURSOR     ;GET INDEX IN LINE
5320 5DF2 AEB025         LDX CHAR1      ;GET NUMBER OF BLOCKS
5330 5DF5 8EAF25         STX CCOUNT     ;AND SAVE
5340 5DF8 ADB125         LDA CHAR2      ;GET OUTPUT CHARACTER
5350 5DFB 9150    HLINE1 STA (CLAL),Y   ;OUTPUT IT
5360 5DFD CCAA25         CPY LEN        ;ARE WE DONE
5370 5E00 F0E9           BEQ DPLOT      ;YES, GO BACK
5380 5E02 C8             INY            ;BUMP THE INDEX
5390 5E03 CEAF25         DEC CCOUNT     ;DECREMENT THE COUNT
5400 5E06 F0E3           BEQ DPLOT      ;BRANCH IF DONE
5410 5E08 D0F1           BNE HLINE1     ;LOOP BACK FOR NEXT CHARACTER
5420              ;
5430              ; DEFW1: DEFINE WINDOW (X)
5440              ; ------------------------
5450              ;
5460 5E0A ACB125  DEFW1  LDY CHAR2      ;GET WINDOW NUMBER
5470 5E0D C006           CPY #MAXWIN
5480 5E0F B024           BCS RET1
5490 5E11 ADA625         LDA HAL        ;SAVE VIDEO PARMS IN TABLES
5500 5E14 99CF5E         STA HALTB,Y
5510 5E17 ADA725         LDA HAH
5520 5E1A 99D55E         STA HAHTB,Y
5530 5E1D ADA825         LDA ELAL
5540 5E20 99DB5E         STA ELALTB,Y
5550 5E23 ADA925         LDA ELAH
5560 5E26 99E15E         STA ELAHTB,Y
5570 5E29 ADAA25         LDA LEN
5580 5E2C 99E75E         STA LENTB,Y
5590 5E2F AD9F25         LDA WCOLOR
5600 5E32 99ED5E         STA COLRTB,Y
5610 5E35 60      RET1   RTS
5620              ;
5630              ; SETW1: SET WINDOW (X)
5640              ; AND SET COLOR
5650              ; ---------------------
5660              ;
5670 5E36 20455E  SETW1  JSR SETW2     ;SET WINDOW PARAMETERS
```

```
5680 5E39 AD9F25         LDA WCOLOR    ;NOW SET COLOR
5690 5E3C 8DB125         STA CHAR2
5700 5E3F 20745E         JSR COLOR
5710 5E42 4C7C5B         JMP RETURN
5720              ;
5730              ; SETW2: SET WINDOW PARAMETERS
5740              ; ---------------------------
5750 5E45 ACB125  SETW2  LDY CHAR2     ;GET WINDOW NUMBER
5760 5E48 C006           CPY #MAXWIN
5770 5E4A B0E9           BCS RET1
5780 5E4C B9DB5E         LDA ELALTB,Y ;LOAD VIDEO PARMS FROM TABLES
5790 5E4F 8DA825         STA ELAL
5800 5E52 B9E15E         LDA ELAHTB,Y
5810 5E55 8DA925         STA ELAH
5820 5E58 B9CF5E         LDA HALTB,Y
5830 5E5B 8DA625         STA HAL
5840 5E5E B9D55E         LDA HAHTB,Y
5850 5E61 8DA725         STA HAH
5860 5E64 B9E75E         LDA LENTB,Y
5870 5E67 8DAA25         STA LEN
5880 5E6A B9ED5E         LDA COLRTB,Y
5890 5E6D 8D9F25         STA WCOLOR
5900 5E70 200C5C         JSR HOME
5910 5E73 60             RTS
5920              ;
5930              ; COLOR: SET WINDOW TO COLOR (X)
5940              ; ------------------------------
5950              ;
5960 5E74 AD9C25  COLOR  LDA CLEARC     ;SAVE CLEAR CHARACTER
5970 5E77 48             PHA
5980 5E78 ADB125         LDA CHAR2      ;GET COLOR NUMBER
5990 5E7B 8D9C25         STA CLEARC     ;AND SAVE
6000 5E7E 8D9F25         STA WCOLOR     ;SAVE WINDOW COLOR
6010 5E81 8D9D25         STA COLORC     ;SET CHARACTER COLOR
6020 5E84 A905           LDA #5         ;SAVE CURRENT WINDOW
6030 5E86 8DB125         STA CHAR2
6040 5E89 200A5E         JSR DEFW1
6050 5E8C ADA725         LDA HAH        ;CHANGE $DXXX TO $EXXX
6060 5E8F 4930           EOR #$30
6070 5E91 8DA725         STA HAH
6080 5E94 ADA925         LDA ELAH
6090 5E97 4930           EOR #$30
6100 5E99 8DA925         STA ELAH
6110 5E9C 201A5C         JSR CLEAR      ;OUTPUT THE COLOR
6120 5E9F A905           LDA #5         ;RESTORE WINDOW PARMS
6130 5EA1 8DB125         STA CHAR2
6140 5EA4 20455E         JSR SETW2
6150 5EA7 AD9C25         LDA CLEARC     ;SET WINDOW COLOR REG.
6160 5EAA 8D9F25         STA WCOLOR
6170 5EAD 68             PLA
6180 5EAE 8D9C25         STA CLEARC     ;RESTORE CLEAR CHARACTER
6190 5EB1 200C5C         JSR HOME
6200 5EB4 60             RTS
6210              ;
6220              ; SCOLOR: SET CHARACTER COLOR
6230              ; ---------------------------
6240              ;
6250 5EB5 ADB125  SCOLOR LDA CHAR2      ;GET COLOR NUMBER
6260 5EB8 8D9D25         STA COLORC     ;AND SET
6270 5EBB 60             RTS
6280              ;
6290              ; COUT: OUTPUT CHARACTER (X)
6300              ; ----------------------------
6310              ;
6320 5EBC ADB125  COUT   LDA CHAR2      ;GET THE CHARACTER
6330 5EBF 8DAE25         STA TEMP       ;AND SAVE
6340 5EC2 4C375B         JMP DISPLY     ;OUTPUT THE CHARACTER
6350              ;
6360              ; SETVID: SET VIDEO,COLOR, AND SOUND REGISTER
6370              ; SET VREG SO PRESENT STATUS CAN BE READ
6380              ; ------------------------------------------
6390              ;
6400 5EC5 ADB125  SETVID LDA CHAR2      ;GET THE CONTROL BYTE
6410 5EC8 8D44DE         STA VSIZE      ;OUTPUT TO VIDEO BOARD
6420 5ECB 8D9E25         STA VREG       ;SAVE AT VREG
6430 5ECE 60             RTS
```

```
6440                ;
6450                ; START OF TABLES FOR DEFINED WINDOWS
6460                ; ----------------------------------
6470                ;
6480 5ECF 00        HALTB   .BYTE 00
6490 5ED0 00                .BYTE 0
6500 5ED1 00                .BYTE 0
6510 5ED2 00                .BYTE 0
6520 5ED3 00                .BYTE 0
6530 5ED4 00                .BYTE 0
6540 5ED5 D0        HAHTB   .BYTE $D0
6550 5ED6 00                .BYTE 0
6560 5ED7 00                .BYTE 0
6570 5ED8 00                .BYTE 0
6580 5ED9 00                .BYTE 0
6590 5EDA 00                .BYTE 0
6600 5EDB C0        ELALTB  .BYTE $C0
6610 5EDC 00                .BYTE 0
6620 5EDD 00                .BYTE 0
6630 5EDE 00                .BYTE 0
6640 5EDF 00                .BYTE 0
6650 5EE0 00                .BYTE 0
6660 5EE1 D7        ELAHTB  .BYTE $D7
6670 5EE2 00                .BYTE 0
6680 5EE3 00                .BYTE 0
6690 5EE4 00                .BYTE 0
6700 5EE5 00                .BYTE 0
6710 5EE6 00                .BYTE 0
6720 5EE7 40        LENTB   .BYTE 64
6730 5EE8 00                .BYTE 0
6740 5EE9 00                .BYTE 0
6750 5EEA 00                .BYTE 0
6760 5EEB 00                .BYTE 0
6770 5EEC 00                .BYTE 0
6780 5EED 0F        COLRTB  .BYTE 15
6790 5EEE 00                .BYTE 0
6800 5EEF 00                .BYTE 0
6810 5EF0 00                .BYTE 0
6820 5EF1 00                .BYTE 0
6830 5EF2 00                .BYTE 0
6840                ;
6850                ; CNTLTB: CONTROL CODE TABLE. CONTAINS THE
6860                ; THE ADDRESS OF THE ROUTINE-1. ADDRESS IS
6870                ; PUSHED ON THE STACK AND THEN CALLED BY
6880                ; DOING AN RTS.
6890                ; ---------------------------------------
6900                ;
6910 5EF3 265D      CNTLTB  .WORD MASWIN-1          ;1  SET MASTER WINDOW
6920 5EF5 A15C              .WORD SET64-1           ;2  SET TO 64 CHAR/LINE
6930 5EF7 B25C              .WORD SET32-1           ;3  SET TO 32 CHAR/LINE
6940 5EF9 445C              .WORD CSCRN-1           ;4  CLEAR 540 VIDEO
6950 5EFB 755C              .WORD PLOT-1            ;5  VERTICAL PLOT
6960 5EFD 755C              .WORD PLOT-1            ;6  HORIZONTAL PLOT
6970 5EFF 7B5B              .WORD RETURN-1          ;7  BELL(NOT IMPLEMENTED)
6980 5F01 B65B              .WORD BSPACE-1          ;8  BACKSPACE
6990 5F03 EF5C              .WORD WINDOW-1          ;9  SET WINDOW TO COLOR
7000 5F05 CE5B              .WORD LF-1              ;10 LINE FEED
7010 5F07 EF5C              .WORD WINDOW-1          ;11 SET CHAR COLOR
7020 5F09 195C              .WORD CLEAR-1           ;12 CLEAR WINDOW
7030 5F0B C55B              .WORD CR-1              ;13 CARRIAGE RETURN
7040 5F0D CC5C              .WORD DHOME-1           ;14 DEFINE AS HOME
7050 5F0F DE5C              .WORD DLRCW-1           ;15 SET LOWER R CORNER
7060 5F11 EF5C              .WORD WINDOW-1          ;16 DEFINE WINDOW
7070 5F13 EF5C              .WORD WINDOW-1          ;17 SET WINDOW
7080 5F15 EF5C              .WORD WINDOW-1          ;18 VIDEO CONTROL
7090 5F17 EF5C              .WORD WINDOW-1          ;19 CHARACTER OUT
7100 5F19 755C              .WORD CURPOS-1          ;20 CURSOR POSITION
7110 5F1B C35C              .WORD CUP-1             ;21 CURSOR UP
7120 5F1D 7B5B              .WORD RETURN-1          ;22 (UNUSED)
7130 5F1F 7B5B              .WORD RETURN-1          ;23 (UNUSED)
7140 5F21 355C              .WORD FORWRD-1          ;24 CURSOR RIGHT
7150 5F23 7B5B              .WORD RETURN-1          ;25 (UNUSED)
7160 5F25 7B5B              .WORD RETURN-1          ;26 (UNUSED)
7170 5F27 7B5B              .WORD RETURN-1          ;27 (UNUSED)
7180 5F29 7B5B              .WORD RETURN-1          ;28 (UNUSED)
7190 5F2B 0B5C              .WORD HOME-1            ;29 HOME CURSOR
7200 5F2D 655C              .WORD CLINE-1           ;30 CLEAR REST OF LINE
7210 5F2F 7B5C              .WORD CRWIN-1           ;31 CLEAR REST OF WINDOW
7220                ;
7230 5F31=                  ZZZZ=*
7240                        .END
```

DMS-65D: True Random Access Files for OS-65D V3.3

One of the biggest drawbacks of OS-65D is the way it handles data files in general, and random access data files in particular. If you go by the book, 65D limits you to record sizes that are powers of two in length. That is, 2, 4, 8, 16, 32, 64, 128, or 256. If your data file needs records that are 129 bytes long, 65D forces you to the next larger record size, 256 bytes, thus wasting 127 bytes of disk space between each record. Even worse perhaps, is that fields within records are stored sequentially, forcing the user to read and write the entire record even when manipulating only one field.

8" disk systems have always had the advantage of being able to use OS-65U which allows direct access to each byte on the diskette as well as having simultaneous access to up to 8 different files. In conjunction with this ability, Ohio Scientific developed their OS-DMS series of software. OS-DMS is a much-maligned data base management system that many 65U packages have been based upon. Most of the criticism centers around the application software from OSI, not the structure of the system. While not as sophisticated as much of the data base software for other systems, OS-DMS is a functional file structure that remains the standard for most of the 65U users.

DMS65D is an out and out copy of the OS-DMS file structure and I used it for two reasons. First, I have used OS-DMS heavily and so have a lot of others. Second, it's an easy structure to understand. Let's take a look at that structure;

Imagine a sheet of graph paper. Instead of looking at it as a grid of intersecting lines, look at it as a series of boxes, with the box at the upper left hand corner being box #0 and each box after that being numbered consecutively higher to the bottom of the page. These boxes are our data file with each box holding a single character. The capacity of our data file is equal to the number of boxes on the

```
10 REM- Data File Manager for OS-65D V3.3
20 GOTO1000
30 :
40 REM- Construct Device 6 Current Track String
50 c6=FNa(PEEK(9004)):t6$=RIGHT$(STR$(c6+kh),k2):RETURN
60 :
70 REM- Construct Device 7 Current Track String
80 c7=FNa(PEEK(9012)):t7$=RIGHT$(STR$(c7+kh),k2):RETURN
90 :
100 REM- Get Record #r6 for Device #6
110 i6=bodf+((r6-k1)*rl):st=INT(i6/ts)+st(k6)
120 GOSUB50:IFc6=stTHEN160
130 d6=PEEK(9005):IFd6=k0THEN150
140 DISK!"sa "+t6$+",1=3a7e/"+pg$:POKE9005,k0
150 DISK!"ca 3a7e="+t6$+",1":POKE 9004,FNb(c6)
160 i=i6-((st-c6)*ts)+bs(k6):ih=INT(i/pg):il=i-ih*pg
170 POKEip(k6),il:POKEip(k6)+k1,ih
175 POKEop(k6),il:POKEop(k6)+k1,ih
180 RETURN
190 :
200 REM- Set Device 6 I/O Pointers to Index(6)
210 i=i6+bs(k6)-(FNa(PEEK(9004))-st(k6))*ts
215 ih=INT(i/pg):il=i-ih*pg
220 POKEip(k6),il:POKEip(k6)+k1,ih
225 POKEop(k6),il:POKEop(k6)+1,ih:RETURN
230 :
240 REM- Set Device 7 I/O Pointers to Index(7)
250 i=i7+bs(k7)-(FNa(PEEK(9012))-st(k7))*ts
255 ih=INT(i/pg):il=i-ih*pg
260 POKE9213,il:POKE9214,ih:POKE9238,il:POKE9239,ih:RETURN
270 :
300 REM- Fetch Record from Device #6
310 GOSUB100:FORk=k1TOnf:i6=bodf+((r6-k1)*rl)+i6(k)
330 GOSUB200:INPUT#k6,a$(k):NEXTk:RETURN
340 :
400 REM- Put Record Out to Device #6
410 GOSUB100
420 FORk=k1TOnf:i6=bodf+((r6-k1)*rl)+i6(k):GOSUB200
430 PRINT#k6,a$(k):NEXTk:RETURN
440 :
700 REM- Display Record Contents
710 PRINT#dv," #";TAB(k4);"Field Name";TAB(32);"Contents"
720 PRINT:FORk=k1TOnf
730 PRINT#dv,k;TAB(k4);n$(k);TAB(32);a$(k):NEXTk:PRINT#dv
740 RETURN
750 :
800 REM- Main Menu
810 :
820 PRINT!(28);&(k9,k0);"DMS-65D Data File Manager"
830 PRINT&(k5,k2);"(1) Directory"
840 PRINT&(k5,k3);"(2) Create a DMS-65D Master File"
850 PRINT&(k5,k4);"(3) Edit a DMS-65D Master File"
860 PRINT&(k5,k5);"(4) Print a DMS-65D Master File"
900 PRINT&(k9,kt);"Your Choice ";:INPUTy$:k=VAL(y$):TRAP0
910 PRINT!(28);:IFk=k0THENEND
920 IFk<k1ORk>k4ORk<>INT(k)THEN820
930 ON k GOTO 2000,3000,4000,5000
990 :
1000 k0=0:k1=1:k2=2:k3=3:k4=4:k5=5:k6=6:k7=7:k8=8:k9=9:kt=10
1010 aa=ASC("A"):az=ASC("Z"):a0=ASC("0"):a9=ASC("9"):kh=100
1020 pg=256:hex$="0123456789abcdef":sx=16:tt=32:di=11897
```

sheet of graph paper.

Now, let's define what a random access data file is. A random access data file is a file in which each piece of data within the file is positioned in a defined location. This allows the programmer to immediately "jump" to the Nth piece of data without having to read in N-1 pices of data, as is necessary with sequential files. Most often, but not always, random access data files are composed of groups of related information. These groups are called records. The easiest way to illustrate a record is a mailing list. A typical mailing list entry would contain the following information:

Name, Address, City, State, Zip Code

Each entry within a record is called a field. In this example, each record contains 5 fields. When a random access data file is being created, the programmer defines the maximum number of characters each field will be allowed to hold. This allows him to calculate precisely the size of the each record and thus, the position of each record and each field within the file. For example, if we know that each record is 50 bytes long, multiplying 50 by the number of the record to be manipulated, yields the position of the beginning of that record number. Going back to our sheet of graph paper, the position of a record or field corresponds to the box number we defined earlier. The software used to manipulate the data file maintains a position pointer to the file. The value of this position pointer is called an INDEX. Under DMS65D, or more accurately under OS-65D, a separate pointer is maintained for both input from and output to the data file. In the program presented here, the indeces are stored in the variables "ip(k6)" and "op(k6)".

When creating a data file application, the specifications of the data file must either be incorporated into the application software, or be included in the data file itself. It is apparent that the most efficient method is to incorporate the file specifications into each data file so that the same application software can be used with many different files. However, this

```
1030 POKE2972,13:POKE2976,13:REM- Disable Comma & Colon
1040 DEF FNa(x)=kt*INT(x/sx)+x-INT(x/sx)*sx
1050 DEF FNb(x)= sx*INT(x/kt)+x-INT(x/kt)*kt
1060 ht=FNa(PEEK(11687)):dt=FNa(PEEK(11716)):e=35
1070 DIM Index(k7),bs(k7),be(k7),st(k7),et(k7),cu(k7),df(k7)
1080 DIM ip(k7),op(k7),f$(ht),ut(ht)
1090 bs(k6)=PEEK(8998)+PEEK(8999)*pg:REM- Buffer Start Address
1100 bs(k7)=PEEK(9006)+PEEK(9007)*pg
1110 be(k6)=PEEK(9000)+PEEK(9001)*pg:REM- Buffer End Address
1120 be(k7)=PEEK(9008)+PEEK(9009)*pg
1130 ts=(be(k6)-bs(k6)):pg$=MID$(hex$,ts/pg+k1,k1)
1140 dt$=RIGHT$(STR$(dt+kh),k2)+","
1150 ip(k6)=9132:op(k6)=9155:ip(k7)=9213:op(k7)=9238
1160 GOTO800
1999 :
2000 REM- Directory Printer
2010 GOSUB50000:GOSUB11100
2020 PRINT!(20);TAB(21);"Directory":PRINT
2030 FORk=k0TOht:IFLEN(f$(k))=k0THEN2080
2040 PRINTTAB(x*19);LEFT$(f$(k),k6);
2041 p=k8:IFk>k9THENp=k7
2050 PRINTTAB(x*19+p);ASC(MID$(f$(k),k7,k1));
2051 p=12:IFk>k9THENp=10
2060 PRINTTAB(x*19+p);ASC(RIGHT$(f$(k),k1));
2070 x=x+k1:IFx=k3THENx=k0:PRINT
2080 NEXTk:PRINT:PRINT
2090 INPUT"Press <RETURN> to Continue ";y$
2100 PRINT!(20);:GOTO800
2110 :
3000 REM- Create New DMS-65D Master File
3010 PRINT"DMS-65D Master File Creation Utility":PRINT
3020 GOSUB50000:GOSUB11100
3021 PRINT"File Names may be up to 5 characters long":PRINT
3030 INPUT"Enter the name for this new Master File ";y$
3035 PRINT:IFLEN(y$)>k5THENPRINT"TOO LONG!":PRINT:GOTO3030
3040 FORk=k1TOLEN(y$):c=ASC(MID$(y$,k,k1))
3050 IFc=>ASC("a")ANDc<=ASC("z")THENc=c-tt
3060 f$=f$+CHR$(c):NEXTk
3070 IFLEN(f$)<k5THENf$=f$+" ":GOTO3070
3080 f$=f$+"0":PRINT
3090 FORk=k0TOht:IFf$<>LEFT$(f$(k),k6)THENNEXTk:GOTO3120
3100 PRINT"THE NAME ";CHR$(34);F$;CHR$(34);"IS IN USE"
3110 GOTO59000
3120 PRINT"How many FIELDS did you want in ";f$;
3130 INPUTy$:nf=VAL(y$):IFnf<=k0ORnf<>INT(nf)THEN3120
3140 DIM n$(nf),fl(nf):PRINT
3150 FORk=k1TOnf
3160 PRINT"FIELD #";k:PRINT
3170 INPUT"Enter the FIELD NAME ";n$(k):PRINT
3180 INPUT"Enter the FIELD LENGTH ";fl(k):PRINT
3190 IFfl(k)>71THEN3180
3200 fl(k)=fl(k)+k1:NEXTk
3210 PRINT!(20);"File: ";f$:PRINT
3220 PRINT" #   Field Name";TAB(32);"Field Length":PRINT
3230 FORk=k1TOnf:PRINTMID$(STR$(k),k2);".";TAB(k6);n$(k);
3240 PRINTTAB(36);fl(k)-k1:NEXTk:PRINT
3250 INPUT"Are these alright ";y$:y$=LEFT$(y$+" ",k1)
3260 PRINT:IFy$<>"y"THENRUN
3270 PRINT"How many RECORDS did you want in ";f$;
3280 INPUTy$:PRINT:nr=VAL(y$):IFnr<=k0THEN3270
3290 rl=k0:FORk=k1TOnf:rl=rl+fl(k):NEXTk
```

method also dictates that all of the data files to be used by the application software must store the file specifications in a uniform manner. We have already defined the critical elements of the file specifications; the number of fields in each record, the length of each field, and the number of records the file can hold. On the surface, this would appear to be enough information to use the data file, but that's not the case. We also need to know where the first piece of data has been stored in the file, and how many pieces have been stored in the file. These extra parts of the file specification are incorporated into two numbers; the beginning of the data file and the end of the data file. In DMS65D, all of this information is stored at the front of the file in an area called the "header". The following table illustrates the contents of the header:

INDEX	DESCRIPTION
0	File Name. Allows double-checking for proper file being opened.
6	File Type. Allows file typing for key files.
9	EODF - Index to End of Data File.
20	BODF - Index to Beginning of Data File.
31	RL - Record Length.
42	NR - Number of Records allowed in file.
53	Start of storage of Field Names and Field Lengths.

BODF will be the first free byte after the last field name/field length entry. When the software first opens the data file, it reads in the values of "eodf", "bodf", "rl", and "nr". The following calculation determines how many records have been stored in the file:

tn = int((eodf-bodf)/rl)

where "tn" equals the total number of records. Following that, a counter is initialized to zero and a field name/field length pair is read. After each pair is read, the counter is incremented by one and the current input pointer (or index) is checked. The program continues to read in field name/field length pairs until the

```
3300 REM- Compute Header Length
3310 l=53:FORk=k1TOnf:l=l+LEN(n$(k))+k1
3320 l=l+LEN(STR$(fl(k)))+k1:NEXTk
3330 REM- Compute File Length (in TRACKs)
3340 bodf=l:hl=l+nr*rl:nt=INT(hl/ts)+k1:t=dt+k1
3350 IFnt>(ht-dt)+k1THENPRINT"TOO LONG!":PRINT:GOTO3270
3370 tk=k0
3380 IFut(t)=k1THEN3400
3381 tk=tk+k1:IFtk=ntTHEN3420
3390 t=t+k1:IFt>htTHENPRINT"NOT ENOUGH ROOM!":GOTO59000
3391 GOTO3380
3400 t=t+k1:IFt>htTHEN3390
3410 GOTO3370:REM-.Reset "tk"
3420 s=k1:st=t-tk+k1:et=t:st(k6)=st:et(k6)=et
3430 DISK!"co 2e79="+dt$+RIGHT$(STR$(s),k1)
3440 FORi=diTOdi+pg-k1STEPk8:IFPEEK(i)=eTHEN3460
3450 NEXTi:s=s+k1:IFs=k2THEN3430
3451 PRINT"DIRECTORY FULL!":GOTO59000
3460 t=i:i=di+pg:NEXTi
3470 FORk=k1TOk6:POKEt+k-k1,ASC(MID$(f$,k,k1)):NEXTk
3480 POKEt+k-k1,FNb(st):POKEt+k,FNb(et)
3490 DISK!"sa "+dt$+RIGHT$(STR$(s),k1)+"=2e79/1":GOSUB10000
3500 FORk=stTOet:t$=RIGHT$(STR$(k+kh),k2):DISK!"in "+t$
3510 DISK!"sa "+t$+",1=3a7e/"+pg$
3520 NEXTk:DISK open,k6,f$:DISK get,k0
3530 PRINT#k6,f$:PRINT#k6,"1"
3540 i6=53:GOSUB210
3560 FORk=k1TOnf:PRINT#k6,n$(k):PRINT#k6,fl(k):NEXTk
3570 bodf=PEEK(op(k6))+(PEEK(op(k6)+k1)*pg)-bs(k6):eodf=bodf
3580 i6=k9:GOSUB210:PRINT#k6,eodf
3590 i6=20:GOSUB210:PRINT#k6,bodf
3600 i6=31:GOSUB210:PRINT#k6,rl
3610 i6=42:GOSUB210:PRINT#k6,nr
3620 DISK close,k6:RUN
3630 :
4000 REM- Edit DMS-65D Master File
4010 GOSUB13000
4020 PRINT!(28);"DMS-65D Master File Editor":PRINT
4030 PRINT"(1) Add a New Record"
4040 PRINT"(2) Change an Old Record"
4050 PRINT"(3) Delete a Record"
4051 PRINT"(4) Return to Main Menu"
4060 PRINT:INPUT"    Your Choice ";y$:k=VAL(y$)
4070 IFk<k1ORk>k4ORk<>INT(k)THEN4020
4080 ON k GOTO 4100, 4400, 4600, 4900
4090 :
4100 REM Add a Record
4110 IFtn=nrTHENPRINT"FILE FULL":GOSUB60000:GOTO4020
4120 FORk=k1TOnf:PRINT
4130 PRINT"Enter ";n$(k):PRINTTAB(k2);
4140 FORl=k1TOfl(k)-k1:PRINT"-";:NEXTl:PRINT
4150 INPUTa$(k):l=LEN(a$(k))
4160 IFl<fl(k)THENNEXTk:GOTO4180
4170 PRINT"TOO LONG !":PRINT:GOTO4130
4180 PRINT!(28);" ";TAB(k1);"Name";TAB(32);"Contents":PRINT
4190 FORk=k1TOnf:PRINTk;TAB(k1);n$(k);TAB(32);a$(k):NEXTk
4200 PRINT:INPUT"Are These Alright ";y$:y$=LEFT$(y$+" ",k1)
4210 PRINT:IFy$="y"THEN4300
4220 INPUT"Which one did you want to change ";y$:k=VAL(y$)
4230 IFk<k1ORk>nfTHENPRINT"WHAT ??":PRINT:GOTO4180
4240 PRINT"Enter ";n$(k):PRINTTAB(k2);
```

index is equal to bodf. When BODF is reached, the counter equals the number of fields in each record. At this point, four arrays are dimensioned, each equal in size to the number of fields.

The arrays are:
n$(x) - Field Name Storage
fl(x) - Field Length Storage
i6(x) - Field Index Storage
a$(x) - Field Contents Storage

After the arrays are set up, the input index is reset to 53 and the field name/field length pairs are re-read and stored in the proper arrays. Along the way, the variable "i" is used to calculate the index of each field within each record. This allows us to immediately set either the input or output index to an individual field. This last feature is not completely implimented in this program, but it is available for your use.

With the information described so far, we can find the absolute position within the file of any piece of data we want to get ahold of. However, the job isn't done yet. We also need to determine two other values. The first is the track number on which the data we want resides and the memory address it will be called into when the track is read by our software. The BASIC command "DISK OPEN" under OS-65D performs much of the dirty work for us automatically. Once 65D locates the file to be opened, it stores three track numbers in a table. Also included on this table are three other vital pieces of information. This table is shown in Table 1.

These addresses are stored in "bs(k6)", "be(k6)", "st(k6)", "et(k6)", "cu(k6)", and "df(k6)", respectively. The defined functions FNa(x) and FNb(x) translate BCD values to decimal and decimal to BCD respectively. You'll note the discrepency between the labels using the suffix "5" and the device number "6". This is due to the way BASIC calculates the device number for OS-65D. In OS-65D, the input or output device number is stored in a single byte. More than one output device can be made active simultaneously, but only one active input device is allowed. The "5" for

```
4250 FOR1=k1TOf1(k)-k1:PRINT"-";:NEXT1:PRINT
4260 INPUTa$(k):l=LEN(a$(k)):IFl<fl(k)THEN4180
4270 PRINT"TOO LONG":PRINT:GOTO4240
4280 :
4300 tn=tn+k1:r6=tn:GOSUB400:GOTO4020
4380 :
4400 REM- Change an Old Record
4410 PRINT:PRINT"File Contains";tn;"Record(s)":PRINT
4420 IFtn=k0THENPRINT"NO RECORDS ON FILE":GOSUB60000:GOTO4020
4421 PRINT"(1) Edit by Record Number"
4422 PRINT"(2) Edit by Searching File":PRINT
4423 INPUT"    Your Choice ";y$:k=VAL(y$):PRINT
4424 IFk<k1ORk>k2ORk<>INT(k)THEN4410
4425 ON k GOTO 4430,4600
4430 INPUT"Which RECORD NUMBER did you want to see ";y$
4440 PRINT:k=VAL(y$):IFk<k1ORk>tnORk<>INT(k)THEN4430
4450 r6=k:GOSUB300
4460 PRINT!(28);:dv=PEEK(8993):GOSUB700
4480 INPUT"Did you want to change this record ";y$
4490 PRINT:IFLEFT$(y$+" ",k1)<>"y"THEN4560
4500 INPUT"Enter the FIELD NUMBER you wanted to change ";y$
4510 PRINT:k=VAL(y$):IFk<k1OR(k>nf)ORk<>INT(k)THEN4500
4520 PRINT"Enter ";n$(k):PRINT:PRINTTAB(k2);
4530 FOR1=k1TOf1(k)-k1:PRINT"-";:NEXT1:PRINT
4540 INPUTa$(k):PRINT:l=LEN(a$(k)):IFl<fl(k)THEN4560
4550 PRINT"TOO LONG!":PRINT:GOTO4520
4560 GOSUB400:GOTO4020
4570 :
4600 REM- Search File for Editing
4610 GOSUB8000:PRINT
4620 INPUT"Which FIELD NUMBER did you want to search in ";y$
4630 PRINT:k=VAL(y$):IFk<k1OR(k>nf)ORk<>INT(k)THEN4610
4640 PRINT"What STRING did you want to find in ";n$(k);
4650 INPUT" ";ss$:PRINT:l=LEN(ss$):IFl<fl(k)THEN4670
4660 PRINT"TOO LONG !":GOSUB60000:GOTO4610
4670 sf=k:sl=LEN(ss$)
4671 GOTO6000:REM- Remove this if Searches FAIL
4675 FORr6=k1TOtn:GOSUB300
4679 x=LEN(a$(sf)):FOR1=k1TOx
4680 IFMID$(a$(sf),1,sl)=ss$THEN1=x:NEXT1:GOTO4700
4681 NEXT1
4690 NEXTr6:PRINT"STRING NOT FOUND":GOSUB60000:GOTO4020
4700 PRINT!(28);:dv=PEEK(8993):GOSUB700
4710 INPUT"Is this the right record ";y$
4720 IFLEFT$(y$+" ",k1)<>"y"THEN4690
4730 x=r6:r6=tn:NEXTr6:r6=x:GOTO4460
4740 :
4800 REM- Mark a Record for Deletion
4810 PRINT"File contains";tn;"record(s)":PRINT
4820 IFtn=k0THENGOSUB60000:GOTO4020
4830 INPUT"Which RECORD NUMBER did you want to delete ";y$
4840 PRINT:k=VAL(y$):IFk<k1ORk>tnORk<>INT(k)THEN4830
4850 r6=k:GOSUB300:a$(k1)="^P":GOSUB400:GOTO4020
4860 :
4900 REM- Close DMS-65D Master File
4910 DISK get,k0:eodf=bodf+(tn*r1)
4920 i6=bs(k6)+k9:ih=INT(i6/pg):il=i6-ih*pg
4930 POKEop(k6),il:POKEop(k6)+k1,ih
4940 PRINT#k6,eodf: DISK close,k6: RUN
4950 :
```

65D refers to the bit number within that byte. More details on this are available in the OS-65D V3.3 Tutorial Manual.

Alright, getting back to the subject, the calculation to determine which track holds the record we want is done by first calculating the index to the start of the record and putting it in "i6". Then, the size of the buffer is calculated by subtracting "bs(k6)" from "be(k6)" and storing it in ts. Since the size of both buffer #6 and buffer #7 is identical, we don't need to put it in an array. The calculation to determine the track that holds the record we want is as follows:

r6 = desired record number
i6 = bodf + r6*rl
wt = st(k6) + int(i6/ts)

Where "wt" is the wanted track. After we have calculated the track we want, the program checks to see if that track is already in the buffer. If it is not, the program first checks to see if the buffer is "dirty" and if so, the contents are written out to disk - then the wanted track is called into the buffer. When the program determines that the proper track is in the buffer, it goes on to find the individual record within the buffer.

The calculation for the actual RAM address where the record will start is a bit stickier. It is:

i = i6 - ((cu(k6)-st(k6)) *ts) + bs(k6)

In the program, "il" holds the least significant byte and ih holds the most signifcant byte of the memory address. The calculation is the record index, less the number of bytes held on disk in front of the track currently in the buffer, plus the address of the start of the buffer. Once the calculation is completed, "il" and "ih" are passed to OS-65D so that BASIC can use INPUT#k6, or PRINT#k6, for reading and writing and also so that if the contents of a field crosses a track boundary, BASIC will handle calling the next track into memory automatically.

The Edit function of DMS65D allows you to add new records, alter current

```
5000 REM- File Dump Routine
5010 GOSUB13000:PRINT
5020 PRINT"File contains";tn;"record(s)":PRINT
5030 IFtn=k0THENPRINT"FILE EMPTY":GOTO59000
5040 INPUT"Which RECORD NUMBER did you want to start with ";y$
5050 PRINT:sr=VAL(y$):IFsr<k1ORsr>tnORsr<>INT(sr)THEN5020
5060 INPUT"Which RECORD NUMBER did you want to end with ";y$
5070 PRINT:er=VAL(y$):IFer<srORer>tnORer<>INT(er)THEN5020
5080 INPUT"Enter the OUTPUT DEVICE NUMBER ";y$
5090 PRINT:dv=VAL(y$):IFdv<k1ORdv>k8THEN5080
5100 FORr6=srTOer:GOSUB300:GOSUB700:NEXTr6
5110 PRINT:INPUT"Press <RETURN> to continue ";y$
5120 DISK close,k6:GOTO800
6000 REM- Fast Device #6 Search Routine
6010 r6=k1:GOSUB100:REM- Initialize Pointer to BODF
6020 TRAP6200:DISK find,ss$
6030 i6=PEEK(ip(k6))+(PEEK(ip(k6)+k1)*pg)-bs(k6)-k1
6040 i6=i6+(FNa(PEEK(9004))-st(k6))*ts
6050 r6=INT((i6-bodf)/rl)+k1
6052 GOSUB300:l=LEN(a$(sf))
6060 FORk=k1TOl
6070 IFMID$(a$(sf),k,sl)=ss$THEN6090
6080 NEXTk:r6=r6+k1:GOSUB100:GOTO6020
6090 k=l:NEXTk:dv=PEEK(8993):GOSUB700
6100 INPUT"Is this the correct record ";y$
6110 IFLEFT$(y$+" ",k1)<>"y"THENr6=r6+k1:GOSUB100:GOTO6020
6130 TRAP0:GOTO1160
6140 :
6200 TRAP0:PRINT"STRING NOT FOUND":GOSUB60000:GOTO1020
6210 :
7999 :
8000 REM- Display Fields
8010 PRINT!(28);"File: ";f$:PRINT
8011 PRINT" #";TAB(k4);"Field Name";TAB(32);"Length":PRINT
8020 FORk=k1TOnf:PRINTk;TAB(k4);n$(k);TAB(34);fl(k)-k1
8030 NEXTk:RETURN
8040 :
10000 REM- Fill Buffer #6 with Zeroes
10010 FORk=k0TO17:READa:POKEdi+k,a:NEXTk
10020 POKE di+k1,INT(ts/pg)
10030 POKE8955,121:POKE8956,46:x=USR(x):RESTORE:RETURN
10040 DATA 162,12              :REM-  LDX #$0C
10050 DATA 160,0               :REM-  LDY #$00
10060 DATA 152                 :REM-  TYA
10070 DATA 153,126,058         :REM-  STA $3A7E,Y
10080 DATA 200                 :REM-  INY
10090 DATA 208,250             :REM-  BNE *-4
10100 DATA 238,128,046         :REM-  INC $2E80
10120 DATA 202                 :REM-  DEX
10130 DATA 208,244             :REM-  BNE *-10
10140 DATA 96                  :REM-  RTS
11100 s=k1:REM- Gather Directory
11101 FORk=k0TOht:ut(k)=k0:f$(k)="":NEXTk
11105 DISK!"co 2e79="+dt$+RIGHT$(STR$(s),k1)
11110 FORi=diTOdi+pg-k1STEPk8:IFPEEK(i)=eTHEN11150
11120 st=FNa(PEEK(i+k6)):et=FNa(PEEK(i+k7))
11130 FORj=k0TOk5:f$(st)=f$(st)+CHR$(PEEK(i+j)):NEXTj
11140 f$(st)=f$(st)+CHR$(st)+CHR$(et)
11146 FORk=stTOet:ut(k)=k1:NEXTk
11150 NEXTi:IFs=k1THENs=k2:GOTO11105
11160 RETURN
```

records, and to mark records for deletion. When a record is marked for deletion, "P" is written in field #1 of that record, but the rest of the record is left intact. The add a new record function asks you to make entries for each field in a record. Then it redisplays your entries for your approval before actually writing them out to disk. You may make as many changes as you like before approving a record. There are two ways of choosing a current record to be edited. The first is to select a record by it's record number. However, since you may not know the record number but you will likely know the current contents of a record you want to change, a field search function is available.

The search function asks you which field number to search in and what should be searched for in that field. You'll note that the software actually includes two different search routines. The one that is enabled uses the OS-65D "DISK FIND" command. This is a fast machine code search, but it does have one drawback. The software will search the entire file for the string to the last track, even if it has to look beyond the last record stored in the file. Another search routine written entirely in BASIC is also included in the code and requires only that the "GOTO6000" statement be removed for it to be enabled. The BASIC routine will be slower if there are many records to be searched, but it will also discover that it cannot find the search string faster if there are very few records currently in the file. The BASIC routine demonstrates more clearly how a field search would work.

I hope you enjoy DMS65D and begin to build your own data files and application software. **BE SURE TO RUN THE "CHANGE" PROGRAM TO CREATE AT LEAST ONE DISK BUFFER BEFORE ENTERING DMS65D INTO YOUR SYSTEM!** Next month, we'll discuss a simple mailing list manager program which is based on DMS65D. For exercise, try writing a routine that removes records marked for deletion from a data file and frees up space in the data file. Good luck and have fun!

```
13000 REM- Open a DMS-65D Master File on Device 6
13010 TRAP50000:GOSUB50000
13020 INPUT"File Name ";f$:PRINT:IFLEN(f$)>k5THEN13020
13030 IFLEN(f$)<k5THENf$=f$+" ":GOTO13030
13040 f$=f$+"0":DISK open,k6,f$:TRAP0
13050 st(k6)=FNa(PEEK(9002)):et(k6)=FNa(PEEK(9003))
13060 i6=k9:GOSUB210:INPUT#k6,eodf
13090 i6=20:GOSUB210:INPUT#k6,bodf
13100 i6=31:GOSUB210:INPUT#k6,rl
13110 i6=42:GOSUB210:INPUT#k6,nr
13120 i6=53:GOSUB210:nf=k0
13130 INPUT#k6,y$,k:nf=nf+k1
13140 i6=(PEEK(9132)+PEEK(9133)*pg)-bs(k6)
13150 i6=i6+( FNa( PEEK(9004) ) - FNa( PEEK(9002)) )* ts
13160 IFi6<bodfTHEN13130
13170 IFPEEK(9004)=PEEK(9002)THEN13190
13180 DISK!"co 3a7e="+RIGHT$(STR$(FNa(PEEK(9002))),k2)+",1"
13190 i6=53:GOSUB210:DIM n$(nf),fl(nf),i6(nf),a$(nf):i=k0
13200 FORk=k1TOnf:INPUT#k6,n$(k),fl(k):i6(k)=i:i=i+fl(k):NEXTk
13210 tn=INT((eodf-bodf)/rl):RETURN
13220 :
50000 INPUT"Drive (A/B/C/D) ";y$:y$=LEFT$(y$+" ",k1)
50010 PRINT:c=ASC(y$):IFc>azTHENc=c-tt
50020 IFc<aaORc>ASC("D")THEN50000
50030 DISK!"se "+CHR$(c):RETURN
50040 :
50000 REM- Show File Not Found
50010 PRINT:PRINT"FILE: ";f$;" NOT FOUND":PRINT
50020 :
50999 REM- Abort!
59000 GOSUB60000:RUN
59010 :
60000 FORk=k1TO3000:NEXTk:RETURN
```

Table 1

ADDRESS	LABEL	DESCRIPTION
$2326	BUFST5	Memory address of start of device number 6 buffer.
$2328	BUFEN5	Memory address of end of device number 6 buffer (+1).
$232A	TRK5	Track number of 1st track in file in Binary Coded Decimal.
$232B	MAX5	Track number of last track in file in Binary Coded Decimal.
$232C	CUR5	Track number of track currently in the buffer in BCD.
$232D	DFLG5	Buffer dirty flag. If 0, it means that the buffer hasn't been altered since it was read in. If 1, it has.

Cross Reference Utility (REF)

(Editor's Note: We are much indebted to Larry Hinsley for releasing this software to the public domain and thus allowing any non-commercial use.)

by Software Consultants
6435 Summer Avenue
Memphis, TN 38134

The Cross Reference Utility (REF) is a high speed, memory resident utility running under OS-65D. The command "REF" lists all occurrences of BASIC variables, line numbers, and numeric constants for the program currently in the workspace. It sorts and lists all variables and numbers to either the console or a printer.

REF is enabled by running the installation program written in BASIC and provided here. The machine code for the REF command is stored at the top of the workspace. The BASIC program will automatically install it at the top of memory. The machine code for REF occupies 1K of RAM and reduces the amount of memory available for your programs by that same amount.

Installing REF disables the BASIC keyword "LET". After installation, programs including the keyword "LET" will no longer run. Of course, in all such programs, simply removing the word "LET" will allow the program to run. The same installation program used to install REF will also remove it and return your system to normal.

To begin installing REF, you must first create 3 files on your disk. The first one is to hold the machine code for REF. Make it one track long and name it "OBJ". The second file is to hold the BASIC program that installs REF. Make this file two tracks long and name it "REF". The third and final file is to hold the assembly language source program. On 8" systems, make it 10 tracks long. On mini-floppy systems, make it 15 tracks. You can make this file smaller if need be by omitting comments where you feel you can do without them. Be sure to write down the track number of the file "OBJ". You'll need it later on. Name this file "REFSRC".

Listing 1

```
20 REM    REF : OS-65D CROSS REFERENCE COMMAND
40 REM    WRITTEN BY SHOF BEAVERS : 04/02/82 : REV 1.2
60 REM    MODIFIED BY RICHARD L. TRETHEWEY 06/28/86
80 REM
100 REM This program is released to the Public Domain by :
140 REM            Software Consultants
160 REM              6435 Summer Ave.
180 REM             Memphis, TN 38134
200 REM              (901) 377-3503
220 REM
240 FOR I = 1 TO 24: PRINT: NEXT
260 F=12681: T=12677: TA=526: LO=670: TP=8960: DB=11897
280 PD=3: REM ..... printer device .....
300 PRINT"*** REF COMMAND ***": PRINT
320 PRINT TAB(3) "1. Enable REF command."
340 PRINT TAB(3) "2. Enable LET command."
360 PRINT: INPUT"Option:";A$: A=VAL(A$): IF A<>1 AND A<>2 GOTO 360
380 ON A GOTO 400,860
400 REM ..... endable ref command, disable let .....
420 POKE LO,ASC("R"): POKE LO+1,ASC("E"): POKE LO+2,ASC("F")+128
440 I=0
460 READ A: POKE DB+I, A: I=I+1: IF A<>96 THEN 460
480 DATA 169,127,141,148,46,173,116,44,141,149,46,169,0,170
500 DATA 141,151,46,173,0,35,56,233,3,141,152,46,173,255,255
520 DATA 141,255,255,238,148,46,208,3,238,149,46,238,151,46
540 DATA 208,237,238,152,46,232,208,229,96
560 POKE 574,121: POKE 575,46: X=USR(X): REM- Install code in RAM
580 M=PEEK(TP): REM Find current last page of user RAM
600 POKE TA,255: POKE TA+1,M-3: REM Put address in dispatch table
620 POKE TP,M-4: POKE 133,M-4: REM Set BASIC, 65D to protect it
760 REM ..... kill auto CRLF on terminal .....
780 FOR I=2813 TO 2815: POKE I,234: NEXT I: REM for alpha print
800 FOR I=2658 TO 2660: POKE I,234: NEXT I: REM for numeric print
820 POKE 23,79: POKE 24,71: REM set auto tabs for terminal
840 PRINT: PRINT "REF Command is now enabled.": PRINT: NEW
860 REM ..... enable let command, disable ref .....
880 POKE LO,ASC("L"): POKE LO+1,ASC("E"): POKE LO+2,ASC("T")+128
900 POKE TA,165: POKE TA+1,9: REM restore dispatch table to LET code
920 M=PEEK(TP): POKE TP,M+4: POKE 133,M+4
940 PRINT: PRINT"LET Command is now enabled.": PRINT: NEW
```

The next step is to enter the assembler you use, type in the assembly language program and save it in the file "REFSRC". The installation program assumes that an appropriately assembled version of REF is stored in front of the BASIC program. Thus, you must first set the origin address on line #580 in the assembly language program given in Listing 4 to reflect your system's memory size. For 24K systems, set the origin at $5C00, 32K systems should use $7C00, and 48K systems should leave the setting at $BC00. In addition, make sure that "DEVICE" in line #290 reflects the printer device number for your system. Don't forget to use the OS-65D device number here, and not the one you use in BASIC programs.

Now that you have the source code properly modified, its time to assemble the program to memory. If you're using the OSI Assembler Editor, be sure to execute the "H" command to protect the high end of memory;

24K systems: H5B00
32K systems: H7B00
48K systems: HBB00

If you're using ASM-Plus, respond with these same numbers when prompted.

Once the machine code is in memory, save it to the object code file "OBJ" you created above with the command:

```
24K systems:  !SA TT,1=5C00/4
32K systems:  !SA TT,1=7C00/4
48K systems:  !SA TT,1=BC00/4
```

where "TT" above is the track number where the file "OBJ" resides on your disk.

Now, leave the assembler you're using and boot up a vanilla version of OS-65D's BASIC. Run the program "CHANGE" and tell it you want to reserve 1034 bytes in front of the workspace. When CHANGE is done, it NEWs itself out of existence and you're ready to type in the installation program from Listing 1.

But before you begin typing in the program, you must call the machine code for REF into memory from the disk file "OBJ". Use the following command to do this;

OS-65D V3.2
8" systems: DISK!"CA 317F=TT,1"
5" systems: DISK!"CA 327F=TT,1"

OS-65D V3.3
All systems: DISK!"CA 3A7F=TT,1"

again, where "TT" is the track number for the file "OBJ".

Now type in and save the installation program with the command;

```
DISK!"PUT REF"
```

Finally, run the installation program and select item #1 to install REF.

To use the REF command, load the program you want to cross-reference into the workspace. If you want to cross-reference a single variable or numeric constant, enter "REF" followed by that variable name or the number at the "OK" prompt in BASIC. For variable names, just enter a one or two character name since that is the maximum size BASIC recognizes as unique. Trailing "%" or "$" for integer and string variables should not be entered. If you want a complete cross-reference of the program, enter the command "REF*" to send the output to the console or "REF#" to send the output to the printer device you have selected.

The output generated by this code is as follows: The variable name or number is printed first, followed by a colon, and then for each occurrance, a line number/count pair is displayed. Separate entries will be displayed for floating point, integer, and string variable types, which will also be differentiated by subscripted and non-subscripted types, allowing for all possible variations. See the example below.

```
10 REM            Cross Reference Utility Example
20 REM
30 A=1:A%=1:A(1)=1:A%(1)=1:A$="X":A$(1)="X"
40 ON T GOTO 40,60
60 T%=1:A$="String constants are not searched,
   i.e.,X=1 not found"
70 GOSUB40:REM Same for Remarks...X=1
80 GOTO10
90 ABCD=1.2578435 E12:ABCD$="X"
95 A=A+A+A+A%+A+A+A+A+A%+A+A+A+A%+A+A+A%+A+A+A+A+A

1   :        30/7    60/1

1.2578435E12  :      90/1

10  :        80/1

40  :        40/1    70/1

60  :        40/1

A   :        30(%1   30($1    30(1    30/%1    30/$1    30/1
             60/$1   95/%4    95/20

AB  :        90/$1   90/1

T   :        40/1    60/%1
```

CompuServe Subscription Kits

CompuServe is the host for the Ohio Scientific Special Interest Group that you've heard about here for so long. It is the largest such network in the country offering many services in addition to OSI SIG. You can send and recieve MCI Mail™ via CompuServe as well as checking airline schedules and rates with the OnLine Airline Guide™, or even check the latest stock market quotes just to name a few.

PEEK [65] is offering CompuServe subscription kits for just $32.00 plus shipping. That's 20% off the 1st price of $39.95. The kit includes an instruction manual and a $25.00 credit to help get you started. Armed with this kit, a modem, and a terminal program you're off and running.

DISK DRIVE RECONDITIONING

WINCHESTER DRIVES

FLAT RATE CLEAN ROOM SERVICE.
(parts & labor included)

Shugart	SA4008	23meg	$550.00
Shugart	SA1004	10meg	$390.00
Seagate	ST412	10meg	$295.00

FLOPPY DRIVE FLAT RATES

8" Single Sided Shugart	$190.00
8" Double Sided Shugart	$250.00
8" Single Sided Siemens D&E Series	$150.00
8" Double Sided Siemens P Series	$170.00

Write or call for detailed brochure
90 Day warranty on Floppy & Large Winch.
1 Yr. Warranty on 5" & 8" Winchesters.

Phone: (417) 485-2501

FESSENDEN COMPUTERS
116 N. 3RD STREET
OZARK, MO 65721

```
10                      ;-----------------
20                      ;** XREF OS65-U **
30                      ;-----------------
40                      ;CROSS REFERENCE OF BASIC VARIABLES
50                      ;
60                      ;---------------------------------
70                      ;SYSTEM ADDRESSES AND SUBROUTINES
80                      ;---------------------------------
90                      ;
100 00C7=               VARPNT=$C7    ; POINTER TO 1ST CHAR IN SEARCH STRING
110 0AEE=               CHROUT=$AEE   ; SUBROUTINE TO PRINT CHAR IN ACC
120 00AF=               BINHI=$AF     ; BINARY HIGH NUMBER
130 00B0=               BINLO=$B0     ; BINARY LOW NUMBER
140 1B44=               BUILD1=$1B44  ; SUBROUTINES TO TAKE BINHI AND
150 1CEC=               BUILD2=$1CEC  ; BINLO - CONVERT TO DECIMAL
160                                   ; RESULT IN PNTBUF
170 0E1E=               SNERR=$0E1E   ; SYNTAX ERROR ROUTINE
180 5FFC=               BSIZE=$5FFC   ; 2 BYTE OFFSET FOR BASIC WORKSPACE
190 0474=               RETBAS=$474   ; RETURN TO IMMEDIATE MODE BASIC
200 0100=               PNTBUF=$0100  ; PRINT BUFFER FOR DECIMAL NUMBERS
210 0016=               PRNPOS=22     ; PRESENT PRINT POSITION
220 0018=               PRNLMT=24     ; TAB PRINT LIMIT
230 0A73=               CRLF=$A73     ; PRINT CR/LF ROUTINE
240 007E=               ENUML=$7E     ; END OF NUMERIC VARIABLES, LOW
250 007F=               ENUMH=$7F     ; END OF NUMERIC VARIABLES, HIGH
260 0080=               EMEML=$80     ; END OF MEMORY, LOW
270 0081=               EMEMH=$81     ; END OF MEMORY, HIGH
280 2DA6=               OUTBYT=$2DA6  ; OUTPUT DISTRIBUTOR
290 0004=               DEVICE=$04    ; PRINTER DEVICE
300                     ;
310                     ;---------------------------------------
320                     ;ZERO PAGE LOCATIONS USED BY THIS ROUTINE
330                     ;---------------------------------------
340                     ;
350 0030=               NUMCNT=$30        ; COUNTER FOR NUMERIC STRING
360 0031=               ZPAGE=$31         ; FIRST ZERO PAGE LOCATION
370 0031=               FPVAR=ZPAGE       ; COUNTER FOR FLOATING POINT VARIABLE
380 0032=               STVAR=ZPAGE+1     ; COUNTER FOR STRING VARIABLE
390 0033=               INVAR=ZPAGE+2     ; COUNTER FOR INTEGER VARIABLE
400 0034=               SFPVAR=ZPAGE+3    ; COUNTER SUBSCRIPTED F.P VARIABLE
410 0035=               SSTVAR=ZPAGE+4    ; COUNTER SUBSCRIPTED STRINGS
420 0036=               SINVAR=ZPAGE+5    ; COUNTER SUBSCRIPTED INTEGERS
430 0037=               VARLEN=ZPAGE+6    ; LENGTH OF SEARCH STRING
440 0038=               SFLAG=ZPAGE+7     ; SEARCH FLAG
450 0039=               LNPNT=ZPAGE+8     ; POINTER TO CHAR IN BASIC LINE
460 003B=               TEMP=ZPAGE+10     ; TEMPORARY STORAGE
470 003C=               TEMP1=ZPAGE+11    ; TEMPORARY STORAGE
480 003D=               TEMP2=ZPAGE+12    ; TEMPORAY STORAGE
490 003E=               LINELO=ZPAGE+13   ; LINE NUMBER LOW
500 003F=               LINEHI=ZPAGE+14   ; LINE NUMBER HIGH
510 0040=               TESTLN=ZPAGE+15   ; LENGTH OF TEST STRING
520 0041=               TABPOS=ZPAGE+16   ; TAB PRINT STOP POSITION
530 0042=               TERM=ZPAGE+17     ; TERMINAL OUTPUT DEVICE
540 0043=               TABLE=ZPAGE+18    ; ADDRESS OF TABLE:ALL VAR ROUTINE
550 0045=               INPOS=ZPAGE+20    ; INPUT POSITION FOR NEW VARIABLES
560 0047=               TEMPT=ZPAGE+22    ; TEMPORARY TABLE FOR VARIABLE SEARCH
570                     ;
580 BC00                *=$BC00
590                     ;--------------
600                     ;INITIALIZATION
610                     ;--------------
620                     ;
630 BC00 48             PHA           ; SAVE THE FIRST CHARACTER
640 BC01 A900     INIT  LDA #$00      ; INIT VARIABLE COUNTERS TO 0
650 BC03 8538           STA SFLAG     ; INITIALIZE SEARCH FLAG
660 BC05 A008           LDY #8        ; SET TO CLEAR 8 ZERO PAGE LOCATIONS
670 BC07 992E00   CLOOP STA ZPAGE-3,Y
680 BC0A 88             DEY           ; GET SET FOR NEXT VARIABLE
690 BC0B D0FA           BNE CLOOP     ; GO DO IT IF NOT 0
700 BC0D 68             PLA           ; RESTORE THE FIRST CHARACTER
710                     ;
720 BC0E C92E           CMP #$2E      ; FIRST CHARACTER A '.'
730 BC10 F027           BEQ DETLEN    ; YES, COUNT AS A NUMERIC
740 BC12 C9AB           CMP #$AB      ; CROSS REFERENCE ALL VARIABLES?
750 BC14 D004           BNE CK1       ; NO, AT LEAST NOT TO THE TERMINAL
760 BC16 8538           STA SFLAG     ; YES, LET'S SET THE FLAG
770 BC18 F03E           BEQ ADJADD    ; BRANCH TO ADJADD
780 BC1A C923     CK1   CMP #'#       ; REFERENCE ALL VARIABLES TO PRINTER?
790 BC1C D004           BNE BEGIN     ; NO, LET'S CHECK FOR SYNTAX
800 BC1E 8538           STA SFLAG     ; YES, SET THAT FLAG
810 BC20 F036           BEQ ADJADD    ; BRANCH TO ADJUST THE ADDRESS FOR BASIC
```

```
 820                    ;
 830 BC22 C930    BEGIN  CMP #$30   ; 1ST CHAR LESS THAN '0'
 840 BC24 901B           BCC JSNERR ; YES, DO SYNTAX ERROR
 850 BC26 C93A           CMP #$3A   ; 1ST CHAR NUMERIC
 860 BC28 B003           BCS TALPHA ; NO IT'S NOT
 870 BC2A 4C39BC         JMP DETLEN ; LETS CONTINUE
 880 BC2D C941    TALPHA CMP #$41   ; 1ST CHAR LESS THAN 'A'
 890 BC2F 9010           BCC JSNERR ; YES, DO SYNTAX ERROR
 900 BC31 C95B           CMP #$5B   ; 1ST CHAR GREATER THAN 'Z'
 910 BC33 B00C           BCS JSNERR ; YES, SYNTAX ERROR
 920 BC35 A901           LDA #$01   ; SET FLAG FOR ALPHA SEARCH
 930 BC37 8538           STA SFLAG  ; 1 CHAR VARIABLE
 940                    ;
 950 BC39 B1C7    DETLEN LDA (VARPNT),Y ; GET CHAR FROM BUFFER
 960 BC3B F007           BEQ  CKLEN ; IF NULL GOTO CKLEN
 970 BC3D C8             INY        ; GET SET FOR NEXT CHARACTER
 980 BC3E 4C39BC         JMP DETLEN ; LET'S GO GET IT
 990 BC41 4C1E0E  JSNERR JMP SNERR  ; DO SYNTAX ERROR AND RETURN TO BASIC
1000                    ;
1010 BC44 8437    CKLEN  STY VARLEN ; SAVE THE VARIABLE LENGTH
1020 BC46 A538           LDA SFLAG  ; IS THIS ALPHA OR NUMERIC
1030 BC48 F00E           BEQ ADJADD ; IT'S NUMERIC SO LET'S GO
1040 BC4A A537           LDA VARLEN ; GET THE VARIABLE LENGTH
1050 BC4C C901           CMP #$01   ; IS IT 1
1060 BC4E F008           BEQ ADJADD ; YES WE ARE READY TO GO
1070 BC50 C903           CMP #$03   ; IS THE LENGTH GREATER THAN 2
1080 BC52 B0ED           BCS JSNERR ; YES, DO SYNTAX ERROR
1090 BC54 A980           LDA #$80   ; SET SEARCH FLAG FOR 2 CHAR VARIABLE
1100 BC56 8538           STA SFLAG
1110                    ;
1120 BC58 2066BF  ADJADD JSR SETADD  ; INITIALIZE POINTER TO BASIC WORKSPACE
1130 BC5B A538           LDA SFLAG   ; WHAT ARE WE SEARCHING FOR
1140 BC5D C9AB           CMP #$AB    ; ALL VARIABLES TO TERMINAL?
1150 BC5F F00A           BEQ ALLVAR  ; YES! LET'S GO
1160 BC61 C923           CMP #'#     ; ALL VARIABLES TO PRINTER?
1170 BC63 F006           BEQ ALLVAR  ; YES! LET'S GO
1180 BC65 206ABD         JSR SEARCH  ; LOOK FOR THIS ONE VARIABLE
1190 BC68 4C7404         JMP RETBAS  ; GOTO BASIC WHEN DONE
1200                    ;
1210                    ;-----------------------------------------
1220                    ;ALLVAR : SEARCH FOR ALL VARIABLES AND OUTPUT
1230                    ;INFORMATION TO TERMINAL(*) OR TO PRINTER(#).
1240                    ;-----------------------------------------
1250                    ;
1260 BC6B ADA62D  ALLVAR LDA OUTBYT  ; SAVE THE PRESENT OUTPUT DEVICE
1270 BC6E 8542           STA TERM    ; AT ZERO PAGE 'TERM'
1280 BC70 A538           LDA SFLAG   ; WHICH OUTPUT?
1290 BC72 C9AB           CMP #$AB    ; TERMINAL?
1300 BC74 F005           BEQ GO
1310 BC76 A904           LDA #DEVICE ; GET THE PRINTER DEVICE NUMBER
1320 BC78 8DA62D         STA OUTBYT  ; SET THE OUTPUT DISTRIBUTOR
1330                    ;
1340 BC7B A57F    GO     LDA ENUMH   ; GET THE HIGH BYTE OF LAST MEMORY
1350 BC7D 8544           STA TABLE+1 ; SET BEGINING OF TABLE
1360 BC7F A57E           LDA ENUML   ; GET THE LOW BYTE
1370 BC81 8543           STA TABLE   ; SET IT
1380 BC83 A000           LDY #$00    ; SET END OF TABLE FLAG
1390 BC85 A9FF           LDA #$FF    ; TO PRESENT END OF TABLE
1400 BC87 9143           STA (TABLE),Y
1410                    ;
1420 BC89 203FBE  FINDVR JSR GETCHR  ; GET CHARACTER FROM BASIC LINE
1430 BC8C D029           BNE SETTAB  ; NOT A NULL-PRESS ON
1440 BC8E 853B           STA TEMP    ; RESET TEMP
1450 BC90 AA             TAX         ; GET SET TO READ NEXT
1460 BC91 A003           LDY #$03    ; TWO CHARACTERS
1470 BC93 88      CNTNUL DEY         ; COUNT THIS CHARACTER
1480 BC94 F00C           BEQ FIND1   ; IFY=0 THEN WE HAVE TESTED THEM BOTH
1490 BC96 20C1BE         JSR BLNPNT  ; INCREMENT BASIC LINE POINTER
1500 BC99 A139           LDA (LNPNT,X) ; GET THE NEXT CHARACTER
1510 BC9B D0F6           BNE CNTNUL  ; IF NOT NULL LET'S CHECK THE NEXT ONE
1520 BC9D E63B           INC TEMP    ; NOT NULL SO 'BUMP' TEMP
1530 BC9F 4C93BC         JMP CNTNUL  ; LET'S FINISH COUNTING NULLS
1540 BCA2 A53B    FIND1  LDA TEMP    ; GET THE NULL COUNT
1550 BCA4 C902           CMP #$02    ; IF 2 NULLS THEN WE ARE DONE
1560 BCA6 D003           BNE CNSCAN  ; NO, PRESS ON
1570 BCA8 4C85BF         JMP OUTVAR  ; YES, OUTPUT THE INFORMATION
1580 BCAB 20C1BE  CNSCAN JSR BLNPNT  ; NOT DONE SO SKIP THE
1590 BCAE 20C1BE         JSR BLNPNT  ; NEXT TWO CHARACTERS
1600 BCB1 20C1BE         JSR BLNPNT  ; GET READY FOR SOME MORE
1610 BCB4 4C89BC         JMP FINDVR  ; LET'S KEEP SEARCHING
1620                    ;
```

```
1630 BCB7 90D0    SETTAB BCC FINDVR  ; NOT ALPHA/NUMERIC : TRY AGAIN
1640 BCB9 A000           LDY #$00    ; STORE VARIABLE IN TEMPORARY TABLE
1650 BCBB 994700 BUILDT  STA TEMPT,Y ; SAVE THIS CHARACTER
1660 BCBE C8             INY         ; SET FOR NEXT CHARACTER
1670 BCBF 203FBE         JSR GETCHR  ; GO GET IT FROM BASIC LINE
1680 BCC2 F002           BEQ SETVAR  ; IF NULL LET'S PUT IT IN TABLE
1690 BCC4 B0F5           BCS BUILDT  ; IF STILL ALPHA TRY THE NEXT
1700 BCC6 A900    SETVAR LDA #$00    ; GET NULL FOR DELIMITER
1710 BCC8 994700         STA TEMPT,Y ; SAVE IT
1720 BCCB C8             INY         ; ADJUST Y FOR THE NULL
1730 BCCC 8437           STY VARLEN  ; SAVE THE VARIABLE LENGTH
1740 BCCE C004           CPY #$04    ; Y<=3
1750 BCD0 9010           BCC SETV1   ; YES, PRESS ON
1760 BCD2 B547           LDA TEMPT,X ; GET FIRST CHARACTER FROM TEMP TABLE
1770 BCD4 C941           CMP #'A     ; LESS THAN 'A'
1780 BCD6 900A           BCC SETV1   ; YES, PRESS ON
1790 BCD8 A900           LDA #$00    ; GET SET TO LIMIT VARIABLE
1800 BCDA A002           LDY #$02
1810 BCDC 994700         STA TEMPT,Y ; PUT IN THE NEW END OF VARIABLE
1820 BCDF C8             INY         ; ADJUST Y
1830 BCE0 8437           STY VARLEN  ; AND SAVE
1840 BCE2 A000    SETV1  LDY #$00    ; SET FOR INDIRECT ADDRESSING
1850 BCE4 A57F           LDA ENUMH   ; SET TABLE TO FRONT FOR SCAN
1860 BCE6 8544           STA TABLE+1
1870 BCE8 A57E           LDA ENUML
1880 BCEA 8543           STA TABLE
1890                     ;
1900 BCEC B143    COMPAR LDA (TABLE),Y ; GET NEXT CHARACTER FROM TABLE
1910 BCEE D94700         CMP TEMPT,Y ; COMPARE THE CHARACTERS
1920 BCF1 900F           BCC FNEXTV  ; IF < GOTO FIND NEXT VARIABLE
1930 BCF3 F002           BEQ CNEXT   ; IF = THEN TEST THE REST
1940 BCF5 B020           BCS INSERT  ; IF > GOTO INSERT THE VARIABLE
1950 BCF7 C8      CNEXT  INY         ; BUMP THE INDEX
1960 BCF8 C437           CPY VARLEN  ; Y=VARIABLE LENGTH
1970 BCFA 90F0           BCC COMPAR  ; IT'S LESS THAN SO TRY AGAIN
1980 BCFC C900           CMP #$00    ; SET ZERO FLAG
1990 BCFE F089           BEQ FINDVR  ; GO FIND NEXT VARIABLE
2000 BD00 D015           BNE INSERT  ; GO INSERT VARIABLE IN TABLE
2010                     ;
2020 BD02 A000    FNEXTV LDY #$00    ; SET FOR INDIRECT
2030 BD04 207EBF         JSR INPNT   ; INCREMENT TABLE POINTER
2040 BD07 B143           LDA (TABLE),Y ; GET NEXT CHARACTER
2050 BD09 D006           BNE FNEXT1  ; IF NOT NULL CONTINUE
2060 BD0B 207EBF         JSR INPNT   ; BUMP THE LINE POINTER
2070 BD0E 4CECBC         JMP COMPAR  ; LET'S TRY AGAIN
2080 BD11 C9FF    FNEXT1 CMP #$FF    ; ARE WE AT THE END?
2090 BD13 F0D7           BEQ COMPAR  ; YES, RETURN TO LOOP
2100 BD15 D0EB           BNE FNEXTV  ; ALWAYS BRANCH TO FIND NEXT VARIABLE
2110                     ;
2120 BD17 A543    INSERT LDA TABLE   ; SAVE CURRENT TABLE POINTER
2130 BD19 8545           STA INPOS   ; AT INPUT POSITION
2140 BD1B A544           LDA TABLE+1
2150 BD1D 8546           STA INPOS+1
2160 BD1F A000           LDY #$00    ; RESET Y FOR INDEXING
2170 BD21 B143    FEND   LDA (TABLE),Y ; GET CHARACTER FROM TABLE
2180 BD23 C9FF           CMP #$FF    ; ARE WE AT THE END?
2190 BD25 F006           BEQ FOUND1  ; YES, TEST MEMORY
2200 BD27 207EBF         JSR INPNT   ; BUMP THE TABLE POINTER
2210 BD2A 4C21BD         JMP FEND    ; LET'S KEEP SEARCHING
2220 BD2D A544    FOUND1 LDA TABLE+1 ; COMPARE PRESENT MEMORY LOCATION
2230 BD2F C581           CMP EMEMH   ; TO END OF MEMORY
2240 BD31 9009           BCC MOVE    ; IT'S COOL SO LET'S GO
2250 BD33 A543           LDA TABLE   ; TEST THE LOW BYTES
2260 BD35 C580           CMP EMEML
2270 BD37 9003           BCC MOVE    ; ALL COOL!
2280 BD39 4C1E0E         JMP SNERR   ; DO OUT OF MEMORY ERROR (SYNTAX ERROR)
2290                     ;
2300 BD3C A200    MOVE   LDX #$00
2310 BD3E A437           LDY VARLEN
2320 BD40 A143    MOVELP LDA (TABLE,X) ; GET CHARACTER FROM TABLE
2330 BD42 9143           STA (TABLE),Y ; SAVE AT TABLE + VARIABLE LENGTH
2340 BD44 A544           LDA TABLE+1
2350 BD46 C546           CMP INPOS+1   ; ARE WE AT THE INPUT POSITION
2360 BD48 D00A           BNE ADJTAB
2370 BD4A A543           LDA TABLE
2380 BD4C C545           CMP INPOS     ; ARE THE LOW BYTES =
2390 BD4E D004           BNE ADJTAB
2400 BD50 A0FF           LDY #$FF
2410 BD52 D00B           BNE PUTIT
2420 BD54 A543    ADJTAB LDA TABLE     ; GET LOW BYTE OF TABLE POINTER
2430 BD56 D002           BNE *+4       ; SKIP DEC. HIGH BYTE IF NOT 0
```

```
2440 BD58 C644           DEC TABLE+1   ; DECREMENT HIGH BYTE
2450 BD5A C643           DEC TABLE     ; DECREMENT LOW BYTE
2460 BD5C 4C40BD         JMP MOVELP
2470                     ;
2480 BD5F C8       PUTIT INY           ; GET SET FOR NEXT CHARACTER
2490 BD60 B94700         LDA TEMPT,Y   ; GET CHARACTER FROM STORAGE
2500 BD63 9145           STA (INPOS),Y ; PUT IT IN THE TABLE
2510 BD65 D0F8           BNE PUTIT     ; IF NOT THE NULL THEN CONTINUE
2520 BD67 4C89BC         JMP FINDVR    ; SEARCH FOR THE NEXT VARIABLE
2530                     ;
2540                     ;---------------------------------------
2550                     ; SEARCH : SUBBROUTINE TO SCAN BASIC
2560                     ; PROGRAM AND LOOK FOR VARIABLE POINTED
2570                     ; TO BY VARPNT. WILL PRINT ANY OCCURANCES
2580                     ; OF THE VARIABLE AND THE NUMBER OF
2590                     ; OCCURANCES WITHIN A SPECIFIC LINE.
2600                     ;---------------------------------------
2610                     ;
2620 BD6A 20730A  SEARCH JSR CRLF
2630 BD6D A000           LDY #$00
2640 BD6F B1C7    PVARLP LDA (VARPNT),Y ; GET CHAR FROM VARIABLE
2650 BD71 F00B           BEQ CONOUT    ; IF NULL THE EXIT PRINT LOOP
2660 BD73 843B           STY TEMP      ; SAVE THE INDEX
2670 BD75 20EE0A         JSR CHROUT    ; PRINT THIS CHARACTER
2680 BD78 A43B           LDY TEMP      ; RESTORE THE INDEX
2690 BD7A C8             INY           ; AND INCREMENT
2700 BD7B 4C6FBD         JMP PVARLP    ; GO PRINT THE NEXT CHARACTER
2710 BD7E A920    CONOUT LDA #$20      ; GO PRINT A SPACE
2720 BD80 20EE0A         JSR CHROUT
2730 BD83 A93A           LDA #$3A      ; PRINT A ':'
2740 BD85 20EE0A         JSR CHROUT
2750 BD88 A900           LDA #0        ; RESET TAB POSITION
2760 BD8A 8541           STA TABPOS
2770 BD8C A516           LDA PRNPOS    ; TAB TO NEXT POSITION
2780 BD8E C909           CMP #9
2790 BD90 9004           BCC SRLOOP    ; NO
2800 BD92 A90A           LDA #10
2810 BD94 8541           STA TABPOS
2820 BD96 203FBE  SRLOOP JSR GETCHR    ; LETS READ A CHARACTER
2830 BD99 D003           BNE S1        ; NOT A NULL, LETS CONTINUE
2840 BD9B 4C8ABE         JMP TEST      ; SEE WHAT THIS NULL MEANS
2850 BD9E 90F6    S1     BCC SRLOOP    ; IF NOT ALPHA/NUMERIC TRY AGAIN
2860 BDA0 A000           LDY #$00      ; GET SET TO INDEX THE INPUT STRING
2870 BDA2 853B           STA TEMP      ; SAVE THE CHARACTER
2880 BDA4 B1C7           LDA (VARPNT),Y ; GET FIRST CHAR IN SEARCH STRING
2890 BDA6 C53B           CMP TEMP      ; ARE THE FIRST CHARACTERS THE SAME
2900 BDA8 F008           BEQ S2        ; YES, LETS CONTINUE
2910                     ;
2920 BDAA 2060BF  CONTSH JSR NXTNAL    ; NO, GET THE NEXT NON-ALPHA CHARACTER
2930 BDAD D0E7           BNE SRLOOP    ; NOT A NULL, LETS CONTINUE
2940 BDAF 4C8ABE         JMP TEST      ; SEE WHAT THE NULL MEANS
2950 BDB2 A538    S2     LDA SFLAG     ; WHAT ARE WE SEARCHING FOR?
2960 BDB4 303D           BMI S5        ; SKIP TEST FOR LENGTH IF 2 CHAR VAR
2970 BDB6 A539           LDA LNPNT     ; SET LINE POINTER BACK 1
2980 BDB8 D002           BNE *+4       ; IF NOT 0 SKIP DEC HIGH BYTE
2990 BDBA C63A           DEC LNPNT+1   ; DEC HIGH BYTE
3000 BDBC C639           DEC LNPNT     ; DEC LOW BYTE
3010 BDBE A539    ADJLPN LDA LNPNT     ; SAVE LINE POINTER FOR LATER
3020 BDC0 853C           STA TEMP1
3030 BDC2 A53A           LDA LNPNT+1
3040 BDC4 853D           STA TEMP2
3050 BDC6 8440           STY TESTLN    ; INITIALIZE TEST LENGTH
3060                     ;
3070 BDC8 E640    CNTLEN INC TESTLN    ; BUMP THE TEST LENGTH
3080 BDCA 203FBE         JSR GETCHR    ; GET THE NEXT CHARACTER
3090 BDCD F002           BEQ S3        ; IF NULL LETS TEST THE RESULTS
3100 BDCF B0F7           BCS CNTLEN    ; IF STILL ALPHA/NUMERIC TRY AGAIN
3110 BDD1 C640    S3     DEC TESTLN    ; ADJUST FOR NON-ALPHA CHARACTER
3120 BDD3 A53C           LDA TEMP1     ; RESTORE LINE POINTER
3130 BDD5 8539           STA LNPNT
3140 BDD7 A53D           LDA TEMP2
3150 BDD9 853A           STA LNPNT+1
3160 BDDB A540           LDA TESTLN    ; LETS SEE IF LENGTH OF TEST STRING =
3170 BDDD C537           CMP VARLEN    ; LENGTH OF SEARCH STRING
3180 BDDF F002           BEQ S4        ; YES, LET'S SEE IF THEY ARE THE SAME
3190 BDE1 D0C7           BNE CONTSH    ; NO, LETS SEARCH SOME MORE
3200                     ;
3210 BDE3 A000    S4     LDY #$00      ; GET SET TO COMPARE THE STRINGS
3220 BDE5 203FBE  S4LP   JSR GETCHR    ; GET NEXT CHARACTER FROM BASIC LINE
3230 BDE8 D1C7           CMP (VARPNT),Y  ; ARE THEY THE SAME
3240 BDEA D0BE           BNE CONTSH    ; NO LETS SEARCH AGAIN
```

```
3250 BDEC C8             INY          ; GET SET FOR NEXT CHARACTER
3260 BDED C640           DEC TESTLN   ; WE HAVE TESTED ANOTHER CHARACTER
3270 BDEF F00C           BEQ FOUND    ; IF 0 THEN WE HAVE CHECKED THE STRING
3280 BDF1 D0F2           BNE S4LP     ; TEST THE NEXT CHARACTER
3290 BDF3 C8       S5    INY          ; GET SET FOR SECOND CHARACTER IN STRING
3300 BDF4 203FBE         JSR GETCHR   ; GET THE NEXT CHARACTER FROM BASIC LINE
3310 BDF7 909D           BCC SRLOOP   ; NON-ALPHA SO PRESS ON
3320 BDF9 D1C7           CMP (VARPNT),Y ; ARE THEY THE SAME
3330 BDFB D0AD           BNE CONTSH   ; NO, LETS SEARCH SOME MORE
3340                     ;
3350 BDFD A538     FOUND LDA SFLAG    ; WHAT ARE WE SEARCHING FOR
3360 BDFF D005           BNE FVAR     ; BRANCH IF SEARCHING FOR A VARIABLE
3370 BE01 E630           INC NUMCNT   ; INCREMENT THE NUMERIC COUNTER
3380 BE03 4C96BD         JMP SRLOOP
3390 BE06 A000     FVAR  LDY #$00     ; INITIALIZE TEMP: DETERMINE VAR TYPE
3400 BE08 843B           STY TEMP
3410 BE0A 2060BF         JSR NXTNAL   ; GET THE FIRST CHAR AFTER THE VARIABLE
3420 BE0D C924           CMP #'$      ; IS IT A STRING?
3430 BE0F D002           BNE F1       ; NO
3440 BE11 E63B           INC TEMP     ; SET TEMP TO 1
3450 BE13 C925     F1    CMP #'%      ; IS IT AN INTEGER?
3460 BE15 D006           BNE F2       ; NO
3470 BE17 A902           LDA #$02     ; YES, ADJUST TEMP
3480 BE19 853B           STA TEMP
3490 BE1B F00B           BEQ F3       ; SEE IF IT'S SUBSCRIPTED
3500 BE1D C928     F2    CMP #'(      ; IS IT A SUBSCRIPTED FLOATING POINT VAR
3510 BE1F D007           BNE F3       ; NO IT'S NOT
3520 BE21 A903           LDA #$03     ; YES, ADJUST TEMP TO REFLECT THIS
3530 BE23 853B           STA TEMP
3540 BE25 4C38BE         JMP TOTAL    ; LETS GO TALLY
3550                     ;
3560 BE28 A53B     F3    LDA TEMP
3570 BE2A F00C           BEQ TOTAL
3580 BE2C B139           LDA (LNPNT),Y
3590 BE2E C928           CMP #'(      ; IS IT A SUBSCRIPTED VARIABLE
3600 BE30 D006           BNE TOTAL    ; NO, LETS TALLY
3610 BE32 E63B           INC TEMP     ; YES ADJUST TEMP TO REFLECT VHIS
3620 BE34 E63B           INC TEMP
3630 BE36 E63B           INC TEMP
3640 BE38 A63B     TOTAL LDX TEMP
3650 BE3A F631           INC ZPAGE,X  ; ADJUST THE PROPER V. COUNTER
3660 BE3C 4C96BD         JMP SRLOOP   ; LETS SEARCH AGAIN
3670                     ;
3680                     ;----------------------------------------
3690                     ; GET CHARACTER ROUTINE
3700                     ; RETURNS WITH CARRY SET IF ALPHA/NUMERIC
3710                     ; CARRY IS CLEAR IF NOT
3720                     ; Z FLAG USED ONLY FOR NULL, END OF LINE
3730                     ;----------------------------------------
3740                     ;
3750 BE3F A200     GETCHR LDX #$00    ; GET SET FOR INDEXED LOAD
3760 BE41 A139           LDA (LNPNT,X) ; GET THE NEXT CHARACTER
3770 BE43 F029           BEQ BACK1    ; IF NULL THEN RETURN
3780 BE45 C98E           CMP #$8E     ; IS IT THE 'REM'
3790 BE47 F027           BEQ REM      ; YES, LET'S GO TO THE NEXT LINE
3800 BE49 C922           CMP #$22     ; HAVE WE FOUND A QUOTATION
3810 BE4B F02C           BEQ QUOTE    ; YES, LETS SKIP IT
3820 BE4D 20C1BE         JSR BLNPNT   ; GET SET FOR NEXT CHARACTER
3830 BE50 C920           CMP #$20     ; IS IT THE SPACE
3840 BE52 F0EB           BEQ GETCHR   ; TRY AGAIN
3850 BE54 C92E           CMP #$2E     ; IS IT A '.'
3860 BE56 D004           BNE G1       ; NO, PRESS ON
3870 BE58 C900           CMP #$00     ; THIS WAS ADDED TO CLEAR THE 'Z' FLAG
3880 BE5A D010           BNE BACK
3890 BE5C C930     G1    CMP #$30     ; CHAR > ASCII '0'
3900 BE5E 900E           BCC BACK1    ; YES, LET'S GO BACK
3910 BE60 C93A           CMP #$3A     ; CHAR ASCII '9' OR LESS
3920 BE62 9008           BCC BACK     ; YES, LET'S RETURN WITH IT
3930 BE64 C941           CMP #$41     ; CHAR LESS THAN ASCII 'A'
3940 BE66 9006           BCC BACK1    ; YES, LET'S GO BACK
3950 BE68 C95B           CMP #$5B     ; CHAR GREATER THAN ASCII 'Z'
3960 BE6A B002           BCS BACK1    ; NO, IT'S NOT
3970 BE6C 38       BACK  SEC          ; SET CARRY FOR ALPHA/NUMERIC
3980 BE6D 60             RTS
3990 BE6E 18       BACK1 CLC          ; CLEAR CARRY (NON-ALPHA)
4000 BE6F 60             RTS
4010                     ;
4020 BE70 20C1BE   REM   JSR BLNPNT   ; GET SET FOR NEXT CHARACTER
4030 BE73 A139           LDA (LNPNT,X) ; GET IT!
4040 BE75 F0C8           BEQ GETCHR   ; WE FOUND A NULL SO TRY AGAIN
4050 BE77 D0F7           BNE REM      ; NO NULL SO GET NEXT CHARACTER
```

```
4060                    ;
4070 BE79 20C1BE  QUOTE JSR BLNPNT  ; BUMP THE LINE POINTER
4080 BE7C A139          LDA (LNPNT,X) ; GET THE NEXT CHARACTER
4090 BE7E F0BF          BEQ GETCHR  ; FOUND THE NULL!
4100 BE80 C922          CMP #$22    ; HAVE WE FOUND THE NEXT QUOTE
4110 BE82 D0F5          BNE QUOTE   ; NO, LET'S GET THE NEXT CHARACTER
4120 BE84 20C1BE        JSR BLNPNT  ; BUMP LINE POINTER PAST THE QUOTE
4130 BE87 4C6EBE        JMP BACK1
4140                    ;
4150 BE8A A200    TEST  LDX #$00    ; INITALIZE TEMP STORAGE
4160 BE8C 863B          STX TEMP
4170 BE8E 20C1BE        JSR BLNPNT  ; GET SET FOR NEXT CHARACTER
4180 BE91 A139          LDA (LNPNT,X) ; GET IT
4190 BE93 F002          BEQ T1      ; IF NULL TRY THE HIGH BYTE
4200 BE95 E63B          INC TEMP    ; BUMP TEMP (NOT DONE YET)
4210 BE97 20C1BE  T1    JSR BLNPNT  ; GET SET FOR NEXT CHARACTER
4220 BE9A A139          LDA (LNPNT,X) ; GET IT!
4230 BE9C F002          BEQ T2      ; IF NULL LETS TEST
4240 BE9E E63B          INC TEMP
4250                    ;
4260 BEA0 A53B    T2    LDA TEMP    ; TEMP TELLS IF WE ARE DONE
4270 BEA2 F019          BEQ DONE    ; WE HAVE FOUND THE 3 NULLS!!
4280 BEA4 20C8BE        JSR PRINT   ; LET'S SEE IF WE FOUND ANY VARIABLES
4290 BEA7 A200          LDX #$00    ; RESTORE THE INDEX
4300 BEA9 20C1BE        JSR BLNPNT  ; GET SET FOR THAT NEXT CHARACTER
4310 BEAC A139          LDA (LNPNT,X) ; GET IT!
4320 BEAE 853E          STA LINELO  ; STORE THE LOW BYTE OF THE LINE NUMBER
4330 BEB0 20C1BE        JSR BLNPNT  ; GET READY AGAIN
4340 BEB3 A139          LDA (LNPNT,X) ; GET THE HIGH BYTE OF THE LINE NUMBER
4350 BEB5 853F          STA LINEHI  ; SAVE IT
4360 BEB7 20C1BE        JSR BLNPNT  ; BUMP THAT LINE POINTER
4370 BEBA 4C96BD        JMP SRLOOP  ; LET'S TRY AGAIN
4380                    ;
4390 BEBD 20C8BE  DONE  JSR PRINT   ; LET'S SEE IF WE FOUND ANY VARIABLES
4400 BEC0 60            RTS
4410                    ;
4420 BEC1 E639    BLNPNT INC LNPNT  ; INCREMENT THE LOW BYTE
4430 BEC3 D002          BNE BLNRET  ; IF NOT ZERO THEN RETURN
4440 BEC5 E63A          INC LNPNT+1 ; INCREMENT THE HIGH BYTE
4450 BEC7 60      BLNRET RTS
4460                    ;
4470                    ;-------------------------------------------------
4480                    ; PRINT ROUTINE : CHECKS VARIABLE COUNTERS - IF ANY
4490                    ; ARE NON-ZERO THEN THE INFORMATION IS PRINTED AND
4500                    ; THE VARIABLE IS CLEARED. USES PDEC TO PRINT THE
4510                    ; INFORMATION IN DECIMAL FORM INSTEAD OF BINARY.
4520                    ;-------------------------------------------------
4530                    ;
4540 BEC8 A006    PRINT LDY #$06    ; GET SET TO CHECK VARIABLE COUNTERS
4550 BECA B93000  CKLOOP LDA ZPAGE-1,Y   ; LOAD THE VARIABLE
4560 BECD D004          BNE OUTPUT  ; FOUND A VARIABLE! LET'S PRINT IT
4570 BECF 88            DEY         ; GET SET FOR NEXT VARIABLE
4580 BED0 10F8          BPL CKLOOP  ; GO IF WE ARE NOT DONE
4590 BED2 60            RTS
4600                    ;
4610 BED3 843C    OUTPUT STY TEMP1  ; SAVE THE Y REGISTER FOR LATER
4620 BED5 A53E          LDA LINELO  ; GET LOW BYTE OF LINE NUMBER
4630 BED7 85B0          STA BINLO   ; PUT AT BINARY LOW
4640 BED9 A53F          LDA LINEHI  ; GET HIGH BYTE OF LINE NUMBER
4650 BEDB 85AF          STA BINHI   ; PUT AT BINARY HIGH
4660 BEDD 18            CLC
4670 BEDE A541          LDA TABPOS  ; PRESENT PRINT POSITION
4680 BEE0 690A          ADC #10     ; ADD TAB FIELD SIZE
4690 BEE2 8541          STA TABPOS  ; STOP PRINTING HERE WHEN DONE
4700 BEE4 C518          CMP PRNLMT  ; COMPARE TO TAB LIMIT
4710 BEE6 9007          BCC TABLP   ; GO TAB OVER TO NEXT FIELD
4720 BEE8 20730A        JSR CRLF    ; NEW LINE IF >=
4730 BEEB A90A          LDA #10     ; RESET TAB POSITION
4740 BEED 8541          STA TABPOS
4750                    ;
4760 BEEF A516    TABLP LDA PRNPOS  ; PRESENT PRINT POSITION
4770 BEF1 C541          CMP TABPOS  ; CHECK IF AT END OF TAB FIELD
4780 BEF3 B008          BCS TABEND  ; IF SO, PRESS ON
4790 BEF5 A920          LDA #$20
4800 BEF7 20EE0A        JSR CHROUT  ; PRINT A SPACE
4810 BEFA 4CEFBE        JMP TABLP   ; CONTINUE LOOP
4820 BEFD 2044BF  TABEND JSR PDEC   ; GO PRINT THE LINE NUMBER
4830 BF00 A43C          LDY TEMP1   ; RESTORE THE Y REGISTER
4840 BF02 C004          CPY #$04    ; NOT SUBSCRIPTED?
4850 BF04 900D          BCC PSLSH   ; IF NOT, PRINT /
4860 BF06 A928          LDA #$28    ; LOAD ASCII FOR '('
```

```
4870 BF08 20EE0A        JSR CHROUT  ; PRINT THAT BABY
4880 BF0B A43C          LDY TEMP1   ; RESTORE INDEX
4890 BF0D 88            DEY
4900 BF0E 88            DEY         ; SUBTRACT 3 FROM THE Y REGISTER
4910 BF0F 88            DEY
4920 BF10 4C1ABF        JMP OUT1    ; SKIP /
4930                    ;
4940 BF13 A92F    PSLSH LDA #$2F    ; LOAD ASCII FOR '/'
4950 BF15 20EE0A        JSR CHROUT  ; PRINT IT    •
4960 BF18 A43C          LDY TEMP1   ; RESTORE INDEX
4970 BF1A C002    OUT1  CPY #$02    ; IF Y<2 THEN SKIP VARIABLE TAGS
4980 BF1C 900F          BCC POCCUR  ; GO PRINT NUMBER OF OCCURANCES
4990 BF1E D008          BNE PINT    ; NO, GO PRINT INTEGER TAG
5000 BF20 A924          LDA #$24    ; LOAD ASCII FOR '$'
5010 BF22 20EE0A        JSR CHROUT  ; PRINT THAT BABY!
5020 BF25 4C2DBF        JMP POCCUR  ; GO PRINT NUMBER OF OCCURANCES
5030 BF28 A925    PINT  LDA #$25    ; LOAD ASCII FOR '%'
5040 BF2A 20EE0A        JSR CHROUT  ; GUESS
5050                    ;
5060 BF2D A43C    POCCUR LDY TEMP1  ; RESTORE INDEX INTO VARIABLE TABLE
5070 BF2F B93000        LDA ZPAGE-1,Y ; GET NUMBER OF OCCURANCES
5080 BF32 85B0          STA BINLO   ; AND SET TO CONVERT
5090 BF34 A900          LDA #0
5100 BF36 85AF          STA BINHI
5110 BF38 993000        STA ZPAGE-1,Y   ; CLEAR THE VARIABLE COUNTER
5120 BF3B 2044BF        JSR PDEC    ; GO PRINT THE DECIMAL VALUE
5130 BF3E A43C          LDY TEMP1   ; RESTORE INDEX
5140 BF40 88            DEY         ; GET READY FOR NEXT VARIABLE
5150 BF41 1087          BPL CKLOOP  ; GO BACK AND TEST NEXT VARIABLE IF < 0
5160 BF43 60            RTS         ; WE HAVE CHECKED ALL THE VARIABLES
5170                    ;
5180                    ;-----------------------------------------------
5190                    ; PDEC : PRINT BINARY NUMBER AS DECIMAL. USES
5200                    ; SYSTEM ROUTINES BUILD1 AND BUILD2 TO TAKE
5210                    ; BINARY NUMBER IN BINLO AND BINHI ($AF,$B0)
5220                    ; AND CONVERT TO DECIMAL. RESULT IS AT $0100
5230                    ; AND IS TERMINATED BY A NULL.
5240                    ;-----------------------------------------------
5250                    ;
5260 BF44 A290    PDEC  LDX #$90    ; INITIALIZE X REGISTER (?)
5270 BF46 38            SEC
5280 BF47 20441B        JSR BUILD1  ; SUBROUTINES TO CONVERT BINARY NUMBER
5290 BF4A 20EC1C        JSR BUILD2  ; TO DECIMAL FOR PRINTOUT
5300 BF4D A001          LDY #1      ; SET INDEX TO NOT PRINT LEADING SPACE
5310 BF4F B90001  PRNTLP LDA PNTBUF,Y ; GET THE NEXT CHAR. OUT OF THE BUFFER
5320 BF52 F00B          BEQ RET     ; IF NULL THEN WE ARE THROUGH
5330 BF54 843D          STY TEMP2   ; SAVE THAT INDEX
5340 BF56 20EE0A        JSR CHROUT  ; OUTPUT THE CHARACTER
5350 BF59 A43D          LDY TEMP2   ; RESTORE THE INDEX
5360 BF5B C8            INY         ; GET SET FOR NEXT CHARACTGR
5370 BF5C 4C4FBF        JMP PRNTLP  ; LET'S GO GET IT!
5380 BF5F 60      RET   RTS         ; WE HAVE PRINTED THE DECIMAL NUMBER
5390                    ;
5400 BF60 203FBE  NXTNAL JSR GETCHR ; GET THE NEXT CHARACTER FROM BASIC LINE
5410 BF63 B0FB          BCS NXTNAL  ; IF ALPHA/NUMERIC TRY AGAIN
5420 BF65 60            RTS         ; NO - GO BACK
5430                    ;
5440 BF66 A960    SETADD LDA #$60   ; INITIALIZE LINE POINTER TO
5450 BF68 853A          STA LNPNT+1 ; NORMAL START OF BASIC WORKSPACE
5460 BF6A A900          LDA #$00
5470 BF6C 8539          STA LNPNT
5480 BF6E ADFC5F        LDA BSIZE   ; GET LOW BYTE OF OFFSET FOR BASIC
5490 BF71 18            CLC         ; GET SET TO ADD
5500 BF72 6539          ADC LNPNT   ; ADD WITH CARRY TO LOW BYTE
5510 BF74 8539          STA LNPNT   ; SAVE THE RESULT
5520 BF76 ADFD5F        LDA BSIZE+1 ; GET HIGH BYTE OFFSET
5530 BF79 653A          ADC LNPNT+1 ; ADD IT
5540 BF7B 853A          STA LNPNT+1 ; SAVE THE RESULT
5550 BF7D 60            RTS         ; GO BACK
5560                    ;
5570 BF7E E643    INPNT INC TABLE   ; INCREMENT THE LOW BYTE
5580 BF80 D002          BNE INRET   ; NOT ZERO TTHEN RETURN
5590 BF82 E644          INC TABLE+1 ; INCREMENT HIGH BYTE
5600 BF84 60      INRET RTS
5610                    ;
5620                    ;---------------------------------------
5630                    ; OUTVAR : ROUTINE TO TAKE VARIABLES FROM
5640                    ; TEMPORARY TABLE AND USE SEARCH TO SCAN
5650                    ; FOR THE SPECIFIC VARIABLE.
5660                    ;---------------------------------------
5670                    ;
```

```
5680 BF85 A57E    OUTVAR LDA ENUML        ; SET TABLE POINTER TO FRONT OF TABLE
5690 BF87 8543           STA TABLE
5700 BF89 A57F           LDA ENUMH
5710 BF8B 8544           STA TABLE+1
5720 BF8D A200    OUTLP  LDX #$00         ; SET X FOR INDIRECT
5730 BF8F A000           LDY #$00         ; RESET Y
5740 BF91 A143           LDA (TABLE,X)    ; GET CHARACTER FROM TEMP TABLE
5750 BF93 C9FF           CMP #$FF         ; ARE WE AT THE END?
5760 BF95 F039           BEQ RBASIC       ; YES, LET'S GET OUT OF HERE
5770 BF97 91C7           STA (VARPNT),Y   ; NO, SAVE THE FIRST CHARACTER
5780 BF99 207EBF  SETVR  JSR INPNT        ; BUMP VARIABLE POINTER
5790 BF9C C8             INY              ; GET SET FOR THE NEXT ONE
5800 BF9D A143           LDA (TABLE,X)    ; GET IT
5810 BF9F 91C7           STA (VARPNT),Y   ; SAVE IT FOR CROSS REFERENCE
5820 BFA1 D0F6           BNE SETVR        ; IF NOT NULL THE KEEP LOOPING
5830 BFA3 207EBF         JSR INPNT        ; BUMP THE VARIABLE POINTER FOR NEXT
5840 BFA6 8437           STY VARLEN       ; SAVE THE VARIABLE LENGTH
5850 BFA8 A000           LDY #$00         ; GET SET TO OUTPUT VARIABLE
5860 BFAA A900           LDA #$00         ; SET THE SEARCH FLAG
5870 BFAC 8538           STA SFLAG
5880 BFAE B1C7           LDA (VARPNT),Y   ; GET THE FIRST CHARACTER
5890 BFB0 C941           CMP #$41         ; CHARACTER LESS THAN 'A'
5900 BFB2 9010           BCC PRLOOP       ; YES, WE ARE READY TO GO
5910 BFB4 A537           LDA VARLEN       ; NO, LET'S TEST THE LENGTH
5920 BFB6 C902           CMP #$02         ; IS IT 2 OR MORE
5930 BFB8 B006           BCS SETFLG       ; YES, GO SET SFLAG
5940 BFBA A901           LDA #$01         ; SET SFLAG FOR 1 CHARACTER VARIABLE
5950 BFBC 8538           STA SFLAG
5960 BFBE D004           BNE PRLOOP       ; ALWAYS BRANCH TO PRINT LOOP
5970 BFC0 A980    SETFLG LDA #$80         ; SET SFLAG FOR 2 CHARACTER VARIABLE
5980 BFC2 8538           STA SFLAG
5990 BFC4 2066BF  PRLOOP JSR SETADD       ; RESET BASIC POINTER TO FRONT
6000 BFC7 206ABD         JSR SEARCH       ; GO SEARCH FOR THIS VARIABLE
6010 BFCA 20730A         JSR CRLF         ; DO CR/LF
6020 BFCD 4C8DBF         JMP OUTLP        ; KEEP LOOPING TILL DONE
6030 BFD0 A542    RBASIC LDA TERM         ; RESTORE TERMINAL DEVICE
6040 BFD2 8DA62D         STA OUTBYT
6050 BFD5 4C7404         JMP RETBAS       ; BACK TO CONSOL MODE
```


Book Bargains!

Now's the time to pick up a copy of the reference manuals you've needed. Don't forget to add shipping costs.

Sam's Service Manuals

These are the only professional guides available for servicing and modifying your OSI equipment. They include full schematics, block diagrams, wave form tracings, parts lists, and diagnostic tips. They were written for the pre-1980 series of OSI systems, but since OSI never has changed that much they are still valuable no matter when your computer was made.

C1P Regular: $7.95 Sale: $4.00
C4P Regular: $15.95 Sale: $10.00
C2/C3 Regular: $39.95 Sale: $25.00

65V Primer

This is an introductory guide to machine code that shows you how to program your video system using the Monitor ROM. An excellent tutorial on the fundamentals of machine code.

Regular: $5.95 Sale: $3.00

User Guides

These are excellent books. They are complete tutorials on all of the standard hardware and software for video systems. Covers many topics not documented anywhere else. If you've been struggling along with just the big blue notebooks, don't wait! Order today!

C1P-MF Regular: $8.95 Sale: $4.00
C4P-MF Regular: $8.95 Sale: $5.00
C8P-DF Regular: $8.95 Sale: $5.00

Assembler/Editor - Ex. Mon. Manual

Until recently, OSI included the Assembler/Editor and Extended Monitor software with all copies of OS-65D. However, even when it was free, there was little documentation accompanying the disks. If you've been looking for instructions on these two programs, this is the book for you!

Regular: $6.95 Sale: $4.00

Professional Computers Set Up and Operations Manual

A valuable guide for installing and using OSI serial systems. Includes an overview of classic OSI software for these systems. The book also provides information on how to program the C3 series using the Z-80 and 6800 microprocessors.

Regular: $9.95 Sale: $6.00

Introductory Manuals

These books don't contain a lot of information that isn't duplicated in many other places. Still, for the first-time user, they can be a valuable reference to keep by your system while you're learning. Specify C1P/C1P-MF, C4P cassette, C4P-MF, or C8P-DF.

Regular: $6.95 Sale: $2.00

How To Program Microcomputers

By William Barden, this book explains the instruction set of the 8000, 6500, and 6800 series of microprocessors. While not OSI-specific, this book contains many valuable algorithms for solving problems in machine code using the microprocessors available in OSI computers.

Regular: $8.95 Sale: $4.00

A Better Random Number Generator (in less than 1 page!)

by Daniel J. McDonald
Asbury College CPO
Wilmore, KY 40390

Have you ever tried to use the random number generator supplied by Microsoft for any amount of time? It really doesn't work too well. I have noticed that after a while, it starts to repeat itself in a cycle of about 60 different numbers or so - clearly not sufficient for any use whatsoever. I happened to mention this to a mathematics professor at Asbury College and he pulled out a copy of Art of Computer Programming, Volume II: Semi-Numerical Algorithms by Donald E. Knuth. In this tome there are many wonderful algorithms, including a real good random number generator. "Its so good", the professor said, "that they don't know how good it is." Supposedly, it will repeat the first order of magnitude once every 2^55 iterations. A machine code implimentation of this algorithm is given in Listing 1 here.

A few notes about installation: Assemble the routine. Then enter the Extended Monitor and set the stack pointers to $00 and $3C respectively. Then, starting at the beginning of the stack, put in 102 pseudo-random values. **(Editor's Note: The references to "STACK POINTER" all refer to the program's own local storage for the table of random numbers and program's internal pointers to that table, NOT THE 6502's STACK POINTER)** You can use BASIC's RND(1) function for this if you like. Finally, save the machine code to disk, noting the track and sector number where you are saving it so you can include the information in the BASIC programs that use the code. Your programs that use this code should always re-save the program and the "stack" of random numbers back to disk to insure a supply of new numbers and thus avoiding the need to always "re-seed" the stack. The BASIC program example in Listing 2 outlines this technique. Note that "XXXX" is the start of the stack, "TT,S" is the track and sector location of where you want to store the machine code on your disk, and "YYYY" is "XXXX"+116, the start of the program itself in memory.

The way the program works is quite simple. The random data that you put in the stack is added to another piece 30 words away. This sum is stored in the old location, so that even after 55 uses of the random number generator, you get a brand new number. You can continually add because the 2-byte words have a limit of 65535 as a maximum value and after that they start over. The routine keeps adding words from different parts of the stack, and the result is a constant flow of unique and random numbers.

```
 10                      ;LABLES
 20 D100=                PA=$D100
 30 D101=                PB=$D101
 40 D102=               STACK=$D102
 50 D1F0=               OUTVAR=$D1F0
 60 1218=               RETVAL=$1218
 70 D1A0                  *=$D1A0
 80 D1A0 18               CLC
 90 D1A1 AE01D1           LDX PB
100 D1A4 BD02D1           LDA STACK,X
110 D1A7 E8               INX
120 D1A8 8E01D1           STX PB
130 D1AB AE00D1           LDX PA
140 D1AE 7D02D1           ADC STACK,X
150 D1B1 9D02D1           STA STACK,X
160 D1B4 A8               TAY
170 D1B5 E8               INX
180 D1B6 BD02D1           LDA STACK,X
190 D1B9 8E00D1           STX PA
200 D1BC AE01D1           LDX PB
210 D1BF 7D02D1           ADC STACK,X
220 D1C2 E8               INX
230 D1C3 8E01D1           STX PB
240 D1C6 AE00D1           LDX PA
250 D1C9 9D02D1           STA STACK,X
260 D1CC 8DF0D1           STA OUTVAR
270 D1CF E8               INX
280 D1D0 8E00D1           STX PA
290 D1D3 8A               TXA
300 D1D4 38               SEC
310 D1D5 E96C             SBC #108
320 D1D7 D003             BNE B1
330 D1D9 8D00D1           STA PA
340 D1DC AD01D1    B1     LDA PB
350 D1DF 38               SEC
360 D1E0 E96C             SBC #108
370 D1E2 D003             BNE FINE
380 D1E4 8D01D1           STA PB
390 D1E7 ADF0D1    FINE   LDA OUTVAR
400 D1EA 4C1812           JMP RETVAL
```

Listing 2

```
10 DEF FNR(X) = INT((PEEK(OUTM)*256+PEEK(OUTM+1))*X/65536)+1
20 REM - where OUTM is the decimal value of OUTM in the ASM code
30 ML = YYYY- (INT(YYYY/256)*256): MH = INT(YYYY/256)
40 POKE 574, ML: POKE 575,MH: REM- Point USR(X) to our code
50 DISK!"CA XXXX=TT,S": REM- Call program into memory at $XXXX
60 X=USR(X): REM- Generate a new random number
70 A = FNR(100): REM- Fetch a number between 1 and 100

....program text to end. At the end of the program, where it quits

1000 DISK!"SA TT,S=XXXX/1"
```

Challenger 4x4 Character Set

by D. G. Johansen
P.O. Box 252
La Honda, CA 94020

(Editor's Note: Mr. Johansen is the author of the BETA/65 language used in this article.)

This article shows how to display characters on your screen which are four times larger than normal. This is a perfect size for display to several viewers and those with impaired vision. Larger characters support video applications such as message boards, score boards, teleprompter, etc.

The Challenger C4P has 64 columns and this allows 16 characters in the 4x4 format. This is suitable for displaying two or three words across the screen. With 32 rows available for the C4P video screen, up to 8 lines may be displayed in 4x4 format. This is enough for two or three sentences.

Figure 1 shows a set of ideal components for building a 4x4 character set. As each cell is 2x2, the final character has 8x8 cells, just the right size to duplicate the ASCII character set magnified by four.

The term "ideal" is defined as follows: (1) The set is complete - this means that all 16 combinations are available and (2) The set is logically ordered with bit-mapping as shown in Figure 2.

There are clearly major advantages to such a set. First, by having a complete set, all possible combinations are available. Also, bit-mapping to a 64x128 element screen would be feasible. Finally, by logically ordering the set it is more easily manipulated by software. For example, a character inversion would correspond to logical inversion of the lower four bits.

In Table 1, the correspondence of the Challenger character set and the logically ordered set is given. It is necessary that substitution be made for the "L-shaped" components 135, 139, 141, and 142. Depending on best-fit esthetics, either a "full" or

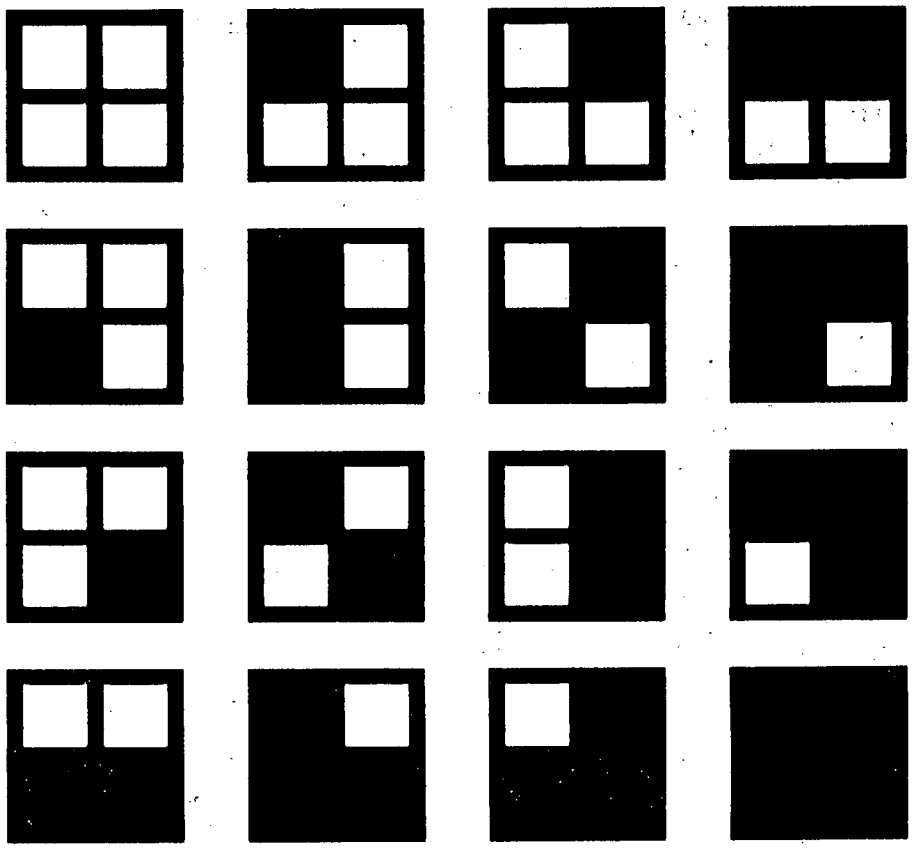

Figure 1 - Ideal Components for 4x4 Character Set

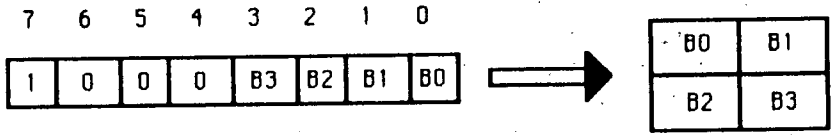

Figure 2 - Ideal Bit-mapping

"half-diagonal" may be used for these components. The result is distinctive in appearance to several of the characters. This lends a definite personality to the displayed message.

Listing 1 shows the 4x4 character set for ASCII values from 32 (space) to 95 (underscrore), including numbers, upper-case letters, plus most of the common alpha-numeric characters. For entry convenience, the last two numbers of the line number correspond to the ASCII value of the character.

Listing 2 shows a short program written in BETA/65 which displays a message in 4x4 characters across your screen. The subroutine SHOW_4x4, starting at line 100, prints to the screen the data-field characters referenced to the parameter named "label". The screen position is indicated by the argument values passed to the parameters named "line" and "column".

Several calls are made to SHOW_4x4, starting at line 10. The data field information is given in lines above 1000, with each character data preceded by its symbolic name. Again, there is no significance to the line numbers in the data field other than entry convenience. Each call passes to the subroutine the desired reference field information (preceding "%") and the value field information (following "%").

OSI Assembler Symbol Table Dump Utility

by Matt Holcomb
382 Newark Street
Aurora, CO 80010

I thought I would share one the the utilities I've written, a program which sorts and prints the OSI Assembler's symbol table list. To use it, simply (1) load/assemble this program into unused memory; (2) load AND ASSEMBLE your target program using any "A" command (A, A1, A2,A3); and (3) enter "!GO 8000" (or wherever you've put this utility in memory). A word of caution though: Make sure the symbol list generated in step (2) doesn't overwrite the dump utility code. Use the "Hxxxx" command to limit the OSI Assembler's memory usage.

A few general comments: The OSI Assembler stores 6 character symbols in a compressed 4 byte field. Bytes 0 and 1 represent the first 3 characters of the symbol name in LO/HI format. Bytes 2 and 3 hold the last 3 characters in the same LO/HI format. And bytes 4 and 5 hold the assembly-time value of the symbol.

There are 40 valid characters which can make up a symbol name. Each character is assigned a numerical value:

0 = <SPACE>
1 through 26 = "A" to "Z"
27 through 36 = "0" to "9"

So, a 6 character symbol can be compressed into a 4-bit word as:

BYTE 0
char1*40^2 + char2*40^1 + char 3

BYTE 1
char4*40^2 + char5*40^1 + char6

Notice that the maximum "word" (arising from "$$$") would be 39x1600 + 39*40 + 39 = $F9FF. The assembler flags undefined symbols simply by setting the MSB of the 2nd 3 character word above this highest value (namely, to $FF), and storing the character that would normally be there in the MSB of the value field (byte 5, above).

To see how this compression works, use the assembler's (undocumented) <QUOTE> command (i.e. "). Simply follow a quotation mark with up to 3 characters, and the Assembler will generate its 2 byte representation. For example:

```
.WORD "SYM, "BOL
```

generates:

```
B57A  (for "SYM" LO/HI)
E40E  (for "BOL" LO/HI)
```

Similarly, opcodes can be encoded:

```
407D .WORD "TAX
```

You'll find (among other things) the 6502 mneumonics encoded at $0Fxx. (For those of you who are real hackers, disassemble the assembler itself... you're in for quite a few ELEGANT surprises!)

The program in Listing 2 is not noted for speed, which serves to point out that video routines should be comitted to high-speed machine code. The proper role of high-level languages such as BETA/65, should be to set up and LINK the machine activity. The advantages of low-level and high-level languages are speed and flexibility, respectively. These are complimentary, and an optimized program would take this into account.

The 4x4 character set presented here provides an alternative size between normal (1x1) and "high-res" (8x8) ASCII characters for display on your Challenger screen.

Listing starts on page 38

Call for Articles

As noted in Column One, PEEK [65]'s library of articles is extremely low. I hope you'll take the time to share some of the work you've done with the rest of us. Thanks a lot.

AD$

FOR SALE: Two Cipher interface boards and DEI cartridge Tape backup drives. Originally $3500 ea. Both fully checked and aligned. Edward Dell (603) 924-9464

FOR SALE: 12 fully populated 520 boards. Each provides 16K of static RAM. Not tested. $50.00 plus shipping. Contact PEEK[65]

FORTH $24.95. Utilities available also. Free catalog. Aurora Software, 37 South Mitchell, Arlington Heights, IL 60005

Have you got something to sell? Why not take out a classified ad in PEEK? Ads cost 35 cents per word, not including "price" words. Copy is due 30 days before the cover month.

DON'T FORGET TO RENEW!

OSI-CALC: SPREADSHEET PROGRAM

OSI-CALC has been a smash hit here at PEEK[65]. Written entirely in BASIC by Paul Chidley of TOSIE, the program gives you a 26 column by 36 row spreadsheet with many features. Don't let the fact that it's written in BASIC fool you. It's VERY FAST.

Each cell can contain text (left or right justified) or numeric data (in floating point or dollar format) or a formula which computes its results based on the contents of the other cells. Formulas can perform addition, subtraction, multiplication or division using cell contents and/or numeric constants. Spreadsheets can be stored on disk, and the program does very nice printing too.

OSI-CALC requires 48K of memory and OS-65D V3.3. Specify video or serial system and mini-floppy or 8" disks. Price $10.00 plus $3.70 shipping ($13.70 total).

```
                ; -ASM Symbol Listing Program-
                ; Matt Holcomb :: 26 May, 1986
                ;
000A=           SYMSTR = $0A        ; start of ASM symbol table
0018=            E:O:W = $18        ; end of workspace
0030=             WORD = $30
0032=             BYTE = $32
0034=              MEM = $34
0036=             MEM2 = $36
0038=             FLAG = $38
0039=             TEMP = $39
003A=            FIELD = $3A
003E=            VALUE = FIELD+4
                ;
0020=            SPACE = 32
000D=               CR = 13
000A=               LF = 10
                ;
2343=           OUTPUT = $2343
2D73=           STROUT = $2D73
2D92=           PRT2HX = $2D92
2AC6=           DFLTIO = $2AC6
2322=           OUTDST = $2322
2F79=              DIR = $2F79
                ; ---------------------------------------------------------
8000            *=$8000
                ; Swap back ASM constants from DOS context.
                ;
8000 AD832F             LDA SYMSTR+DIR
8003 AE842F             LDX SYMSTR+1+DIR
8006 D006               BNE NRESET          Backup SYMSTR -- on return
8008 ADC280             LDA BAKSYM          from DOS, SYMSTR is reset
800B AEC380             LDX BAKSYM+1        to zero.
800E 850A       NRESET  STA SYMSTR
8010 860B               STX SYMSTR+1
8012 8DC280             STA BAKSYM
8015 8EC380             STX BAKSYM+1
8018 AD912F             LDA E:O:W+DIR
801B 38                 SEC
801C E905               SBC #5              Point to start of 1st symb
801E 8518               STA E:O:W
8020 AD922F             LDA E:O:W+1+DIR
8023 E900               SBC #0
8025 8519               STA E:O:W+1
                ;
8027 AD2223             LDA OUTDST          output only to video
802A 48                 PHA
802B ADC62A             LDA DFLTIO
802E 8D2223             STA OUTDST
8031 20732D             JSR STROUT
8034 52         .BYTE 'Reading & sorting ...',CR,LF,0
804C 202781             JSR TEW:M
804F 20BD80     RSLOOP  JSR P:CR            Print CR
8052 20C480             JSR MEMSYM          Read & print symbol
8055 204381             JSR INSERT          Add & sort
8058 203081             JSR SUB6CK          Point to next symbol
805B D0F2               BNE RSLOOP          and repeat for all.
                ;
805D 20732D             JSR STROUT
8060 0D         .BYTE CR,'       ',CR,LF,0
                ;
806A 68                 PLA
806B 8D2223             STA OUTDST          Reset I/O
                ;
806E 202781             JSR TEW:M           Now, PRINT the sorted list
8071 A000               LDY #0
8073 8439               STY TEMP            # of symb/line counter
8075 20C480     PRTL1$  JSR MEMSYM          Read & print symbol
8078 20732D             JSR STROUT
807B 3D         .BYTE '= ',0
807C 20
807D 00
807E A63D               LDX FIELD+3         Check to see if symbol is
8080 E8                 INX                 defined.  If not, X=$FF.
8081 F012               BEQ NODEF
                ;
8083 A924               LDA #'$             Symbol defined: print its
8085 204323             JSR OUTPUT          value in HEX format.
8088 A53F               LDA VALUE+1
808A 20922D             JSR PRT2HX
808D A53E               LDA VALUE
```

```
800 808F 20922D          JSR PRT2HX
810 8092 4C9E80          JMP DEF:OK
820                ;
830 8095 20732D NODEF    JSR STROUT         Symbol not defined.
840 8098 75       .BYTE 'undef',0
850                ;
860 809E E639    DEF:OK  INC TEMP           increase # on line count
870 80A0 A539            LDA TEMP
880 80A2 2903            AND #%00000011     Use %00000001 for 32
890 80A4 D006            BNE SAMLIN         char/line systems.
900 80A6 20B880          JSR P:LFCR         Print CR LF
910 80A9 4CB380          JMP PRTL3$
920 80AC 20732D SAMLIN   JSR STROUT         Tab to next column
930 80AF 20       .BYTE '   ',0
930 80B0 20
930 80B1 20
930 80B2 00
940                ;
950 80B3 203081 PRTL3$   JSR SUB6CK         Point to next symb &
960 80B6 D0BD            BNE PRTL1$         repeat for whole list.
970                ;
980 80B8 A90A    P:LFCR  LDA #LF
990 80BA 204323          JSR OUTPUT
1000 80BD A90D   P:CR    LDA #CR
1010 80BF 4C4323 J:OUT   JMP OUTPUT         And, we're done.
1020               ;
1030 80C2 0000   BAKSYM  .WORD 0            Backup SYMSTR
1040               ;
1050               ; ---------------------------------------------------------
1060               ; MEMSYM :: read symbol into memory & print it
1070               ;
1080 80C4 A005   MEMSYM  LDY #5
1090 80C6 B134   MEMSL1  LDA (MEM),Y
1100 80C8 993A00         STA FIELD,Y
1110 80CB 88             DEY
1120 80CC 10F8           BPL MEMSL1
1130               ;
1140               ; Now, PRINT the compressed 4-bit symbol.
1150               ;
1160 80CE A53A           LDA FIELD          First 3 characters
1170 80D0 A43B           LDY FIELD+1
1180 80D2 20DF80         JSR DECBLK
1190 80D5 A53C           LDA FIELD+2        Second 3 characters
1200 80D7 A43D           LDY FIELD+3
1210 80D9 C0FF           CPY #$FF
1220 80DB D002           BNE DECBLK         If symbol undef, get
1230 80DD A43F           LDY VALUE+1        from VALUE field.
1240               ;
1250 80DF 8530   DECBLK  STA WORD
1260 80E1 8431           STY WORD+1
1270 80E3 A006           LDY #40*40/256
1280 80E5 A940           LDA #40*40
1290 80E7 200D81         JSR DECNRM         Extract first char.
1300 80EA A000           LDY #40/256
1310 80EC A928           LDA #40
1320 80EE 200D81         JSR DECNRM         Extract second char.
1330 80F1 A430           LDY WORD           Residue is third char.
1340 80F3 98     DECSTR  TYA                Translate 0-39 into
1350 80F4 18             CLC                AZ09:.$ format
1360 80F5 F012           BEQ SPACE.
1370 80F7 E91A           SBC #'Z+1-'A
1380 80F9 900C           BCC AZ.
1390 80FB E90B           SBC #':+1-'0
1400 80FD 9006           BCC I0:.
1410 80FF F002           BEQ I..
1420               ; If we're here, we have A=1 for a '$'
1430 8101 69F4           ADC #'$-'.-1-1
1440 8103 69F2   I..     ADC #'.-':-1-1
1450 8105 69DF   I0:.    ADC #':-'Z-1
1460 8107 693A   AZ.     ADC #'Z-SPACE
1470 8109 6920   SPACE.  ADC #SPACE
1480 810B D0B2           BNE J:OUT          Print it.
1490               ;
1500 810D 8433   DECNRM  STY BYTE+1         Divide WORD by BYTE
1510 810F 8532           STA BYTE
1520 8111 A000           LDY #0             Result returned in Y
1530 8113 38             SEC
1540 8114 A530   DECNL1  LDA WORD
1550 8116 38             SEC
1560 8117 E532           SBC BYTE
1570 8119 AA             TAX
```

```
1580 811A A531           LDA WORD+1
1590 811C E533           SBC BYTE+1
1600 811E 90D3           BCC DECSTR
1610 8120 8531           STA WORD+1
1620 8122 8630           STX WORD
1630 8124 C8             INY
1640 8125 B0ED           BCS DECNL1          branch
1650                     ;-------------------------------------------------
1660 .                   ; TEW:M :: Transfer E:0:W to MEM
1670                     ;
1680 8127 A518   TEW:M   LDA E:0:W
1690 8129 A619           LDX E:0:W+1
1700 812B 8635   SAXMEM  STX MEM+1
1710 812D 8534   SAMEM   STA MEM
1720 812F 60     SARTS   RTS
1730                     ;-------------------------------------------------
1740                     ; SUB6CK :: Decrement MEM by 6 & compare w/SYMSTR
1750                     ;   .
1760 8130 A534   SUB6CK  LDA MEM
1770 8132 A635           LDX MEM+1
1780 8134 E408           CPX SYMSTR+1
1790 8136 D004           BNE SUB6K1
1800 8138 C50A           CMP SYMSTR
1810 813A F0F3           BEQ SARTS           Return BEQ if at end
1820 813C E906   SUB6K1  SBC #6              carry set
1830 813E CA             DEX
1840 813F B0EC           BCS SAMEM
1850 8141 90E8           BCC SAXMEM
1860                     ;-------------------------------------------------
1870                     ; INSERT :: Add SYMBOL to list & sort by alpha.
1880                     ;
1890 8143 A900   INSERT  LDA #0              FLAG indicates when a
1900 8145 8538           STA FLAG            swap is needed.
1910 8147 A518           LDA E:0:W
1920 8149 A619           LDX E:0:W+1
1930 814B 8637   SAXM2   STX MEM2+1
1940 814D 8536   SAM2    STA MEM2
1950 814F A438           LDY FLAG            If we know we need to
1960 8151 D030           BNE INSL.1          swap, don't waste time
1970 8153 C8             INY                 checking for it!
1980 8154 B136           LDA (MEM2),Y        Y=1
1990 8156 C53B           CMP FIELD+1         msb, first 3 char.
2000 8158 D027           BNE TEST
2010 815A 88             DEY
2020 815B B136           LDA (MEM2),Y        Y=0
2030 815D C53A           CMP FIELD           lsb, first 3 char.
2040 815F D020           BNE TEST
2050 8161 A53D           LDA FIELD+3         msb, second 3 char.
2060 8163 C9FF           CMP #$FF            Handle undef. symbols
2070 8165 D002           BNE YDEF              .
2080 8167 A53F           LDA VALUE+1
2090 8169 8539   YDEF    STA TEMP
2100 816B A003           LDY #3
2110 816D B136           LDA (MEM2),Y
2120 816F C9FF           CMP #$FF
2130 8171 D004           BNE YDEF2
2140 8173 A005           LDY #5
2150 8175 B136           LDA (MEM2),Y
2160 8177 C539   YDEF2   CMP TEMP
2170 8179 D006           BNE TEST
2180 817B A002           LDY #2
2190 817D B136           LDA (MEM2),Y        lsb, second 3 char.
2200 817F C53C           CMP FIELD+2
2210 8181 9011   TEST    BCC INSL.2          Alpha compare
2220 8183 A005   INSL.1  LDY #5              Swap routine...
2230 8185 B136   INSL.3  LDA (MEM2),Y
2240 8187 AA             TAX
2250 8188 B93A00         LDA FIELD,Y
2260 818B 9136           STA (MEM2),Y
2270 818D 963A           STX FIELD,Y
2280 818F 88             DEY
2290 8190 10F3           BPL INSL.3
2300 8192 8438           STY FLAG            Y=$FF -- always swap now
2310                     ;
2320 8194 A536   INSL.2  LDA MEM2            Check if done:
2330 8196 A637           LDX MEM2+1            If so, RTS.
2340 8198 E435           CPX MEM+1             Else, move to next
2350 819A D004           BNE INSL.4             symbol and continue.
2360 819C C534           CMP MEM
2370 819E F08F           BEQ SARTS
2380 81A0 E906   INSL.4  SBC #6              carry set
2390 81A2 B0A9           BCS SAM2
2400 81A4 CA             DEX
2410 81A5 90A4           BCC SAXM2           branch
```

OSI SIG Data Library
Where the Megabytes Bite

This is the part of OSI SIG where we keep program and text files. Like the other areas of the SIG, the Data Library is devided up into sections with each section dealing with a particular topic. All of the sections in the Data Library directly correspond to the sections in the message base. For example, section 0 is our "General" topic section. For the Data Library, we use it to hold text files which describe the various parts and functions of OSI SIG. Section 1's topic is OS-65D and all of the files in that section of the Data Library refer to that operating system. A full description of the topic of each section in the SIG Data Library is available by entering "DES" at the "DLx:" prompt in the Data Library (where "*" is the number of the section involved).

One thing that is important to note up front is that the SIG Data Library can be used in two different ways or "modes", as they are often called. The default mode is called the Menu Mode. In the Menu Mode, the primary commands that are available to you are displayed on a menu and you can select them by number. The other mode is the command mode. In the command mode, you enter the actual command. The benefit of the command mode is that it is much faster. You don't have to wait for the menus to be displayed before and after each command. However, the command words are acceptable in either mode. See the "SET" command below for details on selecting a mode.

The first thing you're likely to want to do in the Data Library is to find out what files are available there. There are two commands available to you which will display a list of the files in a section - BROWSE and SCAN. All commands in the Data Library may be abbreviated to the first three letters (or sometimes less) of the command. From now on, when a command is referenced, the portion of the command that is an acceptable abbreviation will be in capital letters and the remainder will be in lower case. For exampe, "Read" would indicate that "R" alone would be acceptable as an abbreviation. Now then, back to Scan and BROwse:

Scan

The Scan command allows you to examine the contents of the SIG Data Library. The format is:

S NAME.EXT[User ID]/option/option...

Each file is listed in the following form:

NAME.EXT DD-MM-YY * nr

where * = size of the file in bytes and nr = number of times the file has been retrived. If the file has the extension .BIN or .IMG, the size (*) will be followed by the approximate down-loaded size.

The order of listing is a function of the option(s) used. The default sequence is in inverse-order of submission date (i.e., most recent first). If you use a file name, or file name with wild cards, then the order is alphabetical by file name. If the /key option is used, the files appear in no particular sequence.

The simplest form is:

S

which will give a brief list of all files. The NAME.EXT may have "wildcards" in them, where "*" in either the NAME or .EXT positions signifies any file will match the "*". A "?" may be used to mean any letter/digit will match in that specific position. For example:

S *.bas

will find any file with an extension of "BAS" in any User ID.

S abc???.xy?

will match any file whose name begins with abc and whose extension begins with "xy"; also, any User ID will match. The form:

S[User ID]

will match any file submitted by that specific user.

The options allowed are:

/age:n - output only if the entry has been SUBmitted within the last n days.

/des - output the description of each file as given by the submittor.

/key:list - select only files which have the given set of keywords. The list may be a series of words separated by commas and/or spaces. If multiple keywords are supplied, there is an implicit "and" operation between them. An asterisk may be used to indicate the "tightness" of the search as follows:

xyz - an exact match with "xyz"
xyz* - any keyword which BEGINS with "xyz"
*xyz - any keyword which ENDS "xyz"
xyz - any keyword which CONTAINS "xyz"

For example:
/key:modem - finds files having the exact keyword "modem"
/key:modem* - finds files with "modem" or "modem7"
/key:*modem - finds files with "modem" or "smartmodem"
/key:*modem* - finds files with "modem", "modem7" or "smartmodem"

Note that keywords may consist of the following characters:
"A" to "Z" (or "a" to "z")
"0" to "9"
"+" "-" "#" "$" "." "_"

BROwse

The BROwse command is similar to the Scan command and accepts the same options. It forces a /des (description) option, and pauses after each file to give you a chance to:

Read, DOWnload, ERAse, or CHAnge

the file.

These options are displayed after each file is listed by the BROwse command. The Read option will type the file out for you. The DOWnload option will

automatically transfer the file to your computer if you are using Term-Plus, Term-32, Term-65U, or TERM-A.ASM as your terminal program. For details on these programs, read the file TPLUS.DOC in Section 0 of the Data Library. The ERAse option marks the file in a way that tells the SYSOP (me) that you want the file removed from the Data Library. You might want to do this if you found an error in the original file. You are only able to mark files for erasure that you have submitted to the Data Library. Lastly, the CHAnge option allows you to replace the keywords and descriptions you originally entered for a file. Again, CHAnge is only available when the file being examined is your own.

Read

The Read command allows you to see specific files. The format is:

R FILENM.EXT

Entering a control-P (hold down the "control" key and press P) causes the printing of the file to stop and you are returned to the top function level.

ERAse

The ERAse command is used to request the removal of one of your files from the Data Library. It is entered simply as:

ERA FILENM.EXT

When the SYSOP (me) recieves your request to have a file removed, he (I) will examine the file in question, and if he (I) agree that the file should be removed, he (I) will do so. Therefore, it is a good idea to leave the SYSOP a message explaining why you requested that the file be removed.

UPLoad

The UPLoad command is used to directly transfer files from your computer to the SIG Data Library Reference Library. UPLoad is most often used like the DOWnload command, in that special terminal programs like Term-Plus will perform the transfer automatically, and additionally, these programs do error-checking along the way to insure that the file is properly transferred. If you do not have a program like Term-Plus, you may instead either type the file in by hand or have your computer "LIST" the file. If you use this second method, you must enter a <CTRL>Z' when you (or your computer) are (is) finished entering or LISTing the file to tell CompuServe that you are finished. The command format is:

UPL FILENM.EXT

where "FILENM.EXT" is the name of the file for the CompuServe Data Library database. You will then be prompted for the file specification on your personal computer which is to be uploaded.

The ".ext" has two special forms as follows: ".BIN" is used for "binary" (ie, 8-bit) data. In OSI SIG, ".BIN" files are used to hold machine code programs. ".IMG" is used for "image" data. Image data is similar to binary, except that it carries with it an identification of the kind of computer from which it came. OSI systems do not use the ".IMG" extension. If you use any other three letter extension in your file name, it will be stored as a 7-bit text file.

After entering the UPL command, you will be prompted for some additional information. First, you will be prompted for a list of "keywords" which users may use with the Scan command to find your file. Please use keywords which identify what you are submitting. Using obscure and inconsistent keywords will hinder other users who are trying to locate your file. For example, if you are submitting an Adventure-type game written in Microsoft basic, you might use the keywords "adventure, game, mbasic."

Note that keywords may be made up of the following characters:

"A" to "Z" (or "a" to "z")
"0" to "9"
"+" "-" "*" "$" "." "_"

All other characters are removed. The non-alphanumeric characters above should only be used as part of the keyword, NOT as separators. For example:

CPM2.2
or CPM+
are acceptable, but
TEST-FILE
or GRAPHICS_PROGRAM*18
are not.

In general, the keywords should be descriptive of the file's CONTENT and not used as a title. You will then be asked for a short description of the file. You will be limited to about 500 characters, or nearly a full 32 x 16 page. If the file is a program, the description should include the models of computers that the program will run on and brief instructions on its use.

DOWnload

The DOWnload command is the same as the DOWnload option of the BROwse command listed above. The only difference is that you are requesting a specific file rather than picking one as you are going through the SIG Data Library Reference Library. Here again, the benefit of being able to use the DOWnload command is that the transfer is done automatically and without error so that the effects of phone line noise and other problems are minimized. Use of the DOWnload command requires a program like Term-Plus. The format of the command is:

DOW FILENM.EXT

SET

The SET command is used to control certain operating characteristics of the Data Library during your visit. The following options are available for the SET command:

BRIEF - shortens some prompts
NO BRIEF - normal prompts
MENU - use menu mode
NO MENU - use command mode
PAUSE - pauses when screen full
NO PAUSE - doesn't

The SET command without options will display your current settings.

4x4 Character Set Listing 1

```
900 !*********************************************
904 !*** DATA FOR CHALLENGER 4x4 CHARACTER SET ***
906 !*** EACH 4x4 CHARACTER IS COMPOSED OF 16 ****
908 !*** CHARACTERS FROM THE CHALLENGER GRA- *****
910 !*** PHICS CHARACTER SET. COMPONENTS ARE *****
912 !*** ORDERED IN ROWS, READING FROM LEFT- *****
914 !*** TO-RIGHT, WITH THE TOP ROW FIRST. *******
916 !*********************************************
918 !
1000 :DATA_4x4
1032 : 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
1033 : 32 157 32 32 32 157 32 32 32 166 32 32 32 166 32 32
1034 : 32 156 156 32 32 168 168 32 32 32 32 32 32 32 32 32
1035 : 32 156 156 32 166 175 175 32 166 175 175 32 32 168 168 32
1036 : 32 176 154 32 166 176 167 32 165 176 170 32 32 166 32 32
1037 : 157 156 165 32 32 165 168 32 165 168 154 32 32 32 155 32
1038 : 165 169 32 32 166 170 32 32 157 166 170 32 32 155 166 32
1039 : 32 157 32 32 32 166 32 32 32 32 32 32 32 32 32 32
1040 : 32 170 32 32 157 32 32 32 166 167 32 32 32 166 32 32
1041 : 32 166 167 32 32 32 157 32 32 32 170 32 32 166 32 32
1042 : 165 157 165 32 32 177 168 32 165 177 169 32 32 166 32 32
1043 : 32 165 32 32 165 170 154 32 32 157 32 32 32 32 32 32
1044 : 32 32 32 32 32 32 32 32 32 157 32 32 32 168 32 32
1045 : 32 32 32 32 32 165 154 154 32 32 32 32 32 32 32 32
1046 : 32 32 32 32 32 32 32 32 32 32 32 32 32 32 166 32 32
1047 : 32 32 165 32 32 165 168 32 165 168 32 32 32 32 32 32
1048 : 165 155 169 32 157 165 161 32 157 168 157 32 32 155 168 32
1049 : 32 176 32 32 32 157 32 32 32 157 32 32 32 155 168 32
1050 : 165 155 169 32 32 165 170 32 165 168 32 32 166 155 155 32
1051 : 166 155 161 32 32 165 156 32 165 32 157 32 32 155 168 32
1052 : 32 165 156 32 165 168 156 32 166 155 175 32 32 32 168 32
1053 : 157 155 155 32 166 155 169 32 165 32 157 32 32 155 168 32
1054 : 32 170 155 32 157 154 167 32 157 32 157 32 32 155 168 32
1055 : 166 155 161 32 32 165 156 32 32 156 32 32 32 168 32 32
1056 : 165 155 169 32 166 154 170 32 157 32 157 32 32 155 168 32
1057 : 165 155 169 32 166 154 176 32 32 32 170 32 166 155 32 32
1058 : 32 32 32 32 32 32 166 32 32 32 32 166 32 32 32 32 32
1059 : 32 32 32 32 32 32 166 32 32 32 157 32 32 32 168 32 32
1060 : 32 165 168 32 165 168 32 32 32 169 32 32 32 32 168 32
1061 : 32 32 32 32 166 155 155 32 166 155 155 32 32 32 32 32
1062 : 32 169 32 32 32 32 169 32 32 165 168 32 32 168 32 32
1063 : 165 155 169 32 32 165 168 32 32 166 32 32 32 166 32 32
1064 : 165 155 169 32 157 157 157 32 157 166 168 32 32 155 155 32
1065 : 32 170 167 32 157 32 157 32 157 155 161 32 166 32 166 32
1066 : 157 155 169 32 157 154 170 32 157 32 157 32 166 155 168 32
1067 : 165 155 169 32 157 32 32 32 157 32 165 32 32 155 168 32
1068 : 157 155 169 32 157 32 157 32 157 32 157 32 166 155 168 32
1069 : 157 155 155 32 157 154 167 32 157 32 32 32 166 155 155 32
1070 : 157 155 155 32 157 154 167 32 157 32 32 32 166 32 32 32
1071 : 165 155 155 32 157 32 32 32 157 32 177 32 32 155 155 32
1072 : 157 32 157 32 157 154 176 32 157 32 157 32 166 32 166 32
1073 : 32 177 168 32 32 157 32 32 32 157 32 32 32 155 168 32
1074 : 32 32 157 32 32 32 157 32 165 32 157 32 32 155 168 32
1075 : 157 32 170 32 157 170 32 32 157 166 167 32 166 32 166 32
1076 : 157 32 32 32 157 32 32 32 157 32 32 32 166 155 155 32
1077 : 157 167 176 32 157 157 157 32 157 32 157 32 166 32 166 32
1078 : 157 32 157 32 157 169 157 32 157 32 161 32 166 32 166 32
1079 : 165 155 169 32 157 32 157 32 157 32 157 32 32 155 168 32
1080 : 157 155 169 32 157 154 170 32 157 32 32 32 166 32 32 32
1081 : 165 155 169 32 157 32 157 32 157 166 170 32 32 155 166 32
1082 : 157 155 169 32 157 154 170 32 157 166 167 32 166 32 166 32
1083 : 165 155 169 32 166 154 167 32 165 32 157 32 32 155 168 32
1084 : 166 161 155 32 32 157 32 32 32 157 32 32 32 166 32 32
1085 : 157 32 157 32 157 32 157 32 157 32 157 32 32 155 168 32
1086 : 157 32 157 32 157 32 157 32 166 167 170 32 32 166 32 32
1087 : 157 32 157 32 157 165 157 32 157 170 161 32 166 32 166 32
1088 : 157 32 157 32 32 169 168 32 165 168 169 32 166 32 166 32
1089 : 157 32 157 32 32 169 168 32 32 157 32 32 32 166 32 32
1090 : 166 155 161 32 32 165 168 32 165 168 32 32 166 155 155 32
1091 : 157 155 168 32 157 32 32 32 157 32 32 32 166 155 168 32
1092 : 165 32 32 32 32 169 32 32 32 32 169 32 32 32 32 32
1093 : 166 155 156 32 32 32 156 32 32 32 156 32 166 155 168 32
1094 : 32 32 32 32 32 170 167 32 166 32 166 32 32 32 32 32
1095 : 32 32 32 32 32 32 32 32 32 32 32 32 166 155 155 32
```

4x4 Character Set Listing 2

```
4 ! ****************************************
5 ! *** PROGRAM DISPLAYING TEST MESSAGE ***
6 ! *** TO VIDEO SCREEN ILLUSTRATING USE **
7 ! *** OF CHALLENGER 4x4 CHARACTER SET ***
8 ! ****************************************
9 !
10 CALL SHOW_4x4 aB % 10,10
20 CALL SHOW_4x4 aE % 10,14
30 CALL SHOW_4x4 aT % 10,18
40 CALL SHOW_4x4 aA % 10,22
50 CALL SHOW_4x4 a& % 10,26
60 CALL SHOW_4x4 a6 % 10,30
70 CALL SHOW_4x4 a5 % 10,34
80 END
90 !
100 SUBR SHOW_4x4 label % line,column
110 REF label
120 FOR I=0 TO 3
130 FOR J=0 TO 3
140 READ X
150 PRINT AT(line+I*64+column+J),CHR$(X)
160 NEXT J
170 NEXT I
180 RET
998 !
1047 :a& 32 32 165 32 32 165 160 32 165 160 32 32 32 32 32 32
1053 :a5 157 155 155 32 166 155 169 32 165 32 157 32 32 155 160 32
1054 :a6 32 170 155 32 157 154 167 32 157 32 157 32 32 155 160 32
1065 :aA 32 170 167 32 157 32 157 32 157 155 161 32 166 32 166 32
1066 :aB 157 155 169 32 157 154 170 32 157 32 157 32 166 155 160 32
1069 :aE 157 155 155 32 157 154 167 32 157 32 32 32 166 155 155 32
1084 :aT 166 161 155 32 32 157 32 32 32 157 32 32 32 166 32 32
```

BASIC/DOS Interface Code for OS-65U

I wrote this program a long time ago. All it does is to allow you to read or write specific sections of any floppy disk to or from any RAM address in memory. I wrote it to be able to pull sectors off 65U disk into memory so that I could store them on 65D disks for disassembly and other purposes. It can also be helpful for repairing files in extreme emergencies.

However, it also shows the essential elements of the BASIC/DOS interface code that is built into OS-65U. You'll see variations on this code in most of the OS-65U utility programs such as DIR, CREATE, and DELETE. The essential premise behind the code is to allow the BASIC programmer to execute low-level disk operations and to have the results of those operations be made available to the program.

Disk operations are routed through BASIC's USR(X) function. The function returns the result of the operation. A result of 0 means no errors occurred. Any other value is the disk error number.

```
10 REM- *** OS-65U DISK READ/WRITE UTILITY      ***
40 :
60 :
70 UL=PEEK(8778) : UH=PEEK(8779)
80 :
90 REM- DISABLE <CTRL> 'C' CHECKING AND SAVE CURRENT STATUS
100 :
110 CC=PEEK(2073) : POKE 2073,96
120 :
130 REM- SET UP DOS READ/WRITE VECTOR
140 :
150 POKE8778,192 : POKE8779,36  :REM- $24C0
160 :
170 REM- SET UP ISR PUT IN SUBROUTINE
180 :
190 POKE 9432,243 : POKE 9433,40
200 :
210 REM- SET UP ISR GET IN SUBROUTINE
220 :
230 POKE 9435,232 : POKE9436,40
240 :
250 CB=9889:REM- CONTROL BLOCK  $26A1
260 :
270 Q=256:REM- ONE PAGE
280 :
290 REM- GET DISK ADDRESS FROM USER
300 :
310 INPUT"ENTER DISK ADDRESS FOR READ/WRITE";DA
320 :
330 REM- GET NUMBER OF BYTES FROM USER
340 :
350 PRINT"HOW MANY BYTES ARE TO BE READ/WRITTEN":GOSUB900:NB=A
360 :
370 REM- GET RAM ADDRESS FROM USER
380 :
390 PRINT"FOR THE MEMORY ADDRESS":GOSUB900:RA=A
400 :
410 REM- GET OPERATION TYPE FROM USER
420 :
430 INPUT"READ OR WRITE (R/W)";RW$: RW$=LEFT$(RW$,1)
440 IF RW$ <> "R" AND RW$ <> "W" THEN PRINT : GOTO 430
450 :
```

I have published parts of this program before, but I wanted to use it again to help point out some details I haven't previously discussed.

As I mentioned, the various 65U utilities often use this procedure. Line 70 saves the user's original USR(X) vector so that it can be restored on exiting. That's just good practice since you can never be sure if the user is running in an environment that depends on some machine code that is already installed, but only sets the pointers upon installation. Line 110 saves the incoming <CTRL>'C' enable status, and turns it off. The same principle applies here.

Lines 130-230 set up pointers in the interface subroutine within OS-65U. This is largely a precautionary measure since these pointers are normally restored by any program that disturbs them, but when you're doing anything that could damage the contents of a disk, it's better to be safe. Since these pointers should be the default settings, we are under no obligation to save and restore their incoming values. Next time we'll pull apart the code itself and discuss how it operates.

I hard-coded the program to only operate on DEVice "A" on purpose. The program would operate on hard disks, but don't do it unless you are *very* confident that you know what you're doing. In any event, I hope you find the program useful.

```
460 REM- CHECK DA, RA, AND NB FOR VALIDITY
470 :
480 IF DA < 0 OR DA > 275967 THEN 310
490 IF RA < 0 OR RA > 65536 THEN 390
500 IF NB < 0 OR NB > 65536 THEN 350
510 :
520 REM- NOW PERFORM CALCULATIONS FOR OPERATION
530 :
540 DH=INT(DA/16777216) : RM=DA-DH*16777216
550 DM=INT(RM/65536)    : RM=RM-DM*65536
560 DL=INT(RM/256)      : RM=RM-DL*256
570 :
580 POKECB+1,RM : POKECB+2,DL : POKECB+3,DM : POKECB+4,DH
590 :
600 POKECB+5,NB-INT(NB/Q)*Q : POKECB+6,INT(NB/Q)
610 :
620 POKECB+7,RA-INT(RA/Q)*Q : POKECB+8,INT(RA/Q)
630 :
640 REM- NOW DO IT
650 :
660 IF RW$ = "R" THEN RW = 0
670 IF RW$ = "W" THEN RW = 1
680 :
690 DEV "A"
700 :
710 ER = USR (RW)
720 :
730 REM- CHECK FOR ERRORS
740 :
750 IF ER THEN GOSUB870
760 :
770 REM- RESTORE USER'S USR(X) VECTOR
780 :
790 POKE 8778,UL : POKE 8779,UH
800 :
810 REM- RESTORE OLD <CTRL> 'C' STATUS
820 :
830 POKE 2073,CC
840 :
850 END
860 :
870 PRINT"*** DEVICE A ERROR #";ER;" AT ADDRESS";DA
880 PRINT:RETURN
890 :
900 PRINT"ENTER THE DECIMAL VALUE OR HEX VALUE PRECEEDED"
910 INPUT"BY A '$' ";A$:IFLEFT$(A$,1)="$"THEN940
920 FORX=1TOLEN(A$):C$=MID$(A$,X,1):IFC$<"0"ORC$>"9"THEN900
930 NEXTX:A=VAL(A$):RETURN
940 A=0:IFLEN(A$)<2THEN900
950 FORX=2TOLEN(A$):C$=MID$(A$,X,1):IFC$<"0"THEN900
960 IFC$<="9"THENA=A+VAL(C$)*(16^(LEN(A$)-X)):GOTO990
970 IFC$<"A"ORC$>"F"THEN900
980 A=A+(ASC(C$)-55)*(16^(LEN(A$)-X))
990 NEXTX:RETURN
```

DISK LABEL MAKER FOR HOOKS
By: Jack Noble (72737,100)
746 N. 165th St.
Seattle, WA 98133

Here's a little 10 liner that has saved me a lot of aggravation in keeping track of just what's on which disk. It prints the disk directory in four columns in condensed print onto sticky backed address labels available at Radio Shack. There's really not much to the program since it makes use of 'HOOKS' directory format which is in four columns anyway. To use the program you load the fanfold labels into your printer and run the program. Then put the first disk that you want a label for in the active drive and press any key. The label will be printed and the program will wait for you to insert the next disk after which you again press any key. As you can see from the sample, you can print a directory of up to 20 files per disk on one of these address labels. The control codes given are for an EPSON MX-80 printer--you should adjust these as required for your printer. I stick the labels right on the disk cover as they come out of the printer so I no longer have to worry about mixing up jackets or loosing the loose directory printouts that I used previously.

```
5 REM**DIRECTORY LABEL MAKER**
10 POKE$B6B8,$4C:POKE$B6B9,$CC
15 POKE$B6BA,$B6:REM NO BANNER
20 DISK!"IO ,0A:REM #4 PRINTER
30 PRINTCHR$(15):REM CONDENSED
40 PRINTCHR$(27);"C";CHR$(6)
50 CALL$252B:REM WAIT FOR KEY
60 D*:REM PRINT DIRECTORY
70 PRINTCHR$(12);:REM FORMFEED
80 GOTO 50:REM DO ANOTHER
```

```
OS65D3 00-06  TERM+  07-11  DIR$   12-12  OS65D  13-13
PRINT  14-14  BASIC  15-16  BEXEC$ 17-18  CHGPAS 19-20
PGMKEY 21-21  MODSET 22-23  PGMFUN 24-24  XFER   25-26
CNVRT  27-29  BINRUN 30-30  CBMODE 31-31  LOG ON 32-32
MSGTRE 33-34  FILGE  35-37
```

Letters to the Editor

Editor;

I just finished looking through the latest issue of PEEK and decided it was time for me to send my reader survey form. I have had the letter hand written for quite a while now, but never got around to typing it in the computer.

I agree with your view of the OSI video board being a stumbling block to new software. I purchased a Generic Color Plus video board about a year ago and have been pleased with it. The main advantage to this board is that it works along with the OSI 540 board and it only takes a couple of bytes of memory. However, as I indicated in the survey, I would like to see more software available that utilizes this board.

If a new graphics board is designed, I would like to see a board similar to the Color Plus but with an 80 column display and 640x400 pixels. If we are going to do something, we might as well go all the way. I would like to see possibly a software package sold with the board. A bare board would be fine with me.

As for ideas on a new operating system, here are my ideas: (1) Include a WINDOW command that would allow you to jump back and forth between windows. Also have the command put a box around the window. (2) Include the Color Plus code. (3) Include the BSR X-10 code for the home control system. (4) Include a CALL statement to call different machine code programs from BASIC without having to reset the pointers for the USR(X) function. (5) If you are familiar with the CA-20 board and manual, they show some commands in the manual from something called Process Control BASIC. OSI said this BASIC was never finished but some of the commands would be nice. These included a TIME and DATE command for the on-board clock. There were also commands for the CA-22 board (analog/digital converter board) which I use with my temperature probe program. (6) Include a full-screen editor. The CEGMON ROM had a nice editor and also a good window system. (7) A screen dump to printer would be nice also.

Good luck on all your software projects and hope to talk to you on CompuServe.

John Schneider
326 Chestnut Street
Wheeling, WV 26003

Dear John,

Thanks for all of the suggestions. The video board problem is going to be a tough one to crack. Through PEEK, I have been trying to inspire several people to design a new board that will see us through the foreseeable future, but there is nothing imminent.

One crucial element in the design of such a board is the resolution. It's certainly going to have to be capable of 80 columns for it to gain widespread support in the OSI community. After all, people aren't going to be willing to shell out a lot of money for a new board and put up with some inevitable software incompatibilities unless there are substantial gains to be had. I simply don't know enough about the hardware to make any concrete suggestions. All I can do is point out what I consider to be minimal design goals.

If you hardware wizards are listening, please remember that the OSI video community is largely made up of people who are using televisions and inexpensive monitors. Please make sure that anything you design is capable of composite video output. If we make the upgrade too expensive, it will never take hold.

Speaking of the video community, as I mentioned in the article on the User Survey, serial system owners almost universally said they didn't care about a graphics board. I think this is likely due to the fact that they see no benefits to a second display just for occasional graphs. However, since replacing the 540 board on video systems would also necessarily mean replacing the keyboard interface, this would seem to me to be a most opportune time to make it possible for OSI users to attatch one of the replacement keyboards for the IBM PC's which have been so widely praised. If we could produce a combination video/keyboard upgrade, we would be vastly increasing the size of the potential market for this hardware and thus lowering the costs to all of us - not to mention making a quantum leap forward possible in the software.

Rick

Editor:

While I was filling out (the User Survey), an idea occurred to me. On the form, I requested that you publish topics concerning changes to OS-65D, but I suddenly thought that while I know 65D fairly well, I know absolutely nothing about OS-65U. Is it possible that what I really want is already in OS-65U?

Anyway, maybe a brief descriptions of the different operating systems and a list of the different variations that exist for each might be an interesting topic for PEEK. If possible, could the discussion for each system include hardware requirements, features, and peripherals supported?

A little history of my machine might help explain my request: I ordered my machine as a Challenger with 12K, paper tape BASIC, and 430 cassette board in June 1977. What was delivered was one of the first Challenger II's (500 CPU, three 420C boards, and 430 cassette). Over the years, I upgraded the machine with video (first a 540, then a 540B-1) and homemade keyboard, then added a 470-110 disk kit. Then I upgraded the 65F and 65A ROMs to a SYGMON ROM plugged into one of the BASIC ROM locations with the required decode logic on a piggyback board. I now have two 8" drives, the original GSI 110 as drive B and a Siemens FDD100-8 as drive A. I've also added a D&N BIO-1600 with serial, paralell, memory, and a battery backed-up clock (on the Diablo port), a D&N MEM-CM9 board with memory only, and a 2K block of memory on a 420C

board addressed at $E800.

(As far as software is concerned), I've gone from paper tape BASIC and assembler to a home-grown cassette tape block transfer program, to a tape operating system (??) written locally by another OSI user, to OS-65D V1.0 (with handwritten directories) to V2.0, to V3.x, and finally to V3.3 last year.

My dealer moved away about 8 years ago to become the west coast distributor for OSI, and then left OSI entirely when MA/COMM bought OSI. There are some questions I need answered and few places to get them answered.

I've tried some of the software that came with OS-65D V3.3 and some of it doesn't work. For instance, (using) the MODEM program after changing the ACIA address to match mine, I consistantly drop every other character at 300 baud. I think that most newer machines run at 2 MHz while mine is old and runs at 1 MHz (I've tweeked iit and found that it runs reliably at 1.4 MHz and fails at 1.7 MHz. I tried a 6502B, but can't get it to boot at any speed, even .9 MHz.I think I have a couple of slow memories or address decode chips).

The OS-65D V3.3 printer driver drives me batty. I wrote my own driver in the $E800 2K to perform the skip over perforations, but I can't figure out how to defeat the built-in driver. It doesn't ever pass the <CTRL>'C' to the printer. This makes it hard to sub and superscript and still have each page start at the right place.

Where is, and how does the new keyboard driver live and work? I again had my own to handle uppoer and lower case and <CAPS LOCK>. My keyboard also has 63 keys, not 54, so while I can handle it with V3.2, 3.3 expects other codes and my driver won't work with 3.3. 3.3 is also inconsistent between BASIC, and the Assembler/Editor-Extended Monitor, the lower case only works with BASIC.

I like my OSI, but after using a PC-clone at work I miss some of the features that MS-DOS has such as open files on both drive A and drive B at the same time and dynamic file creation without running CREATE or including the same code in each program.

Sincerely,
Alan G. Albright
2935 Hypoint Avenue
Escondido, CA 92027

Dear Alan,

Thanks for all your comments. To answer some of your questions, the leap from OS-65D to OS-65U isn't as great as many people perceive. The two share many fundamental design principles. The core of the BASICs in both operating systems is identical, making the transition fairly easy once you get familiar with the way OS-65U handles data files. My series last year on this topic should help clear that up. OS-65U does answer your prayer for the ability to have files open simultaneuosly on different drives, though. So do look into it.

I think you're probably right in suspecting that your problems with OS-65D V3.3 stem from your non-standard hardware. However, I don't think you've gone so far afield as to make it impossible to overcome them.

The MODEM program that comes with OS-65D is very simple. For it to be dropping characters at 300 baud is extremely unusual. I can't tell from your letter where the problem might be. Most of my problems in this area stem from the slowness of the keyboard polling software in 3.3. At 1 MHz, I can see where you may really run into trouble with it. My advice would have to be to try to find a copy of the 3.2 version of that program (which OSI published in a couple of places) or port the 3.3 version to 3.2. The latter will require disassembling the machine code, but it's short and the only change you'd have to make would be to change the JSR's to the input and output routines.

Under V3.3, the keyboard poll is located at $3590. Higher up in the operating system, the OS-65D dispatch table still points to the old address of $252B, but from there 3.3 merely JSR's to a JMP to the real location stated above. Going directly to $3590 will have no effect on any software except for saving a few milliseconds.

I don't know why you're having so much trouble with the automatic paging under 3.3. I have found, however, that by NOT initializing it with the PRINT#1,!(??,??) command (sorry, I forget the code) that I can position the paper in my printer with absolute accuracy. Try just cold booting and see how it works without intervention. The code for this resides within the old keyboard polling routine slightly above $252B, but if you get that deep you'll also want to check the latches in the OUTCH routine at $2343.

As far as the ASM/EM not accepting lower case, you're absolutely correct, although neither of those programs would benefit greatly from the ability. Oh sure, lower case in assembly language programs can be helpful, but not crucial. Thankfully the rest of the OS is case-blind.

Rick

Last Call on Backissue Sale

The backissues of PEEK[65] contain a wealth of information not available anywhere else at any price. From cassette systems to multi-user hard disks, PEEK has been the source of innovative support to the OSI community since 1980.

If your library of PEEK backissues is incomplete, now is the time to fill in the holes in your collection. Backissues are available from January 1981 to date. Full year backissues cost $6.00 per set plus $3.00 shipping. Single issues are $1.00 each plus $.75 each. For multiple set orders, reduce shipping per set by 50%. Order today. This sale ends September 30, 1986.

SOFTWARE FROM PEEK!

Term-Plus
A smart terminal program running under OS-65D V3.3 which allows capturing and transmitting to and from disk. Term-Plus also supports error-free file transfers and cursor addressing on CompuServe. Memory size does not limit the size of files that can be captured or transmitted. Video systems get enhanced keyboard driver with 10 programmable character keys. 10 programmable function keys on both serial and video systems. Utilities included allow translating captured text files into OSI source format for BASIC and Assembler programs or into WP-2/WP-3 format, translating OSI source files into text files for transmitting to non-OSI systems, and printing captured text files. Runs on all disk systems, mini's or 8", except the C1P-MF. $35.00.

Term-32
Same as Term-Plus, but for OS-65D V3.2. Video system support includes enhanced keyboard driver, but uses V3.2 screen driver. $35.00.

Term-65U
Patterned after Term-Plus, Term-65U is a smart terminal program for OS-65U (all versions) running in the single user mode. Allows capturing text to disk files. Term-65U will transmit text files, or BASIC programs as text. The program will also send WP-3/Edit-Plus files as formatted text and can transmit selected fields in records from OS-DMS Master files with sorts. Includes utility to print captured text files and convert them into WP-3/Edit-Plus files for editing. $50.00

ORDER TODAY!

ASM-Plus
ASM-Plus is a disk-based assembler running under OS-65D V3.3 that allows linked source files enabling you to write very large programs, regardless of system memory size. ASM-Plus assembles roughly 8 to 10 times faster than the OSI Assembler/Editor and is compatible with files for that assembler. ASM-Plus adds several assembly-time commands (pseudo-opcodes) for extra functionality. Included is a file editor for composing files that allows line editing and global searches. $50.00

Edit-Plus
Word processor styled after WP-3-1, although not quite as powerful. Edit-Plus allows composing and editing WP-3 compatible files and to have those files printed as formatted text. Edit-Plus uses line-oriented editing, as opposed to the screen editing of WP-3, and also allows global search and replace. Edit-Plus fixes problems in WP-3 including pagination, inputs from the console, and file merging (selectable line numbers from the merged file). Edit-Plus can perform a trivial right-justification, but it does not support true proportional spacing. Requires OS-65D V3.3. $40.00

Data-Plus 65U Mail Merge
A program to insert fields from OS-DMS Master files into WP-3 documents. Output can be routed to a printer or to a disk file for printing later or for transmission via modem using Term-65U. Insertions are fully selectable and are properly formatted into the output. Perfect for generating form letters. $30.00

Data-Plus Nucleus
Data-Plus Nucleus is a replacement package to the OS-DMS Nucleus from OSI. All of the programs from the original except SORT have been duplicated and enchanced and new software, the MC-DMS Interface, has been added. The name "MC-DMS" stems from the extensive use of machine code support built into the utilities to replace slower, BASIC code. Features include; (1) MC-DMS Interface code supports up to 8 Master files simultaneously without requiring OPEN/CLOSE commands under Level 3 at every file access. The only 65U software support needed for Level 3 file access is semiphores. This produces a significant increase in speed. READ, WRITE, and FIND commands operate on the field level. FIND skips over embedded garbage between fields eliminating the need for embedded blanks, and automatically stops on the last record in the file. (2) Machine code DIR utility. Ultra-fast. Automatic paging. ^C interrupt. Can selectively list by file type or can search for file name matches with wildcards. (3) Machine code file manager. Creates, deletes, or renames files in a flash. The file manager is linked to the Master/Key file creation utility. (4) Machine code file transfer/merge. Grabs up to 30 records per pass. Single/dual drive. Fully selectable field specifications. Also allows searching for matches in source and destination files for linked merges. (5) Machine code single/dual drive floppy diskette copier. Moves up to 7 tracks per pass. (6) Disk-based mailing label printer. Stores printing format designs on disk. Selectable fields and record range, Key file access, searches, and more. (7) Disk-based report writer. Stores report format designs on disk. Same features as above, but with formatted columns by type and width. (8) Edit-Plus 65U. Most of the same features as the 65D version. Suitable for correspondence and form letters. (9) Data-Plus Mail Merge. Complete documentation allows implimenting the MC-DMS Interface into your own applications. $150.00

SHORTEST HEX/DEC-DEC/HEX GOSUB VERSION

By: R. N. Hislop
5B Awatea Street
Porirua, New Zealand

As a follow-up to my HEX/DEC: DEC/HEX conversion that you published in the Dec. '84 issue, here is an even shorter version which is excellent for use in GOSUBs, and a Stand-Alone version too.

STAND-ALONE PROGRAM

```
0 RUN3
1 L=ASC(H$)-48:L=L+7*(L>9):
  N=N*16+L:H$=MID$(H$,2):
  IFH$GOTO1
2 A=INT(D/16):B=D-A*16:H$=
  CHR$(B-7*(B>9)+48)+H$:D=A:
  IFDGOTO2
3 PRINT"***DEC="N"HEX="H$:
  PRINT:INPUT"HEX,DEC";H$,D:
  N=0:PRINT:GOTO1
```

GOSUBs SHORTEST HEX/DEC or DEC/HEX Conversions?

```
0 RUN10
1 L=ASC(R$)-48:L=L+7*(L>9):
  N=N*16+L:R$=MID$(R$,2):
  IFR$GOTO1
2 RETURN
3 A=INT(D/16):B=D-A*16:R$=
  CHR$(B-7*(B>9)+48)+R$:D=A:
  IFDGOTO3
4 RETURN
5 :
10 PRINT"LINEs 1 & 3 are used
   in normal way.  Have N=0:
   just
11 PRINT"prior to GOSUB1 and
   R$="": prior to GOSUB3.
12 PRINT"Do not use as vari-
   ables elsewhere in program
14 PRINT"L,R$,N,A,B,D
15 PRINT"LINEs 100 and 200
   just for testing. R$ and D
   would
16 PRINT"derive from program.
17 :
100 PRINT:INPUT"HEX=";R$:
    GOSUB1:PRINT,"DEC ="N:
    N=0:L=0:GOTO100
150 :
200 PRINT:INPUT"DEC=";D:
    GOSUB3:PRINT,"HEX="R$:
    R$="":B=0:GOTO200
```

```
RUN 10
LINEs 1 & 3 are used in normal
way.  Have N=0: just prior  to
GOSUB1  and  R$=:  prior  to
GOSUB3.  Do not use as vari-
ables elsewhere in  program
L,R$,N,A,B,D.  LINEs  100  and
200 just for testing.  R$  and
D would derive from program.
```

Watch This
Space Grow!

PEEK [65]
PO Box 586
Pacifica, CA 94044
415-359-5708

Bulk Rate
U S Postage
PAID
Pacifica, CA
Permit #92
Zip Code 94044

DELIVER TO:

GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

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

TOTAL	$_____
CA Residents add 6% Sales Tax	$_____
C.O.D. orders add $1.90	$_____
Postage & Handling	$ 3.70
TOTAL DUE	$_____

POSTAGE MAY VARY FOR OVERSEAS

Name _____

Street _____

City _____ State _____ Zip _____