PEEK (65)

The Unofficial OSI Users Journal

MARCH 1985
VOL.6, NO.3

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

INSIDE

Column One

The food and prices in Europe were great, but it is greater to be back. Aside from those delicacies that add girth, one highlight was a visit with David Livesay, of 68000 fame, at his home near Liege, Belgium. The package that he has put together (see his ad in the December '84 issue) is quite amazing. At this point, the 68000 is primarily handling the math functions for the 6502, but even here the speed-up is dramatic. Anyone who does a lot of math should have a second look. David's other contribution, Search for Line Number (January issue), is another gem. The speed improvement was dramatic. This should become a standard where speed is needed.

Now that we have given you the above solutions, a challenge. Who will write an article on the best way to manage the disk head lift on the 4P-MF - better yet, with motor shut-down? That shouldn't be difficult as I hear that several of you have done it.

Another challenge; to the WP-6502 hackers (see page 16). You have documented it, modified it and fixed it, but can you make it clean up after itself? Regrettably, it modifies the operating system which almost guarantees a "crash" when running the next program in a multi-user environment. Here's a chance for a Hacker to make a lot of business users stand up and notice the Hacker.

One last question. On page 23, you will find a piece on cottage industry activities. To broaden the field a bit, we are interested in what you are doing with your machines. Recently, I have talked with a number of you and have had my eyes opened by some of the things you are up to. Won't you please take the time to drop us a line and give us a profile and/or how your machine makes money.

In return for all the questions, here are some answers. Well, how about Brian Hartson's new series on OSI hardware - what it is, how it works and interrelates with the system, the shortfalls and opportunities for improvement. It is a complex subject, but the object is to bring the hardware neophyte up to speed. In these days, we certainly can use all the help and understanding we can get.

Rick Trethewey's final installment on Machine Language programming is probably the best of his nine articles. Best of all, even though this is the last, Rick says he has another trick or two up his sleeve.

For the "Elf" types and number crunchers, Puckett's regression package is the ultimate of its type. Its size and scope dwarf anything we know of for the OSI.

Graphics buffs will find some proof of the pudding in Earl Morris's follow up on the Color Plus Board. It is a shame we don't have color and motion on the printed page.

In the manufacturers corner, and I now count three of them, there is a hype of activity working feverishly toward totally new or variation machines (I count at least 5). New CPUs (yup! 68000), op systems, languages and utilities, but all running most, if not all, OS-U programs. All this bodes well for the OSI world, but for the moment we will just have to wait until next month for some of the specifics. There is fever in the air!

Lastly, recent innovations are making it almost possible to add high density MF drives, hard disks, OS-U and the like to the "P" machines. No, it is not cheap, but certainly a lot cheaper than it used to be. The hold up is that manufacturers are not convinced that "P" users will want to up-grade. Write us! We will pass the word where it will do the most good.

P.S. It's tax time. Check first, as PEEK may be tax deductible.

THE INSIDE STORY

A new series designed to bring the hardware beginner up to speed. The series will examine the overall system and all commonly used 6502 based OSI boards, for C4-P machines through time-sharing.

By: Brian Hartson
Tech. Editor.

Over the years, PEEK has had lots of articles and letters that are concerned with specific portions of OSI hardware. Those who have read these articles either already know the hardware down to the minute details or have followed the instructions carefully to get the desired results. At the completion of the project, knowledge was gained, but only in the specific area.

My aim in this series will be to try to give you the overall picture of what goes on in your box. We will look at the overall system, the individual boards and how they interrelate. Along the way, we may even suggest changes or areas for improvement to make your machine perform better and/or faster.

Because this series is to help you get more from your machine through better understanding, I will be watching for your comments, suggestions and questions about those areas that need special attention or things that are still not clear to you. Just write to me at PEEK(65).

During the course of this series, I will try to cover the OSI world, but we have to start somewhere. So, arbitrarily I have chosen the C-2 and C-3 systems. Even if this is not your area, read along. There is more similarity than dissimilarity with whatever you have.

published monthly

Editor - Eddie Gieske

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

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

All subscriptions are for 1 year and are payable in advance in US Dollars.

For back issues, subscriptions, change of address or other information, write to:

PEEK (65)
P.O. Box 347
Owings Mills, MD 21117 (301) 363-3268

To begin, I will assume that you have had the cover off and with the help of the highly recommended Sams Manual, you have figured out which board is the CPU, memory, etc. Seriously, if you don't have a copy of Sams, get one from PEEK or elsewhere.

So, now you know where things are, physically. All well and good, but what we will be concerned with first is where things are the way the machine sees things. In order for the system to work, it has to know where, in the machine's memory, it can find the various services it will need. It also needs to reserve chunks of memory to perform household chores. To keep things simple, let's lump them all together and call them "hardware devices". So that we can have an easy reference, we have the following "System Map" that tells us where, in memory, every hardware device is located; according to its hexadecimal address.

SYSTEM MAP

C-3 SYSTEM

Hex Address	device
C000-C0FF	470/505 FLOPPY DISK CONTROLLER
gap	
C200-C2FF	HARD DISK CONTROLLER
gap	
C400-C4FF	DIABLO PARALLEL PRINTER CONTROLLER
gap	
C700-C7FF	96 LINE PARALLEL INTERFACE
gap	
CC00-CCFF	LEVEL3 NETWORK CONTROLLER
CD00-CDFF	VOICE I/O CONTROLLER
CE00-CEFF	LEVEL 3 LOCAL CONTROLLER
CF00-CFFF	LEVEL 1 CONTROLLER
D000-DFFF	LEVEL 3 EXECUTIVE RAM
E000-EFFF	HARD DISK DUAL PORT RAM BUFFER
gap	
F200-F2FF	510 SCRATCHPAD RAM
gap	
F400-F4FF	CENTRONICS PARALLEL PRINTER CONTROLR
gap	
F700-F7FF	510 PIA: PROCESSOR SELECT LEVEL3 BANK SWITCH
gap	
FB00-FBFF	430 I/O CONTROLLER
FC00-FCFF	CONSOLE PORT
FD00 FDFF	HARD DISK BOOTROM SPACE
FE00-FEFF	65A MONITOR ROM SPACE
FF00-FFFF	FLOPPY DISK BOOTROM SPACE

C-2 SYSTEM
WHERE DIFFERENT FROM C-3 ABOVE

D000-DFFF	540 VIDE0 RAM
E000-E7FF	COLOR VIDEO RAM
gap	
FC00-FCFF	CONSOLE PORT
FD00-FDFF	POLLED KEYBOARD ROM SPACE
FE00-FEFF	65V MONITOR ROM SPACE
FF00-FFFF	BASIC ROM SUPPORT

It is easy to see that OSI was wasteful of memory space. Just look at all the wasted gaps. Wasted to most people, but this is where some programmers put their special bits of code. OSI could have put all the controllers in the F000-FCFF space and given us 4K more of user space. The block from C000 through CFFF is pretty much common to all OSI machines and addresses things like the disk controllers and boards such as the 550 and 555. D000 through FFFF, in C-2s and personal machines contains support for polled video systems and ROM BASIC. In the C-3 and larger machines, as they are serial systems and no need for video, this space is used to provide support for time-sharing, additional I/O and disk boot.

We now have a general picture of the OSI computer. Now for a little detail. Each board in the system has one or more functions so that, to make an OSI computer, many boards are needed. These boards are then connected together by a backplane or motherboard. The OSI backplane is a parallel structure that provides a roadway for all address, data and control signals to reach each board. There is no decoding or control done on the backplane. The following is the Pin definition of the OSI backplane.

OSI BACKPLANE PINOUT

PIN	DEFINITION	
01	WAIT	LOW TRUE
02	NMI	LOW TRUE
03	IRQ	LOW TRUE
04	DATA DIRECT.	HIGH TRUE
05	DATA 00	
06	DATA 1	
07	DATA 2	
08	DATA 3	
09	DATA 4	
10	DATA 5	
11	DATA 6	
12	DATA 7	
13	UNDEFINED	
14	UNDEFINED	
15	UNDEFINED	
16	UNDEFINED	
17	RESET	LOW TRUE
18	UNDEFINED	
19	ADD 19	
20	ADD 18	
21	ADD 16	
22	ADD 17	
23	+12 VOLTS	
24	-9 VOLTS	
25	+5 VOLTS	
26	+5 VOLTS	
27	GROUND	
28	GROUND	
29	ADD 06	
30	ADD 07	
31	ADD 05	
32	ADD 08	
33	ADD 09	
34	ADD 01	
35	ADD 02	
36	ADD 03	
37	ADD 04	
38	ADD 00	
39	PHASE 2	
40	R/W	
41	VMA	
42	VMA AND PHASE 2	
43	ADD 10	
44	ADD 11	

Continued

45 ADD 12
46 ADD 13
47 ADD 14
48 ADD 15

Let's define the above signals and explain what they do:

The WAIT signal is used by controllers, or memory, to switch the processor speed when they are addressed. When WAIT goes low it slows the processor clock to 500 KHtz.

NMI is the nonmaskable interrupt. Unlike IRQ, this interrupt cannot be ignored or delayed. When this signal goes low the processor finishes the current instruction, then loads its program counter with the address that is contained in memory locations FFFA and FFFB. It then transfers control to the program that starts at the 16 bit address that is contained in these memory addresses (FFFA and FFFB). These addresses are referred to as the NMI VECTOR.

IRQ is interrupt request, that is, a controller is requesting processor time. Unlike NMI the processor can ignore or mask this interrupt. This signal like NMI also has a VECTOR and it is the 16 bit address that is contained in locations FFFE and FFFF.

DATA DIRECT: This signal controls the direction of data flow into or out of the board. The signal is high for a Write operation and low for a Read operation.

RESET is an optional reset line not connected to the processor reset signal. This signal under normal OSI usage is not used.

PHASE 2 is the system clock, all data transfers take place during the phase 2 period. This signal controls the bi-directional data receiver/drivers inside the microprocessor during read/write times. This signal along with the read/write signal make the signal called Data Direction.

R/W is the Read/Write signal. When high, a Read operation will occur, when low, a Write operation will occur.

VMA or Valid Memory Address is a signal that only applies when using the 6800 microprocessor that is on the 510 board, otherwise this signal is pulled high.

VMA and PHASE 2: This signal is the same signal as Phase 2. The only difference is that it is used as an enable signal by the controller boards. When high a data transfer can take place.

DATA 00 to DATA 07 are the data lines. These signal lines on the backplane are bi-directional.

ADD 00 to ADD 15 are the 16 address lines that are normally used by the system.

ADD 16 to ADD 19 are the extended address lines used by the timeshare software to switch RAM banks. These signals are generated by a PIA on the 510 board under control of the LEVEL3 software.

There are currently five undefined lines on the backplane. In times gone by lines at pins 13 thru 16 were the Data Lines 8 thru 11, required for the 12 bit operation of the 6100 CPU that was to be used on the 560Z CPU expander board. This board used either or both a Z80 and a 6100 processor. The 6100 is a 12 bit microprocessor compatible with Digital Equipment's PDP-8. I do not know if this board ever made it out the OSI door. Line 18 has always been undefined to my knowledge.

Well, that gives you a quick trip down the backplane. If it didn't all sink in (I wouldn't be surprised - there is a lot in there) go back and read it again. If it is still not clear, hang in there. As we progress through the boards, things should clear up for you.

Next month we will attack the CPU boards.

★

6502 ASSEMBLY LANGUAGE PROGRAMMING CLASS

PART IX

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

I'm sure you've seen BASIC programs that perform seemingly magic and when you go to dope them out - ZAP! You suddenly run into a slug of meaningless DATA statements and the ubiquitous (Gad! I've been waiting months to be able to use that word!);

```
X=USR(X)
```

If you look up the USR function in a manual, you'll find only that it "executes a user-defined machine language program". Swell. Actually, OSI's "The C8P User's Manual" and "The C4P User's Manual" contain a good example of how to use the USR function to your advantage, even though they tried like the devil to under document it and write the code to be as confusing as possible. The idea in the example was to execute a machine code program and then tell BASIC something about what happened.

This is usually referred to as "passing parameters".

Before we go any further, I'd like to clear up a couple of things that haven't been made clear about the USR function under OS-65D in anything I have ever read. First of all, under no circumstances should you blithely enter "X=USR(X)" under OS-65D. OSI wrote a disk read/write utility into OS-65D and USR defaults to it with a read operation. But if location $22D4 was changed accidentally, you could be in for a rude awakening. Secondly, the documentation tells you to change locations 8955 and 8956 to point to your machine code program. What it doesn't tell you is that if you use those locations, OS-65D is in the DOS context. That's fine if all you want to do is a disk access, but if you aren't aware of it and you need BASIC to work, you'll go nuts trying to figure out what happened. BASIC stores the real vector to USR at locations 574 and 575 and you are much better off always using those locations to point to your machine code. If you need the disk, use the routine SWAP at $2CF7 like God and the programmers intended. Okay. Enough pontificating.

Before the advent of OS-65D V3.3, the most common use of the USR function was to do a screen clear. We did a screen clear in one of the earlier lessons. Again, the usual technique was to include the machine code in DATA statements and put the code in memory through a series of READs and POKEs. That done, the program would POKE in the address of where the code resided in memory into 8955/8956 or 574/575 and USR(X) - *BANG* - your screen was clear. Cassette system owners are especially lucky because they have just enough unused space on page 2 ($0200) to hold such a machine code program without having to worry about subsequent programs

overwriting it by accident. You just have to admire the folks at Microsoft though, for having the foresight to make "USR" a function rather than a command. My point is that the way they wrote it, USR can be used as either a command to simply execute machine code, or it can additionally be used to allow machine code programs to directly interact with the language. I bet you were always intrigued by the syntax of "X=USR(X)" as I was. After all, shouldn't X equal *something* after the screen was cleared? This leads us back to the idea of passing parameters.

Let's look at what really happens when BASIC encounters "X=USR(X)". As soon as BASIC sees the variable name "X", it automatically knows it's going to evaluate an equation and so it executes the code for the keyword "LET". LET identifies the type of variable that will be assigned the value of the equation as one of three types; (1) floating-point, (2) integer, or (3) string. Then, after dutifully checking to make sure you put in an equals sign, it jumps to the code that untangles the right-hand side of the equation. This code is a subroutine located at $ØCCD and is called the formula evaluator or "FRMEVL". In our example, FRMEVL sees the USR, and does a JSR to itself to evaluate the expression contained within the parenthesis, before jumping to where locations 574 and 575 tell it the code for USR resides. Actually, FRMEVL in turn calls a routine called EVAL to decipher each individual component in the equation between operators (ie. +, -, *, /, ^, AND, OR, and NOT). When the code pointed to by USR does an RTS back to FRMEVL, FRMEVL in turn does an RTS back to LET which stores the result in the variable we told it to. Keeping track of all of this is no mean feat. If you ever want to feel humble, take a look at a disassembly of BASIC.

Typically, the type of information we'll want to give to BASIC from a machine code program is going to be a number and is further typically a single byte value from Ø to 255. There is a routine that will let you give a signed 16-bit value from -32768 to 32767 to BASIC at $1218 called GIVAYF (which I interpret as GIVe A&Y to the Floating point accumulator). If you put the Most Significant Byte of your value in the 6502's Accumulator and the Least Significant Byte in the Y register and JMP to $1218, BASIC will get the value. I do not advise using the indirect jump vector at $ØØØ8. Some versions of OS-65D do not install the address for GIVAYF there properly. If you're into floating point math or need to pass a full 16-bit positive value to BASIC, I can't help you.... yet.

Okay, what could we want to give BASIC? In the program STRTRK.BAS that I uploaded to OSI SIG recently, I used the USR function to poll the keyboard so that if no keys were pressed, the program could continue on to do something else as opposed to using an INPUT statement which would wait until the user pressed the <RETURN> key before it could continue. The code I used there is for OS-65D V3.3 and one of the main reasons for that is that the V3.3 keyboard poll can be used independently of BASIC and 65D and it doesn't disturb page zero. The Assembly language program that interfaces to BASIC goes like this:

```
10 JSR $3590 ; DO KEYBOARD POLL
20 TAY       ; PUT KEYPRESS IN Y REGISTER
30 LDA #$00  ; INIZ ACCUMULATOR
40 JMP $1218 ; JUMP TO GIVAYF
```

Since this code is independent of its location in memory, it can be used on any size system. After POKEing 574 with the LSB and 575 with the MSB of the address of where the code is stored in memory, "X=USR(X)" will cause "X" to end up holding the result of the keyboard poll. If no keys were pressed, X will equal Ø and if a key was pressed, "X" will hold the ASCII value of that keypress. From there, you could use the CHR$ function to write your own word processor in BASIC.

As I alluded to in my reference to cassette based systems above, an important concern when adding machine code to BASIC program is where to put the code in memory. Another is how to protect that code from getting overwritten by BASIC. In earlier lessons, I provided a memory map of OS-65D V3.3. In that map, all of memory up to $3A79 is reserved for use by OS-65D and the resident language (in our case, BASIC). From $3A79 to the top of your system's contiguous memory is defined as the workspace. Data file buffers notwithstanding, the workspace begins by holding your program. The memory beginning with the end of your program to the top of the workspace is used to hold variables. BASIC stores non-subscripted variables first and then subscripted variables (arrays) in tables and maintains pointers to the starting and ending addresses of these tables. The wild card in this arrangement is string storage. The entries in these tables for string variables do not store the actual strings, but instead hold pointers to where the real strings are stored in memory and the length of the string. BASIC stores the strings beginning at the top of memory, building downward toward the array storage table. Thus it is essential that we restrict BASIC's use of memory in order to protect our machine code.

BASIC maintains the highest available memory address of your system in memory locations 132 and 133 in LSB/MSB format. Altering these locations to a value less than the address of where your machine code will reside will protect the code from being overwritten by BASIC with string storage if you choose to put the machine code at the top of your system's memory. When you choose to alter 132/133, you should do so at the very start of your program and immediately follow it with the CLEAR command. This will insure that BASIC knows its limits and won't lose anything in midstream. The advantage of putting machine code at the top of memory is that the code will remain untouched and available as long as you don't reset your system or re-invoke BASIC with the "BA" command to OS-65D.

The alternative to putting the code at the top of memory is to store the code at the beginning of the workspace, in front of your BASIC program. OS-65U users are well indoctrinated in this technique. Under OS-65D, the BASIC utility program "CHANGE" will alter the start of BASIC to a higher location. Running CHANGE is a bit scary until you decipher the meanings of the obscure prompts, but there are benefits to be reaped from the technique. Putting your machine code in front of your program allows you to store the machine code on disk in the same file as your program, thus making retrieval simple and also eliminating the need to add bulky DATA statements to your program (once the machine code is properly installed, of course). Be fore-

warned however, that you cannot type in a program and then later run CHANGE to add space in front of the program. You must first run CHANGE and then enter and store your BASIC program. With care, you could actually get around this with indirect files, but that can get cumbersome with larger programs. Do it by the book and save yourself trouble.

Back to strings, you'll remember that when I discussed FRMEVL, I said it does a JSR to our USR code. However, when our code RTS's back to FRMEVL, FRMEVL does a check to see that the variable being dealt with is a number and not a string. This is because Microsoft wrote the code to only allow the MID$, RIGHT$, and LEFT$ functions to deal with strings, which is perfectly reasonable considering the other tools in the language. But if you're dead set on using your own code to manipulate strings, there is a way around this problem. The solution is to pull the return address back to EVAL off the stack and return instead to FRMEVL before the string check is made.

The sample program I am including here will take a string from BASIC and reverse it. You'll note that in the assembly source code, I pull the string from BASIC and store it in my own buffer, INBUF. I did this to insure that the original string is not disturbed by anything your applications might need to do. The assembly source code is broken down into three sections. The first section is the set-up code which pulls the string from memory and saves the information about the string. The second section is the string manipulation code and can be replaced by your own application. The last section does the necessary housekeeping to tell BASIC where the resultant string is in memory and does the return to FRMEVL. The BASIC program is also a simple affair, but you'll notice that I moved the pointer at location 133 two pages in front of the machine code. This proved to be necessary in my tests, but I honestly cannot explain it.

Using DEBUG or the OSI Assembler, enter the assembly source code and store it in a file for later use. Next, assemble the code to memory. You might also want to save the code on disk. Next, you can simply exit the assembler, invoke BASIC and type in the

```
10 POKE 133,158: CLEAR: POKE 574,0: POKE575,160
20 INPUT A$
30 B$=USR(A$)
40 PRINT B$
```

```
10; BASIC STRING MANIPULATOR
20;
30; BASIC EXTERNALS
40;
50 ENDATB =$7E     END OF ARRAY TABLE
60 INDEX  =$6F     TEMP. POINTER TO STRINGS
70 FACEXP =$AE     F.P. ACCUM. EXPONENT
80 FACHI  =$AF     F.P. ACCUM. MSB
90 FACMHI =$B0     F.P. ACCUM. NMSB
100 CHKSTR =$0CBE  CHECK FOR STRING VARIABLE
110 FCERR  =$10D0  FUNCTION CALL ERROR
120 FREFAC =$1520  GET POINTER TO STRING
130;
140 INBUF  =$A100
150;
160        *=$A000
170;
180 PNT1   JSR CHKSTR      MAKE SURE IT'S A STRIN
190        JSR FREFAC      FIND STRING IN MEMORY
200        STX PNT2+1      SAVE STRING ADDRESS LS
210        STY PNT2+2      AND MSB
220        STA PNT3+1      AND LENGTH
230        TAY             CHECK LENGTH OF STRING
240        BEQ ERRJMP      ZERO?  ==> ERROR!
250        LDY #$00        INIZ POINTER
260 PNT2   LDA $FFFF,Y     FETCH CHARACTER OF STR
270        STA INBUF,Y     SAVE IT IN INBUF
280        INY             BUMP POINTER
290 PNT3   CPY #$FF        AT END OF STRING?
300        BNE PNT2        NO ==> PNT2
310;
320; INSERT MANIPULATION CODE HERE
330;
340        TYA             XFER Y REG. TO ACCUM.
350        TAX             NOW MOVE IT TO X REG.
360        LDA ENDATB      FETCH TOP OF FREE RAM
370        STA INDEX       GIVE IT TO BASIC
380        LDA ENDATB+1    FETCH TOP OF FREE RAM
390        STA INDEX+1     GIVE IT TO BASIC TOO
400        LDY #$00        INIZ PUT POINTER
410 PNT4   DEX             DECREMENT FETCH POINTE
420        LDA INBUF,X     FETCH A CHARACTER
430        STA (INDEX),Y   SAVE IT IN FREE RAM (B
440        INY             BUMP PUT POINTER
450        CPX #$00        FETCH PTR = 0?
460        BNE PNT4        NO! LOOP! ==> PNT4
470;
480        PLA
490        PLA             CANCEL RTS TO FRMEVL
500        LDA INDEX       FETCH PTR. TO NEW STRI
510        STA FACHI       GIVE IT TO BASIC
520        LDA INDEX+1     FETCH MSB
530        STA FACMHI      SEND IT TOO
540        LDY PNT3+1      LOAD Y REG. W/ STRING
550        STY FACEXP      AGAIN, GIVE IT TO BASI
560        JMP $159F       STORE STRING IN VARIABLE
570;                       AND QUIT
580 ERRJMP JMP FCERR       FUNCTION CALL ERROR!!
590;
600        .END
```

BASIC program above and run it. That program will present you with a mirror image of whatever you enter in response to the INPUT statement.

I would like to gratefully acknowledge the author of the Assembly Source Code for Microsoft OSI-BASIC written by M.K. Miller. Without that book, I would be totally lost. It used to be published by Aardvark, but I'm afraid it is no longer available and that is a shame.

★

MAPPING MACHINE LANGUAGE CODE

To thoroughly document your computer's BASIC or operating system (or any significant machine language program), you need to create a commented map of the routines. 'Resource' is a collection of BASIC pro-

grams which, working together, help you to produce annotated disassemblies.

Last month's PEEK published explanatory text and the first program. 'Resource' now concludes with the rest of the programs and some example results. The author used 'Resource' to aid in generating annotated cross reference lists for the OSI version of Microsoft's BASIC.

RESOURCE PART 2

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

The tables which accompany these final programs comprising "Resource" are selections from annotated cross reference lists for OSI-Microsoft 8K disk BASIC from OS65D V3.2 NMHZ disks. The tables were produced by using "Resource" and the annotations derive from both Jim Butterfield's memory maps (COMPUTE!II, June/July 1980) and my maps of OS65D (COMPUTE!, January-March 1981).

All addresses within the example tables are in hex and the first address on any line is the called address. Thereafter, the addresses refer to the place where the calling code resides. In addition, many of the addresses have preceding letters. These letters mean different things in different tables. In a JMP or JSR table, an M means the calling code is a JUMP instruction.

An S means the calling code is a JUMP TO SUBROUTINE instruction. In the MEMORY table, the letter is always the first letter of the calling opcode. For example,

```
1DF3 STA $0100,Y
```

is referenced in the table beside 0100 as SIDF3. The Zpage table has no leading letters. This table was produced by an early version of "Resource," before the extra information was added.

Editors note:

Resource follow up.

There is more to come!

The preceding article by Mr. Berger was brought to our attention by Mr. Dana Skipworth (Skip) who has been working with the programs and Mr. Tom Berger for quite some time. The result is that "Skip" has put together a series of comments and tips which will appear here in the coming months, along with further notes and utilities from Tom. Stay tuned!

Table 1. Keyword Action Addresses

Word	Token	Address						
END	80	082A	TO	9D		TAN	B9	1FF2
FOR	81	0748	FN	9E		ATN	BA	2056
NEXT	82	0C4D	SPC	9F		PEEK	BB	16UB
DATA	83	0BF9	THEN	A0		LEN	BC	15F6
INPUT	84	0B2C	NOT	A1	1E88	STR$	BD	12E9
DIM	85	0F24	STEP	A2		VAL	BE	1627
READ	86	0B58	+	A3	16D9	ASC	BF	1605
LET	87	09A6	-	A4	16C2	CHR$	C0	1566
GOTO	88	08A6	*	A5	18F4	LEFT$	C1	157A
RUN	89	087E	/	A6	1A0D	RIGHT$	C2	15A6
IF	8A	0929	^	A7	1E4F	MID$	C3	15B1
RESTORE	8B	080A	AND	A8	0E8C	NF	C4	ERROR
GOSUB	8C	0889	OR	A9	0E89	SN	C5	ERROR
RETURN	8D	08D3	>	AA		RG	C6	ERROR
REM	8E	093C	=	AB		OD	C7	ERROR
STOP	8F	0828	<	AC		FC	C8	ERROR
ON	90	094C	SGN	AD	1B34	OV	C9	ERROR
NULL	91	086D	INT	AE	1BC7	OM	CA	ERROR
WAIT	92	169C	ABS	AF	1B53	US	CB	ERROR
EXIT	93	223C	USR	B0	22F7	BS	CC	ERROR
DISK	94	2253	FRE	B1	1204	DD	CD	ERROR
DEF	95	1235	POS	B2	1225	/0	CE	ERROR
POKE	96	1693	SQR	B3	1E45	ID	CF	ERROR
PRINT	97	21AA	RND	B4	1F66	TM	D0	ERROR
CONT	98	0B53	LOG	B5	1B83	LS	D1	ERROR
LIST	99	06B9	EXP	B6	1EC1	ST	D2	ERROR
CLEAR	9A	067C	COS	B7	1FA2	CN	D3	ERROR
NEW	9B	0662	SIN	B8	1FA9	UF	D4	ERROR
TAB	9C							

Cont.

Table 2. Memory Table

```
.     Zpage
0001  L171A
0002  L1733
0003  L172C
0004  L1725
0016  S05FF L05F2 S0612
0017  L0512
0018  S0620
00A0  B0E8B
00A2  B0909
00FF  S1CF6 S1D67 S1D70 S1DB4 S1DBE L1DD1 S1E
      0F
.     Stack
0100  S1DF3 S1E14
0101  L03A6 L107B S1086 S1DEE
0102  L03B1 C03C2 L0FC9 L1077 S1081 S1E05
0103  L03B6 C03BB S1E01
0104  S1E0A
0109  L0C75 S0C8D
010F  L0C90
0110  L0C95
0111  L0C9F
0112  L0C9A
01DE  L0E79
01DF  L0E7E
.     Start of keyword address table
0200  L07F9
0201  L07F5
.     Start of operator hierarchy and address
      table
0266  C0D20 C0D48 L0D64
0267  L0D53
0268  L0D4F
.     Table of BASIC keywords (Start $0284)
0283  L061D
0284  S05E0 L0622 L0736 L073E
.     Error messages
0364  L0456
0365  L045C
.     BIT hiding code
07A9  B057C
08A2  B10CF
0EA2  B08E3
1410  B19BE
.     Constants
1E21  A1D91
1E22  A1D8A
1E23  A1D83
1E24  A1D7C
.     Operand pointing to IO flags
21D5  S2104
21DA  S210A
.     Stack pointer
226F  S211F
.     Table index for OS buffer write routine
228A  S217F
.     Buffer read write data for OS
22C8  L22E2
22C9  L22D6
22CA  L22DC
.     USR pointer to OS and disk
22F2  S22D9
22F3  S22DF
.     OS Input flag
2321  S20F5 L2101 S21D6 S2201 L2215
.     OS Output flag
2322  S20F8 L2107 S215D S21DB L21FE S2208
.     OS Passed char. (Control C)
2325  L0819 S0823
.     OS Disk sector number
265E  S22AC
.     BIT hiding code
28A9  B0E0F
.     OS Default IO flag
2AC6  L20F2
.     BIT hiding code
2CA9  B0E12
.     OS Read buffer pointer
2CE5  S2142
.     OS End of buffer on read
2CED  S2113
.     Transient GET and PUT pointer
2E7A  L22A6
.     OS Swapped value ($E1,$E2) Start pointer
      for buffer read
305A  S2116
305B  S2119
.     Pointer to SOURCE File header
3178  S2126 L2273
.     Number of tracks in SOURCE File
317D  S2136
.     BIT hiding code
3FA9  B0AEB
A4A2  B1AC4
```

Table 3. Zpage Table

```
. ;Index for Zpage, Jump vectors for BASIC
00  05AD 05D9 0609 0627 1357
01  135B 170D 173D 1832 1838 184B 184C 184E
    1850
. ;Search Character
0A  1320 1B0F 1E70 1EE2
0A  090C 0914 0916 0976 0998 0B95 0B9D 0E97
    0EB2 130D
.
. ;Scan between quotes FLAG
0B  130F 1324
0B  05B3 0607 060D 0910 0912 0918 091E 0BA2
    0E9D 0EA9
.
. ;POINTER: Input Buffer, # of subscripts
0C  0EAD 0EB0 0EB4 1028 1091 10DE 1108 112F
    1175 11AD
0C  049C 04F7 050F 05D1 05E9 061A 0E8E 0E95
    0E9B 0EA7
.
. ;Default DIM FLAG
0D  0F33 1059 109E 10D7 1113 116E
02  135F 1736 182E 1834 185A 185C 185E
03  172F 182A 1830 1866 1868 186A
04  1728 1826 182C 1872 1874 1876
.
. ;Type: FF=string 00=numeric
0E  09B5 0A4D 0B91 0CBF 0D10 0D36 0DB4 0E31
    0ED3 0F44
0E  0F63 105E 1097 1204 121A 1368 1601
. ;
0F  09B2 0BBC 0E36 0F46 0F71 105B 109A
.
. ;FLAG: DATA scan; LIST quote; memory
10  05AB 05B9 0600 1372 1396 139F
.
. ;Subscript FLAG; FNx FLAG
11  06AB 074A 0F6B 0F81 0F8C 1240 126A
.
. ;0=Input; $40=GET; $98=READ
12  0B0D 0B5E 0B80 0C12
.
```

Table 4. JMP and JSR Table

```
.   Jump vector for commands
0003  S047A
.
.   Jump vector for evaluation
006F  M0D84
.
.   Jump vector for functions
00A1  S0E03
.
.   CHRGET subroutine:  get BAISC character
00C0  S1615 S1C05 S1C12 S1C35 S2163 M220B
      S2259
00C0  S0484 S06D0 S079F S07E0 M07FD S0960
      S09A0 S0AC1 S0B8E
00C0  S0CB3 S0D00 S0DB6 M0E1B S0E4D S0F48
      S0F53 S0F7B S103D
.
.   Subentry; get previous character
00C6  S1652 S16A3 M2160 S21B0 S21E0
00C6  S04C S0C67 S0F2B S0F30 S0F37 S108A
      S12C5 S15B5 M1624
00C6  S06C7 S0798 S089D S092C S0941 S0A32
      S0B7B S0BC1 S0BDD
.
.   Search stack for FOR and GOSUB activity
03A1  S074F S08D9 S0C58
.
.   Open space in memory
03CF  S0504 S0FF1
03D6  S14A2
.
.   Test stack depth
0412  S075D S088B S0CDD
.
.   Check available memory
041F  S03CF S10EC S1142
.
.   Send error message then:
044C  M1194
044E  M1A87
044E  M085B M0BE6 M0CCA M0E20 M10D2 M1232
      M1352 M14D4 M1821
.
.   Warm start BASIC
```

```
10 REM ** RESOURCE 2 **
20 REM ** SOURCE AND EQUATE FILE BUILDER **
30 REM ** T.R.BERGER 11/80 **
40 REM ** REMOVE COMMA AND SEMICOLON **
50 POKE 2972,13:POKE 2976,13
60 PRINT:PRINT"RESOURCE ** STEP 2-BUILD SOURCE AND EQUATE FILES **"
70 PRINT:PRINT
80 INPUT"SCRATCH FILE";SF$
90 INPUT"OBJECT FILE";OF$
100 PRINT:INPUT"SYMBOL FILE";FS$
110 INPUT"EQUATE FILE";EF$
120 SP$="      "
```



```
130 REM ** COUNT SYMBOLS **
140 POKE 8998,00:POKE 8999,128
150 POKE 9000,00:POKE 9001,140
160 POKE 9006,00:POKE 9007,140
170 POKE 9008,00:POKE 9009,152
180 DISK OPEN,6,FS$
190 REM * SYMBOL COUNTER *
200 SN=-1
210 INPUT #6,IN$
220 IF IN$="XIT" THEN 250
230 SN=SN+1
240 GOTO 210
250 DISK CLOSE,6
260 REM ** LOAD SYMBOLS **
270 DISK OPEN,6,FS$
280 REM * DIMENSION STRING AND MARKER ARRAYS *
290 DIM SS$(SN),SS(SN)
300 FOR I=0 TO SN
310 INPUT #6,SS$(I)
320 NEXT I
330 DISK CLOSE,6
340 REM ** MAIN PROGRAM **
350 REM * LINE NUMBERS AND INCREMENT *
360 CL=10000: IN=10
370 DISK OPEN,6,SF$
380 DISK OPEN,7,OF$
390 REM * LOOP BACK HERE *
400 INPUT #6,IN$
410 IF IN$="XIT" THEN 670
420 REM * GET ADDRESS OF LINE *
430 A1$=LEFT$(IN$,4)
440 REM ** BINARY SEARCH FOR SYMBOL **
450 REM * SEARCH *
460 L=0:R=SN
470 M=INT((L+R)/2)
480 REM *EXIT HERE IF NOT FOUND *
490 IF L>R THEN OU$=SP$+MID$(IN$,5):GOTO 580
500 REM *EXIT HERE IF FOUND *
510 IF A1$=SS$(M) THEN 560
520 IF A1$>SS$(M) THEN L=M+1:GOTO 470
530 R=M-1:GOTO 470
540 REM * END OF SEARCH *
550 REM * CREATE SYMBOL AND MARK ADDRESS USED *
560 SS(M)=1:OU$="HH"+IN$
570 REM * CREATE RESOURCE LINE *
580 OU$=STR$(CL)+" "+OU$
590 REM * INCREMENT LINE NUMBER *
600 CL=CL+IN
610 REM * PRINT LINE *
620 PRINT#7,OU$
630 PRINT OU$
640 GOTO 400
650 REM * LOOP BACK FROM HERE *
660 REM * CLOSE FILES *
670 PRINT#7,IN$
680 PRINT #7,"E"
690 PRINT #7,"E"
700 DISK CLOSE,7
710 DISK CLOSE,6
720 REM *END OF MAIN PROGRAM *
730 REM * WRITE TWO BYTE EQUATES *
740 DISK OPEN,7,EF$
750 REM *FIRST LINE NUMBER *
760 CL=5000
770 REM * TITLE *
780 PRINT#7,STR$(CL)+" ;EQUATES"
790 CL=CL+IN
800 REM * COUNTER FOR EQUATES *
810 K=0
820 REM * PRINT EQUATES *
830 FOR I=0 TO SN
840 REM * SKIP SYMBOLS WHICH ARE LABELS *
850 IF SS(I)=1 THEN 930
860 A1$=STR$(CL)+" HH"+SS$(I)+" = $"+SS$(1)
870 PRINT#7,A1$
880 PRINT A1$
890 REM * NEXT LINE NUMBER *
900 CL=CL+IN
910 REM * INCREMENT EQUATES COUNT *
920 K=K+1
930 NEXT I
940 PRINT#7,"XIT"
950 PRINT#7,"E"
960 PRINT#7,"E"
970 DISK CLOSE,7
980 REM *FINISHED WITH EQUATES *
990 PRINT:PRINT
1000 PRINT"CODE SOURCE FILE REGENERATED":PRINT
1010 PRINTTAB(10) "RESOURCE FILE : "OF$
1020 PRINT TAB(10) "EQUATE FILE: ";EF$
1030 PRINT TAB(10) "SCRATCH FILE:"SF$
1040 PRINT TAB(10) "SYMBOL FILE: "FS$
1050 PRINT TAB(9) SN+1" SYMBOLS"
1060 PRINT TAB(9) K" EQUATES"
1070 PRINT:PRINT"PASS 2 COMPLETED"
1080 PRINT:PRINT:END
```

```
10 REM *** RESOURCE 3 - CROSS REFERENCE BUILDER ***
20 REM *** T.R.BERGER 11/80 ***
30 REM * DELETE COMMA AND SEMICOLON *
40 POKE 2972,13:POKE2976,13
50 PRINT:PRINT"** RESOURCE ** STEP 3-CROSS REFERENCE
   GENERATOR"
60 PRINT:PRINT
70 PRINT TAB(20)"TYPES OF REFERENCES"
80 POKE 8998,00:POKE 8999,128
```

```
90 POKE 9000,00:POKE 9001,140
100 POKE 9006,00:POKE 9007,140
110 POKE 9008,00:POKE 9009,152
120 PRINT:PRINT"B"TAB(10)"BRANCH"
130 PRINT"J"TAB(10)"JSR AND JMP"
140 PRINT"M"TAB(10)"MEMORY"
150 PRINT"Z"TAB(10)"Z PAGE"
160 PRINT:PRINT:PRINT
170 INPUT"YOUR CHOICE (J/B/M/Z)";CR$
180 IF CR$<> "B" AND CR$<>"J" AND CR$<>"M" AND CR$<>"Z" THEN 170
190 PRINT:INPUT"SCRATCH FILE";SF$
200 INPUT"REFERENCE FILE";RF$
210 PRINT:INPUT"NUMBER OF REFERENCES";NR
220 REM * DIMENSION ARRAYS *
230 DIM SS$(NR),SA$(NR),V(NR)
240 REM * SET SYMBOL NUMBER AND TYPE *
250 T=1:SN=-1
260 IF CR$="M" THEN T=2
270 IF CR$="Z" THEN T=3
280 REM * SYMBOL PLUCKER *
290 S=13:NL=4
300 IF CR$="Z" THEN S=15:NL=2
310 REM ** MAIN PROGRAM **
320 DISK OPEN,6,SF$
330 DISK OPEN,7,RF$
340 REM * LOOP BACK HERE *
350 INPUT #6,IN$
360 IF IN$="XIT" THEN 800
370 REM * TOO SHORT, NO SYMBOL *
380 IF LEN(IN$)<16 THEN 350
390 REM *CHECK FOR NO SYMBOL *
400 IF MID$(IN$,11,2)<>"HH" THEN 350
410 REM *DISPLAY LINE WITH SYMBOL *
420 PRINT IN$
430 REM * DETERMINE SYMBOL TYPE *
440 ON T GOSUB 970,1050,1140
450 REM *CHECK FOR RELEVANT SYMBOL *
460 IF FL=0 THEN 350
470 REM * GET ADDRESS OF LINE *
480 A1$=M$+LEFT$(IN$,4)
490 REM *GET SYMBOL *
500 A2$=MID$(IN$,S,NL)
510 REM * SEARCH SYMBOL TABLE *
520 REM * BINARY SEARCH *
530 L=0: R=SN
540 REM * SYMBOL NOT FOUND,INSERT IT *
550 IF L>R THEN 620
560 M=INT((L+R)/2)
570 REM * SYMBOL IN TABLE *
580 IF A2$=SS$(V(M)) THEN 700
590 IF A2$>SS$(V(M)) THEN L=M+1:GOTO 550
600 R=M-1:GOTO 550
610 REM * ADD A SYMBOL *
620 SN=SN+1:SS$(SN)=A2$
630 REM * POINT TO ITS PROPER POSITION IN ORDERING *
640 IF L=SN THEN 680
650 FOR I= SN-1 TO L STEP -1
660 V(I+1)=V(I)
670 NEXT I
680 V(L)=SN:M=L
690 REM * ADD A CROSS REFERENCE *
700 SA$(V(M))=SA$(V(M))+" "+A1$
710 REM * CHECK IF CROSS REFERENCE LINE IS TOO LONG *
720 IF LEN (SA$(V(M))) <50 THEN 350
730 REM * PRINT CROSS REFERENCE LINE *
740 PRINT #7,SS$(V(M))+" "+SA$(V(M))
750 PRINT SS$(V(M))+" "+SA$(V(M))
760 SA$(V(M))=""
770 GOTO 350
780 REM *LOOP BACK FROM HERE *
790 REM * CLOSE SCRATCH FILE *
800 DISK CLOSE,6
810 REM * PRINT REMAINING CROSS REFERENCE LINES *
820 FOR I=0 TO SN
830 IF SA$(V(I))="" THEN 860
840 PRINT #7, SS$(V(I))+" "+SA$(V(I))
850 PRINT SS$(V(1))+" "+SA$(V(I))
860 NEXT I
870 PRINT #7,"XIT"
880 DISK CLOSE ,7
890 REM * END OF MAIN PROGRAM *
900 PRINT:PRINT
910 PRINT TAB(10)CR$" REFERENCES COMPLETED "
920 PRINT TAB(10)"SYMBOLS FOUND: "SN+1
930 PRINT TAB(10)"REFERENCE FILE: "RF$
940 PRINT:PRINT:END
950 REM ** SUBROUTINES **
960 REM * BRANCH AND J (T=1) *
970 IF MID$(IN$,6,1)<>CR$ THEN FL=0: GOTO 1030
980 REM * SIFT OUT BIT INSTRUCTIONS *
990 IF MID$(IN$,6,3)="BIT" THEN FL=0:GOTO 1030
1000 REM *LABEL FOR TYPE *
1010 M$=MID$(IN$,7,1)
1020 FL=1
1030 RETURN
1040 REM * MEMORY (T=2) *
1050 M$=MID$(IN$,6,1)
1060 A2$=MID$(IN$,13,1)
1070 IF M$="J" OR A2$="Z" THEN FL=0:GOTO 1120
1080 REM * SIFT OUT BRANCHES *
1090 IF M$="B" AND MID$(IN$,6,3)<>"BIT" THEN FL=0:GOTO 1120
1100 REM * LABEL TYPE *
1110 FL=1
1120 RETURN
1130 REM * Z PAGE REFERENCES (T=3) *
1140 IF MID$(IN$,13,1)<>CR$ THEN FL=0:GOTO 1190
1150 M$=" "
1160 REM * LABEL FOR INDEXING *
1170 IF LEN(IN$)>16 THEN M$=RIGHT$(IN$,1)
1180 FL=1
1190 RETURN
```

```
10 REM *** RESOURCE 4  Z PAGE EQUATES ***
20 REM T.R.BENGER 11/80
30 PRINT:PRINT
40 PRINT"RESOURCE STEP 4 Z PAGE EQUATE FILE "
50 PRINT:PRINT:INPUT"Z PAGE CROSS REFERENCE FILE";ZF$
60 INPUT"Z PAGE EQUATE FILE";ZE$
70 POKE 8998,00:POKE 8999,128
80 POKE 9000,00:POKE 9001,140
90 POKE 9006,00:POKE 9007,140
100 POKE 9008,00:POKE 9009,152
110 PRINT:INPUT"NUMBER OF SYMBOLS";NS
120 REM * LINE NUMBER AND INCREMENT *
130 FL=1000:IN=10
140 REM * DIMENSION ARRAYS *
150 DIM SS$(NS),V(NS)
160 REM * SYMBOL COUNTER *
170 SN=-1
180 REM * LOAD SYMBOLS *
190 DISK OPEN,6,ZF$
200 PRINT:PRINT"LOADING SYMBOLS"
210 REM * LOOP BACK HERE *
220 INPUT #6,IN$
230 IF IN$="XIT" THEN 470
240 REM * JUST THE Z PAGE REFERENCES *
250 IN$=LEFT$(IN$,2)
260 REM * PUT SYMBOLS IN ORDER *
270 REM * SEARCH FOR SYMBOL *
280 REM * BINARY SEARCH *
290 L=0:R=SN
300 REM * GO ADD NEW SYMBOL *
310 IF L>R THEN 380
320 M=INT((L+R)/2)
330 REM * HAVE THIS ONE ,GET ANOTHER *
340 IF IN$=SS$(V(M)) THEN 220
350 IF IN$>SS$(V(M)) THEN L=M+1:GOTO 310
360 R=M-1:GOTO 310
370 REM * ADD SYMBOL TO LIST *
380 SN=SN+1:SS$(SN)=IN$
390 REM * POINT TO ITS PROPER POSITION IN ORDERING *
400 IF L=SN THEN 440
410 FOR I=SN-1 TO L STEP -1
420 V(I+1)=V(I)
430 NEXT I
440 V(L)=SN
450 GOTO 220
460 REM * LOOP BACK HERE *
470 DISK CLOSE,6
480 REM * SYMBOLS ALL LOADED *
490 REM * PRINT EQUATES *
500 DISK OPEN,6,ZE$
510 REM *TITLE *
520 PRINT #6,STR$(FL)+" ;Z PAGE EQUATES"
530 REM * PRINT EQUATES NOW *
540 FOR I=0 TO SN
550 FL=FL+IN
560 IN$=STR$(FL)+"  HHZZ"+SS$(V(I))+" = $"+SS$(V(I))
570 PRINT #6,IN$
580 PRINT IN$
590 NEXT I
600 PRINT #6,"XIT"
610 PRINT #6,"E"
620 PRINT #6,"E"
630 REM * BUFFER 6 REQUIRES A PUT *
640 DISK PUT
650 DISK CLOSE,6
660 PRINT:PRINT
670 REM * OUTPUT DATA *
680 PRINT TAB(9) SN+1" SYMBOLS"
690 NEXT I
700 V(L)=SN:M=L
710 REM * ADD A CROSS REFERENCE *
720 SA$(V(M))=SA$(V(M))+" "+A1$
730 REM * CHECK IF CROSS REFERENCE LINE IS TOO LONG *
740 IF LEN (SA$(V(M))) <50 THEN 410
750 REM * PRINT CROSS REFERENCE LINE *
760 PRINT #7,SS$(V(M))+" "+SA$(V(M))
770 PRINT SS$(V(M))+" "+SA$(V(M))
780 SA$(V(M))=""
790 GOTO 410
800 REM *LOOP BACK FROM HERE *
810 REM * CLOSE SCRATCH FILE *
820 DISK CLOSE,6
830 REM * PRINT REMAINING CROSS REFERENCE LINES *
840 FOR I=0 TO SN
850 IF SA$(V(I))="" THEN 880
860 PRINT #7, SS$(V(I))+" "+SA$(V(I))
870 PRINT SS$(V(1))+" "+SA$(V(I))
880 NEXT I
890 PRINT #7,"XIT"
900 DISK CLOSE ,7
910 REM * END OF MAIN PROGRAM *
920 PRINT:PRINT
930 PRINT TAB(10)CR$" REFERENCES COMPLETED "
940 PRINT TAB(10)"SYMBOLS FOUND: "SN+1
950 PRINT TAB(10)"REFERENCE FILE: "RF$
960 PRINT:PRINT:END
970 REM ** SUBROUTINES **
980 REM * BRANCH AND J (T=1) *
990 IF MID$(IN$,6,1)<>CR$ THEN FL=0: GOTO 1050
1000 REM * SIFT OUT BIT INSTRUCTIONS *
```

Cont.

```
1010 IF MID$(IN$,6,3)="BIT" THEN FL=0:GOTO 1050
1020 REM *LABEL FOR TYPE *
1030 M$=MID$(IN$,7,1)
1040 FL=1
1050 RETURN
1060 REM * MEMORY (T=2) *
1070 M$=MID$(IN$,6,1)
1080 A2$=MID$(IN$,13,1)
1090 IF M$="J" OR A2$="Z" THEN FL=0:GOTO 1140
1100 REM * SIFT OUT BRANCHES *
1110 IF M$="B" AND MID$(IN$,6,3)<>"BIT" THEN FL=0:GOTO 1140
1120 REM * LABEL TYPE *
1130 FL=1
1140 RETURN
1150 REM * Z PAGE REFERENCES (T=3) *
1160 IF MID$(IN$,13,1)<>CR$ THEN FL=0:GOTO 1210
1170 M$=" "
1180 REM * LABEL FOR INDEXING *
1190 IF LEN(IN$)>16 THEN M$=RIGHT$(IN$,1)
1200 FL=1
1210 RETURN
```

```
10 REM *** RESOURCE S ***
20 REM T.R.BERGER 2/81
30 PRINT TAB(10)"RESOURCE-SINGLE PASS"
40 REM ** REMOVE COMMA AND SEMICOLON **
50 POKE 2972,13:POKE 2976,13
60 POKE 8998,00:POKE 8999,128
70 POKE 9000,00:POKE 9001,140
80 POKE 9006,00:POKE 9007,140
90 POKE 9008,00:POKE 9009,152
100 INPUT"SOURCE FILE";SF$
110 INPUT"RESOURCE FILE";RF$
120 INPUT"EQUATE FILE";EF$
130 INPUT"CROSS REFERENCE FILE";CF$
140 INPUT"SCRATCH FILE";JF$
150 INPUT"NUMBER OF SYMBOLS";NS
160 INPUT"NUMBER OF Z PAGE SYMBOLS";NZ
170 REM **DIMENSION SYMBOL AND POINTER ARRAYS **
180 DIM SS$(NS),SB$(NS),SJ$(NS),SM$(NS),V(NS),S$(NS)
190 DIM ZS$(NZ),ZA$(NZ),U(NZ)
200 REM ** SYMBOL COUNTER **
210 SN=-1:ZN=-1:SP$="     "
220 REM ** FIRST PASS **
230 DISK OPEN,6,SF$
240 DISK OPEN,7,JF$
250 REM ** LOOP BACK HERE **
260 INPUT #6,IN$
270 IF IN$="XIT" THEN 1120
280 IF LEN(IN$)<15 THEN 260
290 REM ** ADJUST SOURCE,PICK UP SYMBOLS **
300 REM A1$=XXXX ADDRESS
310 REM A2$=OPCODE +
320 REM A3$=OPERAND (SYMBOL)
330 REM A4$=ADDRESS MODE
340 REM OU$=A1$+A2$+A3$+A4$
350 REM IN$=INPUT FROM OSI DISASSEMBLER
360 A3$="":A4$=""
370 REM ** GET ADDRESS **
380 A1$=LEFT$(IN$,4)
390 REM ** DO ERRORS **
400 IF MID$(IN$,13,1)="?" THEN A2$=" .BYTE $"+MID$(IN$,6,2):GOTO1070
410 REM ** DO REFORMATTING **
420 REM ** ELIMINATE END SPACES **
430 IN$=MID$(IN$,12):L=LEN(IN$)
440 IF MID$(IN$,L,1)=" " THEN L=L-1:GOTO440
450 IN$=LEFT$(IN$,L)
460 REM ** DO IMPLIED,ACCUMULATOR,IMMEDIATE ADDRESSING **
470 IF L<7 OR MID$(IN$,6,1)="#" THEN A2$=IN$:GOTO 1070
480 REM ** ADJUST OPERAND POSITION **
490 IF MID$(IN$,6,1)="$" THEN K=7:A2$=LEFT$(IN$,5)+" HH":GOTO 520
500 K=8:A2$=LEFT$(IN$,6)+"HH"
510 REM ** Z PAGE CHECK **
520 M=K+2
530 REM ** DO Z PAGE OPERANDS **
540 IF  M>L THEN A3$=RIGHT$(IN$,2):A2$=A2$+"ZZ":GOTO 690
550 IF MID$(IN$,M,1)>"/" THEN 580
560 A3$=MID$(IN$,K,2):A2$=A2$+"ZZ":A4$=MID$(IN$,M):GOTO 690
570 REM ** TWO BYTE OPERAND CHECK **
580 M=K+4
590 REM ** DO TWO BYTE OPERANDS **
600 IF M>L THEN A3$=RIGHT$(IN$,4):GOTO 630
610 A3$=MID$(IN$,K,4):A4$=MID$(IN$,M)
620 REM ** SEARCH FOR SYMBOL **
630 GOSUB 2310
640 REM ** SYMBOL NOT FOUND,INSERT IT **
650 IF L>R THEN 930
660 REM ** SYMBOL FOUND,ADD CROSS REFERENCE **
670 GOTO 1010
680 REM ** SEARCH FOR Z PAGE REFERENCE **
690 L=0:R=ZN
700 REM ** SYMBOL NOT FOUND,INSERT IT **
710 IF L>R THEN 790
720 qA77
730 M=INT((L+R)/2)
740 REM ** SYMBOL FOUND,ADD CROSS REFERENCE **
750 IF A3$=ZS$(U(N)) THEN 870
760 IF A3$>ZS$(U(M)) THEN L=M+1:GOTO 710
770 R=M-1:GOTO 710
780 REM ** ADD SYMBOL **
790 ZN=ZN+1:ZS$(ZN)=A3$
800 REM ** POINT TO PROPER POSITION IN ORDERING **
810 IF L=ZN THEN 850
820 FOR I=ZN-1 TO L STEP-1
830 U(I+1)=U(I)
840 NEXT I
850 U(L)=ZN:M=L
860 REM ** GET ADDRESSING MODE **
870 A5$=" "
880 IF A4$<> "" THEN A5$=RIGHT$(IN$,1)
890 REM ** ADD CROSS REFERENCE TO STRING **
900 ZA$(U(M))=ZA$(U(M))+" "+A5$+A1$
910 GOTO 1070
920 REM ** ADD SYMBOL **
930 SN=SN+1:SS$(SN)=A3$
940 REM ** POINT TO PROPER POSITION IN ORDERING **
950 IF L=SN THEN 990
960 FOR I=SN-1 TO L STEP -1
970 V(I+1)=V(I)
980 NEXT I
990 V(L)=SN:M=L
1000 REM ** FIND CORRECT CROSS REFERENCE TABLE **
1010 A5$=MID$(A2$,2,1):A0=1
1020 IF A5$="B" AND MID$(A2$,2,3)<>"BIT" THEN A0=2
1030 IF A5$="J" THEN A0=3
1040 REM ** ADD CROSS REFERENCE TO TABLE **
1050 ON A0 GOSUB 2250,2270,2290
1060 REM ** GENERATE LINE FOR SCRATCH FILE **
1070 OU$=A1$+A2$+A3$+A4$
1080 PRINT #7,OU$: PRINT OU$
1090 GOTO 260
1100 REM ** LOOP BACK HERE **
1110 REM ** CLOSE SOURCE AND SCRATCH FILES **
1120 PRINT #7,IN$
1130 DISK CLOSE,6
1140 DISK CLOSE,7
1150 REM ** END FIRST PASS **
1160 REM ** PASS 2, WRITE CROSS REFERENCE FILES **
1170 DISK OPEN,7,CF$
1180 PRINT #7,",  CROSS REFERENCES"
1190 PRINT #7,"."
1200 PRINT #7,".  Z PAGE"
1210 PRINT #7,"."
1220 REM ** DO Z PAGE REFERENCES **
1230 FOR I=0 TO ZN
1240 A0$=ZA$(U(I)):ZA$(U(I))="":A2$=ZS$(U(I))
1250 REM ** BREAK UP LONG LINES,PRINT FILE **
1260 GOSUB 2400
1270 NEXT I
1280 PRINT #7,".":PRINT #7,"."
1290 PRINT#7,".  JMP & JSR"
1300 PRINT #7,"."
1310 REM ** DO JMP & JSR REFERENCES **
1320 FOR I=0 TO SN
1330 A0$=SJ$(V(I)):SJ$(V(I))="":A2$=SS$(V(I))
1340 REM ** BREAK UP LONG LINES,PRINT FILE **
1350 GOSUB 2400
1360 PRINT #7,".":PRINT #7,"."
1370 PRINT #7,".  MEMORY":PRINT #7,"."
1380 REM ** DO MEMORY REFERENCES **
1390 FOR I=0 TO SN
1400 A0$=SM$(V(I)):SM$(V(I))="":A2$=SS$(V(I))
1410 REM ** BREAK UP LONG LINES,PRINT FILE **
1420 GOSUB 2400
1430 NEXT I
1440 PRINT #7,".":PRINT #7,"."
1450 PRINT #7,".  BRANCH":PRINT #7,"."
1460 REM ** DO BRANCH REFERENCES **
1470 FOR I=0 TO SN
1480 A0$=SB$(V(I)):SB$(V(I))="":A2$=SS$(V(I))
1490 REM ** BREAK UP LONG LINES,PRINT FILE **
1500 GOSUB 2400
1510 NEXT I
1520 PRINT #7,"XIT"
1530 DISK CLOSE,7
1540 REM ** END REFERENCE FILES **
1550 REM ** GENERATE RESOURCE FILE **
1560 DISK OPEN,6,JF$
1570 DISK OPEN,7,RF$
1580 REM ** LINE NUMBER AND INCREMENT **
1590 CL=10000:IN=10
1600 REM ** LOOP BACK HERE **
1610 INPUT #6,IN$
1620 IF IN$="XIT" THEN 1780
1630 REM ** GET ADDRESS LINE **
1640 A3$=LEFT$(IN$,4)
1650 REM ** SEARCH FOR SYMBOL **
1660 GOSUB 2310
1670 REM ** SYMBOL FOUND,MARK IT,ENTER LABEL **
1680 IF L<=R THEN SS(M)=1:OU$="HH"+IN$:GOTO 1720
1690 REM ** SYMBOL NOT FOUND,DELETE ADDRESS **
1700 OU$=SP$+MID$(IN$,5)
1710 REM ** ADD LINE NUMBER AND OUTPUT **
1720 OU$=STR$(CL)+" "+OU$
1730 CL=CL+IN
1740 PRINT #7,OU$:PRINT OU$
1750 GOTO 1610
1760 REM ** LOOP BACK FROM HERE **
1770 REM ** CLOSE SCRATCH AND RESOURCE FILES **
1780 PRINT #7,IN$
1790 DISK CLOSE,6
1800 DISK CLOSE,7
1810 REM ** RESOURCE FILE DONE **
1820 REM ** DO EQUATE FILES **
1830 DISK OPEN,7,EF$
1840 REM ** LINE NUMBER **
1850 CL=1000
1860 PRINT #7,STR$(CL)+"  ;EQUATE FILE"
1870 CL=CL+IN:PRINT #7,STR$(CL)+" ;"
1880 CL=CL+IN:PRINT #7,STR$(CL)+"  ;Z PAGE"
1890 CL=CL+IN:PRINT #7,STR$(CL)+" ;"
```

Continued

```
1900 REM ** DO Z PAGE EQUATES **
1910 FOR I=0 TO ZN
1920 CL=CL+IN
1930 PRINT #7,STR$(CL)"  HHZZ"ZS$(U(I))"=$"ZS$(U(I))
1940 PRINT STR$(CL)"  HHZZ"ZS$(U(I))" = $"ZS$(U(I))
1950 NEXT I
1960 CL=CL+IN
1970 PRINT #7,STR$(CL)+" ;"
1980 CL=CL+IN:PRINT #7,STR$(CL)+" ;"
1990 CL=CL+IN:PRINT #7,STR$(CL)+"  ;TWO BYTE"
2000 CL=CL+IN:PRINT #7,STR$(CL)+" ;"
2010 REM ** DO TWO BYTE EQUATES **
2020 FOR I=0 TO SN
2030 IF SS(I)=1 THEN 2070
2040 CL=CL+IN
2050 PRINT #7,STR$(CL)"  HH"SS$(V(I))"=$"SS$(V(I))
2060 PRINT STR$(CL)"  HH"SS$(V(I))" = $"SS$(V(I))
2070 NEXT I
2080 PRINT #7,"XIT"
2090 PRINT #7,"E":PRINT #7,"E"
2100 DISK CLOSE,7
2110 REM ** END OF EQUATES **
2120 REM ** FINAL DATA **
2130 PRINT:PRINT TAB(10)"RESOURCE COMPLETE"
2140 PRINT TAB(7)SN+1" SYMBOLS"
2150 PRINT TAB(7)ZN+1" Z PAGE LOCATIONS"
2160 PRINT TAB(8)"SOURCE FILE:      ";SF$
2170 PRINT TAB(8)"SCRATCH FILE      ";JF$
2180 PRINT TAB(8)"EQUATE FILE:      ";EF$
2190 PRINT TAB(8)"RESOURCE FILE:    ";RF$
2200 PRINT TAB(8)"CROSS REF. FILE   ";CF$
2210 PRINT:PRINT:END
2220 REM ** END OF PROGRAM **
2230 REM ** SUBROUTINES **
2240 REM ** MEMORY CROSS REFERENCES **
2250 SM$(V(M))=SM$(V(M))+" "+AS$+A1$:RETURN
2260 REM ** BRANCH CROSS REFERENCES **
2270 SB$(V(M))=SB$(V(M))+" "+MID$(A2$,3,1)+A1$:RETURN
2280 REM ** JMP & JSR CROSS REFERENCES **
2290 SJ$(V(M))=SJ$(V(M))+" "+MID$(A2$,3,1)+A1$:RETURN
2300 REM ** SEARCH FOR SYMBOL **
2310 L=0:R=SN
2320 REM ** SYMBOL NOT FOUND **
2330 IF L>R THEN RETURN
2340 M=INT((L+R)/2)
2350 REM ** SYMBOL FOUND **
2360 IF A3$=SS$(V(M)) THEN RETURN
2370 IF A3$>SS$(V(M)) THEN L=M+1:GOTO 2330
2380 R=M-1:GOTO 2330
2390 REM ** BREAK UP LONG LINES,PRINT CROSS REFERENCE FILE **
2400 L=LEN(A0$)
2410 IF L=0 THEN RETURN
2420 IF L<49 THEN A1$=A0$:A0$="":GOTO 2440
2430 A1$=LEFT$(A0$,48):A0$=MID$(A0$,49)
2440 PRINT #7,A2$" "A1$:PRINT A2$" "A1$
2450 GOTO 2400
```

★ ★ ★

BEGINNER'S CORNER

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

PROBLEM SOLVING

PAIN AND PLEASURE

Writing programs is enjoyable. The reason no doubt is because programming is very much about solving problems. The whole process is highly creative, demanding much mental effort. But if there is too much effort the task ceases to be pleasurable and becomes a chore. And that would never do!

When writing programs, reduction of effort is certain if a few simple techniques of problem solving are employed. Applying them to a problem will provide a much better description of it. The more detailed the description, the better the problem is understood.

Rather than begin coding immediately, it's a good idea to first go through a few preliminaries, and then make a plan. Once the plan is fully formed, it can be coded. The text that follows illustrates a way of doing this.

THE PROBLEM

The problem is: write a program that will produce a list of the names of the months. Examples of program output would be: a list of names beginning with January and ending with December, or a list beginning with April, going on through December, and ending with March. In fact, the program should be capable of printing a list with any number of month names in it, up to a maximum of 12.

Another example of output is:

October
November
December
January
February

Having identified and understood the problem, the question that follows is: "Is this a useful problem to solve?" Well, yes it is. The solution is useful in a budgeting program that produces reports based on financial data. Each monthly report looks back on the previous 12 months or a projection could be made to look forward to the year ahead. Examine the example (fictitious!) at the end of this text.

THE TOOLBOX

The next step is to assemble a TOOLBOX of information that will help solve the problem.

TOOLBOX

1. The 12 names of the month are .... and they will be reduced to three characters, e.g., Jan.

2. In BASIC, lists are best stored in arrays.

3. FOR...NEXT loops are a good way to printing lists.

4. Use INPUT to request the month numbers.

5. IF...THEN can be useful.

At this point it is tempting to begin coding at once, but there are a few more questions that could be asked.

MORE QUESTIONS

"What type of problem is it, and can it be solved?" Some problems cannot be solved on a computer. For example, - "Computer! Solve the Balance of Payments Problem!" Other problems would take too long to solve. For example, - produce a list of all possible 10 character names and print them. Some millions of years would pass before the task would be finished. The problem at hand deals with lists and it can be solved.

"What is the connection between the problem and the information in the TOOLBOX?" As far as one can tell none of the information is redundant, but useful information could be missing.

The month names are to be stored in a list (a one-dimension array). How can this be done? The names could be typed in like this: M$(1)="Jan",M$(2)="Feb"....how tedious. Why not let the computer do the work and read the names in from DATA statements? So add: "and held in DATA statements" to point 1 in the TOOLBOX. It is now time to make a formal plan.

THE PLAN

The plan need not be anything as formal as a diagram. The structure diagram shown here is merely one example of a plan. What is required is that the programmer be clear on the sequence of actions to be followed when writing the program.

The program would naturally begin by clearing the screen and printing a title, if any. Next, the number of months and the array would be declared: M=12 and DIM M$(M). Month names are placed in DATA statements. For reasons of space this box has been omitted from the diagram.

The rest of the program divides out into four blocks, as shown by the first row of the structure diagram. Subsequent rows of the diagram reveal how the problem can be broken up into smaller units. (Read the diagram from left to right, and down from any particular box in any particular row). It is evident that if this procedure is followed correctly then the plan will be complete. Coding is reduced to merely 'copying' the plan into BASIC.

THE SOLUTION

Even the simplest of programs can pose a challenge when it comes to the idea that will produce the required output. Producing a list from any month to December is easy. That problem can be solved using a simple FOR...NEXT loop.

But what if the list required is from October to February? The answer lies in a statement made earlier when the problem was identified - see paragraph four. The list would begin with 'April' and go through 'December' to 'March'. The list is in fact two lists. The first is from 'April' to 'December' and the second is from 'January' to 'April'. What is more significant is that the first list always ends with month twelve, and that the second list always begins with month one.

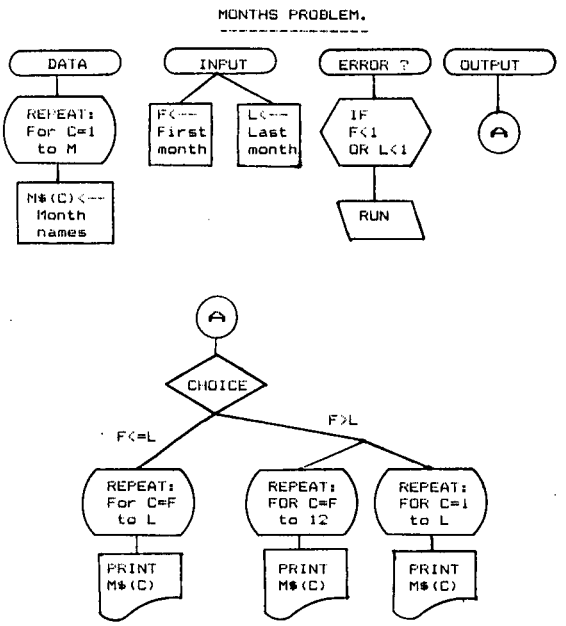
The simpler problem stated initially revealed that a FOR...NEXT loop could be used to produce a list. So use two FOR...NEXT loops to produce the list in the more complex example - see part two of the diagram, labeled 'A'.

The structure diagram, the plan of the solution, clearly states what the required code will be. Can you write the program? Use a FOR...NEXT loop when coding a REPEAT box and IF....THEN for the CHOICE box. Solution next month plus an improved algorithm for OUTPUT.

★

BUDGET PROGRAM. dated 01/01/85

	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	
01 Life In	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	600.00
02 House I	381.23												381.23
03 Car Ins						99.99							99.99
04 All Ris		233.11											233.11
05 Health	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	360.00
06 Other I		45.22											45.22
07 Gas	90.00			90.00		90.00			90.00				360.00
08 Doctor												12.50	12.50
09 Dentist										55.00			55.00
10 Car m'c				49.16	210.21	34.71	96.42				170.23	24.75	585.48
11 Loc Tax	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	1200.00
12 Phone	56.61		45.11		45.11		120.59		40.22		70.84		378.48
13 TV											60.00		60.00
14 Subs					85.00		41.50		57.93			51.90	236.33
15 Electri			122.33			23.93			95.00			78.34	319.60
16 Newspap	58.11							57.81					115.92
17 Vet										20.00			20.00
18 Alimony	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	6000.00
19 AA													0.00
20 OTHER			69.00									12.00	81.00
Exp. =$	1265.95	958.33	916.44	819.16	1020.32	928.63	938.51	737.81	963.15	755.00	981.07	859.49	11143.86
Income =$	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	2000.00	24000.00
Bal. =$	734.05	1041.67	1083.56	1180.84	979.68	1071.37	1061.49	1262.19	1036.85	1245.00	1018.93	1140.51	12856.14



COLOR PLUS REVISITED

By: Earl Morris
3200 Washington
Midland, MI 46840

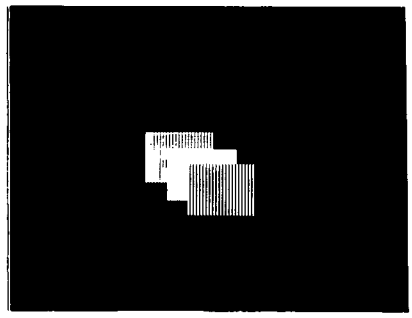
The December 1984 issue of PEEK(65) included a review of Generic's Color Plus board written by Bob Baldassano. I must agree with Bob that this board makes a very nice addition to an OSI system. I wanted to add a few technical details about this hi-res graphics board.

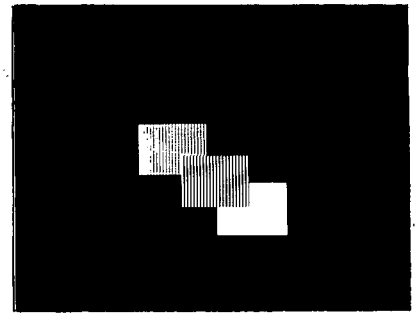
Previous versions of the software would add graphics commands to 65D 3.2 or 3.3. A new version is now available to add these same commands to OS-65U. Thus the Color-Plus board can be used with any of these operating systems.

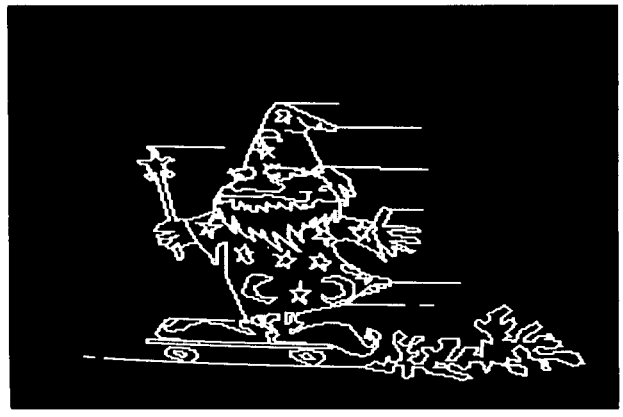
The Color Plus uses the Texas Instruments TMS9918A CRT controller chip. The board comes in two versions: One using 2118 five volt memory, and the other using 4116 memory which requires plus and minus 5 volts and plus 12 volts. The substantial price difference for the five volt chip is the cause of the $50 difference between the two versions of the finished board. Apple compatible joysticks are required. For non-Apple owners, this means joysticks with 150 K ohm pots. One of the several brands of Apple compatible bit pads should also work here.

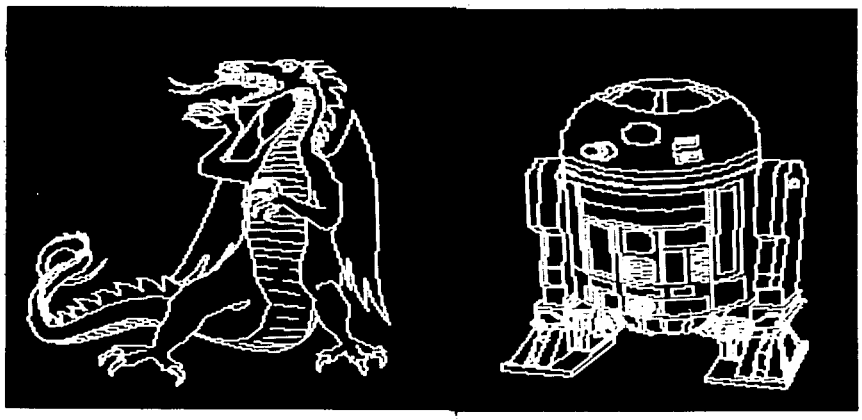
The object code for the graphics patches are included on the demo disk. For the real hackers, the source code for the entire graphics package is available on a second disk at extra cost. The source code is reasonably commented, but requires 32 pages to print out. The new BASIC graphics commands are well documented in the instruction manual. However, the 'MOVIES' program creates pictures by calling a data table from machine code. No instructions are included on how to create your own pictures using this program. If you are a Machine language programmer, this is easily figured out from the 'MOVASM' source code on the DEMO disk.

My biggest complaint about the Color Plus is that the address is set to $C900 and, short of cutting foils, cannot be altered. I happen to use $C800 to $D000 to hide an extra 4K of RAM. If you purchased the source code, the plot routines can be reassembled for a different board location by changing one line of code.









After all these words about this graphics board, I thought the readers of PEEK might want to see what it all looks like. Thus several photographs are shown here of the output from

the MOVIES demo program. The actual output is, of course, not in B&W but in color. The output can be displayed on a B&W monitor in shades of gray. But after seeing the colors, you had better be prepared to dig into your pocket and spend the additional $$$$ for a color monitor. It is difficult to show sprites on a still photograph, but the two photo sequence attempts to demonstrate the solid color block sliding between the two shaded blocks. Sprites don't have to be just blocks, but any shape describable in an 8 by 8 or 16 by 16 grid.

★

WAZZAT CORNER!

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

Accounting and budgeting programs provide comprehensive reports quickly and easily. But before that can happen, some unfortunate soul has to type in all those money amounts! It would be great if that decimal point did not have to be typed EVERY time and if the computer somehow knew that the number was "in" and did not wait for the "Carriage Return". And what does the program do with "q23.85", when it should have been "123.85"? The program listed here (for DOS 3.3) solves all these problems.

The program only accepts the numbers 0 to 9, backspace (for deletions), and the '-' sign if it occurs at the start of the number. As it stands, the program has been designed mainly to demonstrate that the idea works. It can be adapted to not only provide extended input for money amounts, but to also echo the numbers as they are typed in, and provide full editing of amounts via back and forward scrolling through all the entries that have been made.

If more than four digits are required before the decimal point, change "U$" in line 40. Merely add more "#"s - the program takes care of everything else.

Line 120 may be puzzling. A program such as this one needs speed if it is to be useful. So the Boolean Algebra statements in line 120 are used to replace three lines of BASIC IF...THEN statements. They would be as follows:

```
10 PRINT !(28): REM Extended Input for Numbers by LZJ
20 :
30 DIM N(99): M=45: Z=47: V=58: B=95: L$=CHR$(8): R$=CHR$(16)
40 R=9059: U$="####.##": L=LEN(U$)
50 L2$=L$+"#"+L$: FOR C=1 TO L: L1$=L1$+L$: NEXT
60 :
70 D$=" ": PRINT "Type number " U$ L1$;
80 FOR C=1 TO L-1: GOSUB 110: NEXT C: N(X)=VAL(D$)/100
90 PRINT : PRINT USING(U$) TAB( 30) N(X): X=X+1: GOTO 70
100 :
110 DISK !"GO 2336": Y=PEEK(R): Y$=CHR$(Y): IF Y=127 THEN Y=B
120 ON(((Y=M AND C=1) OR (Y>Z AND Y<V))+(Y=B AND C>1)*2)*-1GOTO 150,170
130 GOTO 110
140 :
150 D$=D$+Y$: PRINT Y$;: IF C=L-3 THEN PRINT R$;
160 RETURN
170 IF C=L-2 THEN PRINT L$;
180 D$=LEFT$(D$,LEN(D$)-1): IF C>1 THEN PRINT L2$;
190 C=C-2: IF C<1 THEN C=0
200 RETURN
```

```
120 IF Y=M AND C=1 THEN 150
125 IF Y>Z AND Y<V THEN 150
128 IF Y=B AND C>1 THEN 170
```

If any statement is evaluated as true, BASIC "thinks" -1. This value of -1 is then multiplied to provide the correct value for the "ON" branch - see end of line 120.

The statement in line 120 checks for the minus sign and whether this is the first character typed. If it is, then the branch to line 150 is taken. If the answer is "no" then the program falls through to line 130. Line 125 checks that the character typed is in the range 1 through 9 to 0. Line 128 looks for a backspace and not the first character. It is not possible to backspace off the first character!

Notice that "D$" is initialized to a blank. The way VAL works it does not matter if the leading character is a blank. If D$ is at least one character in length then a null check of D$ is not required before line 180. D$ always enters line 180 with a length of at least two.

Well that's it!

★

A CONVENIENT REGRESSION PROGRAM

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

For an Ohio Scientific, adequate statistical software is hard to find. Unfortunately, programs for least squares multiple regression, one of the most popular and useful statistical tools, are no exception.

Some generic programs are available. (See, for example, Lon Poole and Mary Borchers, Some Common BASIC Programs, 3rd ed. Berkeley, CA, Osborne/McGraw-Hill, 1979; and F. R. Ruckdeschel, BASIC Scientific Subroutines, Peterborough, NH, Byte/McGraw-Hill, 1981. Vol. 2.)

But these and most other programs have severe limitations. A few don't compute "t" or "F" statistics. Almost all don't calculate a Durbin-Watson statistic, necessary for time series analysis. Nor do they perform data transformations to eliminate serial correlation. Moreover, the programs place significant constraints on the number of variables and observations you can use. Also, data can't be read from files, so data available in other programs or files have to be re-keyed. Data transformation -- for example, taking logarithms or first difference -- may also require data to be re-entered.

By contrast, the program listed below, designed to run on an Ohio Scientific (C8-PDF, OSU), is relatively flexible and complete. It calculates:

R squared,

R squared adjusted for degrees of freedom,

the "F" value for the regression,

the standard error for the dependent variable,

the "t" values for the regression coefficients,

the Durbin-Watson statistic, and

rho hat (the estimated regression coefficient for successive regression residuals).

The program will also:

handle any number of observations up to the limit of your disk file,

take at least 20 explanatory variables on a 48K machine,

allow data transformations without re-keying the data,

allow special labeling of variables for output, and

perform Cochrane-Orcutt transformations of the regression to eliminate first order serial correlation.

The program presupposes data are read as string variables from an input file. Data transformations may be made when creating the file. The data need to be read, observation by observation: that is, observation 1 -- variable 1, variable 2, ..., variable k; observation 2 -- variable 1, variable 2, ..., variable k; and so on. No other information about the program is required. It is self-prompting.

With a hard disk, run times for 100 observations and 10 explanatory variables are about 6 minutes. Twenty explanatory variables and 200 observations take close to 29 minutes; 20 explanatory variables and 500 observations take about an hour.

```
10 REM*****************************REG*************************************
11 REM
12 REM                    MULTIPLE REGRESSION PROGRAM
14 REM                        RICHARD H. PUCKETT
16 REM                             12/28/84
17 REM
18 REM     THE PROGRAM WILL TAKE AT LEAST 20 EXPLANATORY VARIABLES,
20 REM WITH THE NUMBER OF OBSERVATIONS LIMITED ONLY BY THE CAPACITY
22 REM OF THE DISK FILE USED BY THE PROGRAM AS AN INPUT DEVICE.  THE
24 REM PROGRAM PRESUPPOSES DATA ARE READ FROM A DISK FILE OBSERVATION
26 REM BY OBSERVATION -- OBSERVATION 1, VARIABLE 1, ..., VARIABLE K;
28 REM OBSERVATION 2, VARIABLE 1, ..., VARIABLE K; AND SO ON.  THE
30 REM NAME OF THE FILE MAY BE ANY LEGAL FILE NAME.
32 REM     OUTPUT INCLUDES R SQUARED, R SQUARED ADJUSTED FOR DEGREES OF
34 REM FREEDOM, THE STANDARD ERROR OF ESTIMATE FOR THE REGRESSION, "T"
36 REM STATISTICS, THE "F" STATISTIC, THE DURBIN-WATSON STATISTIC,
38 REM AND RHO HAT FOR THE REGRESSION RESIDUALS.  THE PROGRAM WILL COM-
40 REM PUTE A COCHRANE-ORCUTT TRANSFORMATION TO ELIMINATE SERIAL CORRE-
42 REM LATION.
44 REM
100 REM                  MAIN
200 GOSUB 1000:REM  TO SET UP DATA FILE
210 GOSUB 2000:REM  TO SET UP REGRESSION
220 CR=1:REM  FLAG TO COMPUTE CROSS PRODUCTS
230 GOSUB 3000:REM  TO INPUT ROUTINE
240 GOSUB 6000:REM  TO COMPUTE COEFFICIENTS
250 CR=0:REM  FLAG TO ACCUMULATE DATA FOR TEST STATISTICS
260 GOSUB 3000:REM  TO INPUT ROUTINE
270 GOSUB 7000:REM  TO OUTPUT TEST STATISTICS
280 PRINT : INPUT "COCHRANE-ORCUTT TRANSFORMATION? (Y/N) ";AN$
290 IF AN$ < > "Y" AND AN$ < > "N" THEN 280: REM  TO TRY AGAIN
300 IF AN$="Y" THEN GOSUB 8000:REM  TO INITIALIZE FOR COCHRANE-ORCUTT
310 IF AN$="Y" THEN 220:REM  TO RESTART FOR COCHRANE-ORCUTT
320 PRINT : INPUT "ANOTHER REGRESSION? (Y/N) ";AN$
330 IF AN$ < > "Y" AND AN$ < > "N" THEN 320: REM  TO TRY AGAIN
340 IF AN$="Y" THEN GOSUB 8500:REM TO INITIALIZE FOR ANOTHER REGRESSION
350 IF AN$="Y" THEN 210: REM  TO SET UP REGRESSION
360 END
999 REM*************************************************************
1000 REM                 SET UP DATA FILE
1010 PRINT : INPUT "NAME OF DATA FILE";NF$
1020 PRINT : INPUT "# OF OBSERVATIONS IN DATA FILE";NO
1030 PRINT : INPUT "# OF VARIABLES IN FILE";NV
1040 ML = 4: REM  MAX # OF OBSERVATIONS STORED IN IMMEDIATE MEMORY
1050 DIM A(NV),CX(NV,2*NV),CY(NV)
1060 DIM LV(NV),SC(NV),VN$(NV),X(ML,NV),Y(ML)
1070 PRINT : PRINT "NAMES OF VARIABLES IN FILE?":PRINT
1080 FOR I=1 TO NV:INPUT VN$(I):NEXT
1100 RETURN
1999 REM*************************************************************
2000 REM                 SET UP REGRESSION
2010 PRINT : PRINT : PRINT "# OF FIRST OBSERVATION USED IN REGRESSION (E.G., 1 OR"
2020 PRINT "5.)  (IGNORE OBS USED IN LAGS OR 1ST DIFF.)"
2030 INPUT NF
2040 PRINT : INPUT "# OF LAST OBSERVATION USED IN REGRESSION";NL
2050 NL=NL-NF+1: REM  # OF OBS IN REGRESSION
2060 PRINT:PRINT"# OF REGRESSION COEFFICIENTS TO BE ESTIMATED, INCLUDING"
2065 INPUT "CONSTANT";K
2070 PRINT : INPUT "# OF DEPENDENT VARIABLE";LV(0)
2080 PRINT : PRINT "# OF EACH EXPLANATORY VARIABLE IN REGRESSION?" : PRINT
2090 FOR I=2 TO K : INPUT LV(I) : NEXT
2100 RETURN
2999 REM*************************************************************
3000 REM                 INPUT ROUTINE
3110 OPEN NF$,1
3200 IF NF=1 THEN 3400: REM  TO READ CURRENT DATA
3210 REM  ELSE READ BACK DATA
3220 FOR I=1 TO NF-1
3230 II=I-(NF-ML)
3240 REM  GET ALL DATA FOR OBSERVATION
3250 FOR J=1 TO NV: INPUT#1, A$: SC(J) = VAL(A$): NEXT
3260 IF II <= 0 THEN 3320: REM  TO END I LOOP
3270 REM  ELSE UNSCRAMBLE DATA IN REGRESSION
3280 TP=LV(0): REM  LOCATION OF Y
3290 Y(II) = SC(TP)
3300 X(II,1)=1: REM  DUMMY FOR CONSTANT
```

WP 6502 V1.2

By: John Whitehead
17 Frudal Crescent
Knoxfield 3180
Australia

John explains and fixes a number of shortcomings of WP 6502, cassette version, many of which are directly or indirectly applicable to disk versions.

I have a cassette based Superboard II with 24 x 24 and 48 x 12 screen, 32K of RAM (mainly 6116LP3) and 28K of EPROM on a Tasker Bus System.

I have an Australian 2K monitor (DABUG 3J) that contains the 48 x 12 screen driver, single key BASIC and correctly decoded keyboard. I modify all text type programs to work in 48 x 12.

I have 3 8K EPROMS containing BASIC utilities, WP6502 and Assembler. These are paged all at $8000 and run there. They are not down loaded, with the exception of small sections of self modifying code between $0222 and $02FF and use workspace from $0300 to the end of RAM.

Over the past year I have noted alterations I wanted to make to WP6502. As my EXMON disassembled listing of WP was a bit tatty, I decided to make an Assembler Source Code listing of it. This was performed by using a Symbolic Disassembler which converts M/CODE into an Assembler Source and puts it out on tape. This tape is then fed into the Assembler. The lines containing data are then tidied up and comments added. The Disassembler is written in BASIC and was converted to Symbolic by myself.

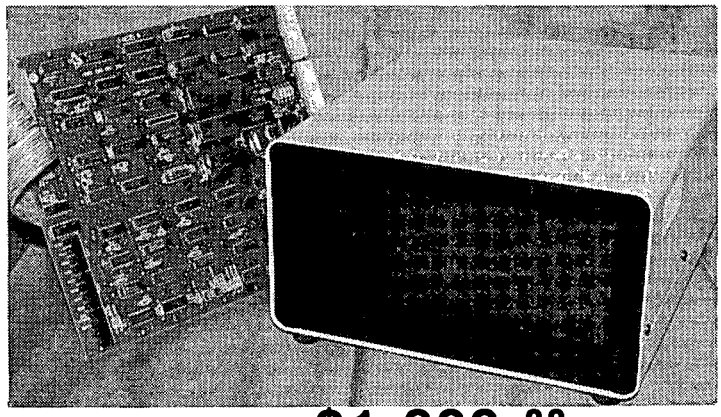
Now, after two months, I have a 32 page Source listing of my DABUG compatible 48 x 12 EPROM version of WP 6502 V1.2. It contains comments on the M/CODE functions that I have found, and mods that I have made. Sub-routines are listed where the calls come from, if there are less than six calls.

When I made my EPROM version of WP, I put the main core in the same place as it was in the tape version with just the individual Bytes changed where needed. Code that was at $0222 to $0FEF was relocated to $8222 to $8FEF which makes the listing compatible with both versions. Also, when I modified the code, I did not re-assemble it; just patched

SUPER HARD DISK Subsystem!

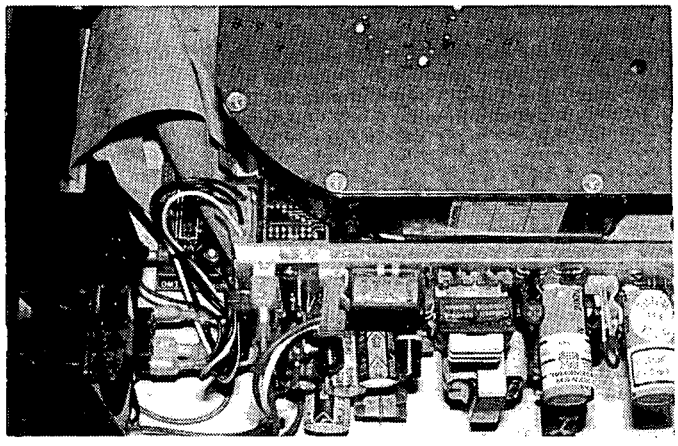
TURNS ANY FLOPPY BASED COMPUTER INTO HARD DISK BASED, INSTANTLY.

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



FROM **$1,999.00**

The SPACE-COM SUPER SUBSYSTEM Uses 5¼'' Industry Standard Hard Disk drives interfaced to the OSI bus by the DS-1 SCSI Host Adapter Board at the computer end and the state of the art OMTI 5000 series Intelligent Disk/Tape Controllers at the disk end. The Denver DS-1 Board not only provides the Bus Translation, but gives Real Time of Day, Day/Week, AM/PM, and Day/Mo. With on board battery, Date and Time are maintained w/o power.



The chassis is beautifully engineered with lighted on/off switch, standard a/c cord, and insulated spade terminals for easy service. A Corcom Emi Filter is incorporated in the a/c jack, and power is provided by an extremely efficient switching power supply. The case is also available in dual, side by side configuration and looks like an IBM PC box. It incorporates a larger power supply and can support 2 Winchester drives, or 1 drive and tape, or 2 5'' floppies in place of one of the above.

Drives can be accessed from any single or multi-user OSI system by running an overlay program on that partition, or can be booted directly by replacing current ROM/PROM with our SCI 500 PROM, available for $49.00 extra.

Single 20 M/B drive (15.7 formatted) single case	$1,999.00
Single 26 M/B drive (21 formatted) single case	$2,199.00
Dual 20 M/B drives (31.4 formatted) dual case	$2,999.00
Dual 26 M/B drives (42 formatted) dual case	$3,299.00
Super Fast 85 M/B drive (70 formatted) single case	$3,999.00
Dual 85 M/B drives (140 formatted) dual case	$6,699.00

SPACE-COM International

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

```
3310 FOR J= 2 TO K: TP=LV(J): X(II,J)=SC(TP): NEXT
3320 NEXT I
3400 REM      READ CURRENT DATA
3410 FOR I=NF TO NL
3420 REM    GET ALL DATA FOR OBSERVATION
3430 FOR J = 1 TO NV: INPUT#1, A$: SC(J) = VAL(A$): NEXT
3440 REM  UNSCRAMBLE DATA IN REGRESSION
3450 TP=LV(0): Y(ML) = SC(TP): X(ML,1) = 1: REM DUMMY FOR CONSTANT
3460 FOR J = 2 TO K: TP = LV(J): X(ML,J) = SC(TP): NEXT
3480 REM     COCHRANE-ORCUTT TRANSFORMATION
3500 IF RH = 0 THEN 3530: REM  SKIP COCHRANE-ORCUTT TRANSFORMATION
3510 Y(ML) = Y(ML) - RH * Y(ML - 1)
3520 FOR J = 1 TO K: X(ML,J) = X(ML,J) - RH * X(ML - 1,J): NEXT
3530 IF CR <> 0 THEN GOSUB 5000: REM  TO FORM CROSS-PRODUCTS
3540 IF CR =  0 THEN GOSUB 5500: REM  TO COLLECT DATA FOR TEST STATS
3600 REM     SHIFT DATA TO REFLECT NEW OBSERVATION
3610 FOR J = 1 TO ML - 1
3620 Y(J) = Y(J + 1)
3630 FOR II = 1 TO K
3640 X(J,II) = X(J + 1,II)
3650 NEXT II,J
3700 NEXT I: REM END MAIN LOOP
3710 CLOSE
3800 RETURN
4999 REM*****************************************************************
5000 REM                FORM CROSS-PRODUCT MATRICES
5020 FOR II = 1 TO K
5030 CY(II) = CY(II) + X(ML,II) * Y(ML)
5040 FOR J= II TO K: REM  USE SYMMETRY PROPERTY OF MATRIX
5050 CX(II,J) = CX(II,J) + X(ML,II) * X(ML,J)
5060 NEXT J,II
5100 RETURN
5499 REM*****************************************************************
5500 REM                DATA FOR TEST STATISTICS
5510 YH=0: REM  Y HAT
5520 FOR J=1 TO K
5530 YH = YH + A(J) * X(ML,J): NEXT
5540 MY = Y(ML) + MY: REM  SUM Y'S
5550 SH = YH * YH + SH: REM  SUM Y HAT'S SQUARED
5560 SY = Y(ML) * Y(ML) + SY: REM  SUM Y SQUARED
5570 E  = Y(ML) - YH: REM  RESIDUAL
5580 DD = E * E + DD: REM  DENOMINATOR OF DURBIN-WATSON
5590 IF I = NF THEN EF = E: REM  SAVE FIRST RESIDUAL
5600 IF I = NF THEN 5660: REM  TO SAVE CURRENT RESIDUAL
5610 REM  ELSE I > NF & EL <> 0
5620 ND = (E - EL) * (E - EL) + ND: REM  NUMERATOR OF DURBIN-WATSON
5630 NR = E * EL + NR: REM  NUMERATOR OF RHO HAT
5640 DR = EL * EL + DR: REM  DENOMINATOR OF RHO HAT
5650 IF I = NL THEN EN = E: REM  SAVE NTH RESIDUAL
5660 EL = E: REM  SAVE CURRENT RESIDUAL
5700 RETURN
5999 REM*****************************************************************
6000 REM                COMPUTE COEFFICIENTS
6010 REM  MAKE CX() SYMMETRIC
6020 FOR I=1 TO K-1: FOR J=I + 1 TO K
6030 CX(J,I) = CX(I,J)
6040 NEXT J,I
6050 N1 = K: REM  PARAMETER FOR MATRIX INVERSION
6060 GOSUB 6200: REM  TO MATRIX INVERSION
6070 REM           VECTOR OF COEFFICIENTS
6080 FOR I=1 TO K: FOR J=1 TO K
6090 A(I) = A(I) + CX(I,K + J) * CY(J)
6100 NEXT J,I
6110 RETURN
6199 REM*****************************************************************
6200 REM                MATRIX INVERSION
6210 REM  MATRIX TO BE INVERTED IS IN LEFT PART OF CX(N1, 2*N1) & IS
6220 REM  DESTROYED. INVERSE IS IN RIGHT PART OF CX. N1 NEEDS TO BE
6230 REM  DEFINED, & CX(N1,2 * N1) NEEDS TO BE DEFINED.
6240 REM  CALLING ROUTINE
6250 FOR R1= 1 TO N1: REM  INITIALIZE CX
6260 FOR C1= (N1 + 1) TO (2 * N1)
6270 CX(R1,C1) = 0
6280 IF (C1 - N1) = R1 THEN CX(R1,C1) = 1
6290 NEXT C1,R1
6300 FOR R1=1 TO N1: REM  ITERATE ON ROWS
6400 FOR K1= R1 TO N1
6410 IF CX(K1,R1) <> 0 THEN 6500: REM  TO NEXT PROCEDURE
6420 NEXT K1
6440 PRINT "SINGULAR MATRIX": STOP
6500 REM     CHECK IF NON-ZERO ELEMENT IS IN ROW R1
6510 IF K1 = R1 THEN 6600: REM  TO CREATE A UNIT VECTOR
6520 REM      ELSE SWAP ROWS K1 & R1
6530 FOR L1 = R1 TO (2 * N1)
6540 SW = CX(K1,L1)
6550 CX(K1,L1) = CX(R1,L1)
6560 CX(R1,L1) = SW
6570 NEXT L1
6600 REM       CREATE A UNIT COLUMN VECTOR IN COLUMN R1
6610 T1 = CX(R1,R1)
6620 FOR C1 = R1 TO (2 * N1)
6630 CX(R1,C1) = CX(R1,C1) / T1
6640 NEXT C1
6650 FOR L1 = 1 TO N1
6660 IF L1 = R1 THEN 6710: REM  TO END LOOP
6670 T1 = CX(L1,R1)
6680 FOR C1 = R1 TO (2 * N1)
6690 CX(L1,C1) = CX(L1,C1) - T1 * CX(R1,C1)
6700 NEXT C1
6710 NEXT L1
6720 NEXT R1: REM  END OF MAIN LOOP
6800 RETURN
6999 REM*****************************************************************
7000 REM                OUTPUT TEST STATISTICS
7020 DEF FN R(Z)=INT(Z*100000+.5)/100000:REM ROUND TO NEAREST 100000TH
7100 TP = LV(0): REM  DEPENDENT VARIABLE LOCATION
7105 REM     HEADER
```

it in. This way I do not require another printout of the whole listing, but the code is not so tidy! In the following mods, where it refers to $833C for example, use $033C.

My Source code can be fed into the Assembler and assembled if there is 32K of free RAM. An assembled listing can be fed into WP 6502 for printing out a bit at a time if there is not enough memory for Assembly.

If you would like a copy of my Source code ready to feed into the Assembler, or an Assembled listing for feeding into WP (state which one), send me one blank C90 tape, money for return postage, plus $2.00, and proof that you already have WP 6502 V1.2 (e.g., WP recorded in checksum on the tape you send). This listing could also be helpful to disk users of V1.2.

The following are the latest changes I have made to WP with the aid of the above Source listing. The changes can be patched in as required. I have put mine in front and behind the main core. With the tape version you will need to go after the existing code and the "start of text" pointer at $0241-2 which should be changed to the end of the added code. Although, most mods are small, it took a long time to find out how to do it.

1) When I first modified WP for DABUG and 48 x 12, I had to change some of the special characters to make it work. The linefeed marker was CHR $7F (DABUG screen clear CHR.) at $8228 and I had to change it to CHR $5B. DABUG 3 did not allow CHR $18 to CHR $1F to be used. Using $5B sometimes made G/EDIT difficult to read. With my modified DABUG 3J, I can use more characters and have changed the linefeed CHR to $1E. This also required 'lowest CHR' at $8225 to be changed to $18. If you have recorded text that has a different linefeed character, it is possible to do a G/EDIT and change them all as:

Press Break and change $0025 to the linefeed character used in the text, e.g., $5B. Run WP at $8F0E. (Normal warm start at $0000 Jumps to $8F0B to reset the variables. By entering three bytes later, resetting is bypassed). Do a G/EDIT (without pressing "return to menu") to any unused character, e.g., ***. Then press Return and do a

```
7110 PRINT : PRINT : PRINT "DEPENDENT VARIABLE - "VN$(TP): PRINT
7112 PRINT#5,: PRINT#5,
7114 PRINT#5, "DEPENDENT VARIABLE - "VN$(TP): PRINT#5,
7200 REM     COEFFICIENTS & T'S
7204 PRINT "COEFFICIENT"
7208 PRINT#5, "COEFFICIENT"
7210 PRINT TAB(4)"(T)":PRINT
7214 PRINT#5, TAB(4)"(T)":PRINT#5,
7220 N = NL: REM # OBS IN REGRESSION
7230 SE = SQR(DD / (N-K)): REM  STANDARD ERROR OF EST
7240 FOR I =1 TO K
7242 PRINT: PRINT#5,
7244 IF I=1 THEN PRINT "CONSTANT":PRINT TAB(1) FN R(A(1))
7248 IF I=1 THEN PRINT#5, "CONSTANT":PRINT#5, TAB(1) FN R(A(1))
7250 TP = LV(I)
7260 IF I > 1 THEN PRINT VN$(TP): PRINT TAB(1) FN R(A(I))
7264 IF I > 1 THEN PRINT#5, VN$(TP): PRINT#5, TAB(1) FN R(A(I))
7270 TP = SE * SQR(CX(I,K+I))
7280 PRINT "(";FN R(A(I)/TP);")"
7284 PRINT#5, "(";FN R(A(I)/TP);")"
7290 NEXT
7300 REM     OTHER STATS
7310 MY = MY / N: REM  MEAN OF Y
7320 R2 = SH / N - MY * MY: REM  VARIANCE OF Y HAT
7337 R2 = R2 / (SY / N - MY * MY): REM  R SQUARED
7340 PRINT : PRINT "R SQUARED  "; FN R(R2)
7344 PRINT#5, : PRINT#5, "R SQUARED  "; FN R(R2)
7350 F = R2 * (N-K) / ((1-R2)*(K-1))
7360 PRINT "F WITH (";(K-1);",";(N-K);") DF  "; FN R(F)
7364 PRINT#5, "F WITH (";(K-1);",";(N-K);") DF  "; FN R(F)
7370 R2= R2-(K-1)*(1-R2)/(N-K)
7380 PRINT "R BAR SQUARED  "; FN R(R2)
7384 PRINT#5, "R BAR SQUARED  "; FN R(R2)
7390 PRINT "STANDARD ERROR OF ESTIMATE  "; FN R(SE)
7394 PRINT#5, "STANDARD ERROR OF ESTIMATE  "; FN R(SE)
7400 DW = ND / DD
7410 PRINT "DURBIN-WATSON STATISTIC  "; FN R(DW)
7414 PRINT#5, "DURBIN-WATSON STATISTIC  "; FN R(DW)
7420 TP = (N-1)*(N-1)
7430 RH = NR/(N-1) - EF*EL/TP
7440 RH = RH/(DR/(N-1)-EL*EL/TP)
7450 PRINT "RHO HAT  "; FN R(RH)
7454 PRINT#5, "RHO HAT  "; FN R(RH)
7460 RETURN
7999 REM************************************************************
8000 REM               RE-INITIALIZE FOR COCHRANE-ORCUTT TRANS
8010 CO = 1: REM  SET COCHRANE-ORCUTT FLAG
8020 NF = NF +1: REM  RESET 1ST OBS
8030 N = N - 1: REM  ADJUST TOTAL OBS
8040 GOSUB 8600: REM  TO ZERO OUT VARIABLES
8050 RETURN
8499 REM************************************************************
8500 REM               SET UP ANOTHER REGRESSION
8510 CO = 0: REM  SET COCHRANE-ORCUTT FLAG
8520 RH = 0: REM  RESET RHO HAT
8530 GOSUB 8600: REM  TO ZERO OUT VARIABLES
8540 RETURN
8599 REM************************************************************
8600 REM               ZERO OUT VARIABLES
8610 REM  CROSS-PRODUCT MATRICES & COEFFICIENT VECTOR
8620 FOR I=1 TO NV
8630 CY(I) = 0: A(I) = 0
8640 FOR J= I TO NV
8650 CX(I,J) = 0
8660 NEXT J,I
8670 REM  PARAMETERS
8680 MY = 0: SH = 0
8690 SY = 0: DD = 0
8700 ND = 0: NR = 0
8710 DR = 0
8720 RETURN
```


second G/EDIT from *** to your linefeed character.

2) When using CTRL keys (I, M, X & B) with the shiftlock up, two characters appear in place of one. This is fixed by inserting STA $41 after STA $0217 at $8558. To do this, replace the STA $0217 with JSR CTRLFIX and at CTRLFIX put STA $0217, STA $41 and RTS.

3) When entering the L/EDIT mode, it allows "FROM" to be used to start editing anywhere in the text. I have altered the "VIEW" and "PRINT" modes to also use "FROM". I have also added a "SIMULATE and HOLD" mode that bypasses turning the printer on, to show where a page ends. (I think this is already in the disk version). Change the existing code between $87E9 and $87FA to BEQ $87F4, CMP#'S (SIMULATE and HOLD), BEQ $87F7, LDA #$60, STA $0247, JSR teletype on (INC $0205 for normal printer), JMP VIEWP, NOP. Add new code as: VIEWP LDA $3A, STA $38, LDA $30, STA $56, VIEWF LDA $0247, PHA, LDA #$4C, STA $0247, JSR $82E1, JSR $8784, JSR $84E9, JSR $8335, JSR $82F4, PLA, STA $0247, JMP $87FB. Also, change the JUMP at $87A1 to JMP VIEWF.

I use a teletype model 35 as a printer. This uses paper on a roll without perforations. So, I have added code to print perforations for me consisting of a line of dashes at the beginning of the first page and at the end of every A4 page. Details of this can be gotten from my listing on the tape.

5) It may be necessary to delete a large amount of text from an existing file to use for another purpose. I have added a Block Delete that works the same as 'DELETE SENTENCE'. You enclose the text to be deleted with a CTRL B and a CTRL X. Use the BLOCK VIEW to check the text to be deleted, then go to L/EDIT and put the cursor under the CTRL B character and press DB.

The new code for this is: BLOKX CMP #'B, BEQ DELB, LDA $47, JMP $8C06, DELB LDX $26, JMP $8C99. Change at $8C8F to NOP, NOP, JMP BLOKX.

(6) When the last word in a line contains a decimal point, the word can end up being split in two. When it's time for WP to do a linefeed, it looks backwards until it finds either a hash "#", a fullstop ".", linefeed marker or a

space. If one is found, it automatically starts a new line. This code is at $884D.

The detecting for a full stop is not needed as it will be followed by a linefeed marker or a space if a new line is needed. So at $8856, change CMP #'. and BNE $8862 to four NOPS. The same thing happens if an embedded character is in the middle of the last word, e.g., AB#C67DE. To fix, it needs extra code to look at the character that comes after the "#". If it's "C", don't split the word.

Change code at $8852 to: CMP $2B, BNE $885A, JMP ENDLX, .BYT $FF add new code somewhere: ENDLX INY, LDA ($14),Y, DEY, and #%01011111, CMP #'C, BEQ ENDLX1, JMP $8862, ENDLX1 JMP $884D

The "AND" instruction above allows both upper and lower case C to be detected. Notice in a line above that "AND #%01011111" has been cut in half. This can now be fixed by replacing the space with #C32.

(7) My last mod was to alter the "ZAP" so that the whole word "ZAP" had to be entered for ZAP to work. As stated at the beginning, a warm start is at $8F0B which sets up variables and prints the menu, then waits for a key press at $8F3F. Below this are all the compares required for the mod. New code needed is: GZAP JSR $83FF, CMP #'A, BNE $8F9D (this branch has to point to JMP $8465; it may not be at 8F9D), JSR $83FF, CMP #'P, BNE $8F9D (as above), JSR $833C, JSR $8345, JSR $8332, JMP $0000. Existing code to change is at $8F45 as: CPX #'Z, BEQ GZAP, JSR $82E1, NOP, NOP, NOP. And at $8F63 as: CPX #'V, BNE $8F73, JMP $8795.

(8) There is another fault with WP that I have not been able to sort out yet and that is to do with workspace full. This is what I have found so far: "ZAP" puts an @ at the start of workspace. "TYPE" checks memory and fills it up with $FB from the first @ to the end of RAM. If the top of RAM is $1FFF, workspace top is set to $1EFF and stored in $10 and $5B. The last 256 bytes are used for line and global editing. When text is entered and the characters get to $1EFE, "TYPE" shows 0 bytes free and $1EFF contains @. When one more character is entered it shows 65535 bytes free. More text can be entered until it reaches $1FFE. OM?> shows and this gives 65280 bytes free. Once the "bytes free" has passed "0 bytes free", line and global editing will not work correctly and may delete all of your text.

For those of you that use WP6502 V1.2 and are not too familiar with M/CODE, have a go at one of the above Mods. As long as you keep your original tape, no harm can be done even if you make a mistake. You will need a mnemonic to hex conversion chart and the extended monitor to check the modified code.

The M/CODE above contains labels. These are swapped for the address where you put the new code, e.g., GZAP in (6) above could be $1003.

It is not possible to have EXMON at the top of RAM and protect it from WP as can be done with BASIC, as WP fills all unused RAM with $FB nearly every time return is pressed. EXMON can be in write protected RAM, EPROM or in a section of RAM that is not continuous from WPS workspace. It can be below WPS workspace and the start of text pointer at $8241-2 set to the end of EXMON.

If you are an expert at M/CODE, you can have a go at (7) above.

★

"MAGIC SQUARS"

By: R. R. Groome
824 W. Main Street
Richmond, IN 47374

Remember the Aardvark rag? The Dec '81 issue had on page 13 a program called "MAGIC SQUARS" which turned out to be a nice graphics ditty. MICRO-COMPUTING/KILOBAUD in the Feb '81 issue had "MAGIC SQUARS" by Dr. Marc Lewis.....but it would not run on C2-4P's.

Here is my revison of that program that does run on OSI. For 1P's drop lines 80 & 1470.

In the listing, the CHR$(29) CHR$(31) type lines are printer commands.

If anyone wants a cassette copy, send me a cassette with a couple programs (anything!), and I will return both on the other side of cassette (C-60).

My system started in 1976 as C2-4P and has grown to a 40K backplane system with PIA, OKI Microline 80 printer, Zenith green tube, D&N memory and disk board, TANDON Disk (5 1/4"), cassette and high speed baud rate generator.

I've been a reader since your beginning...please hang in there and keep publishing! I like the simple do-something programs.

"MAGIC SQUARS" PROGRAM

```
0 GOTO2
2 GOSUB5
3 GOTO40
5 A=PEEK(129):B=PEEK(130):POKE129,192:POKE130,215:S$=" "
10 FOR S=1TO62:S$=S$+" ":NEXT:POKE129,1:POKE130,8:RETURN
40 I=0:J=0:Q9=0:M=0:Q=0:M9=0:P=0:T=0:C=0:R=0:S=0:V=0
60 PRINT"Number squars"
70 PRINT"--------------"
80 POKE56832,0:PRINT
85 :
90 PRINT"Welcome to the world of"
100 PRINT "confusion. There are two"
110 PRINT"versions of Squars"
120 PRINT:PRINT"    SEQUENTIAL"
130 PRINT:PRINT"     & ":PRINT
135 PRINT"     MAGIC SQUARE":PRINT
138 PRINTTAB(3);
140 PRINT"Which is your pleasure ":PRINT
145 INPUT"1 FOR SEQUENTIAL 2 FOR MAGIC";T
150 IF T=1 THEN350
160 IF T<>2THEN140
170 :
180 REM   SET UP MAGIC SQUARE
210 FORI=1TO4
220 FORJ=1TO4
230 READM(I,J)
240 B(I,J)=M(I,J)
250 NEXTJ
260 NEXTI
270 DATA 1,6,15,8,12,11,2,5,10,13,4,3,7,16,9,4
280 I1=4
290 J1=2
300 GOTO440
310 :
320 REM SET UP SEQUENTIAL SQUARE
350 DIMB(4,4)
360 FOR C=1TO4
370 FOR D=1TO4
380 B(C,D)=(C-1)*4+D
390 NEXTD
```

Continued

```
400 NEXTC
410 I1=4
420 J1=4
430 :
435 REM   SCRAMBLE BOARD
440 PRINT"I am now scrambling the board..."
450 PRINT"How difficult do you want it"
460 PRINT:INPUT" 10 to 500 ";Q9
470 PRINT
480 FOR Q=1TOQ9
490 M=INT(3*RND(1)+1)
500 ON M GOTO 510,560,610,660
510 IF I1=1GOTO490: REM M=1
520 B(I1,J1)=B(I1-1,J1)
530 B(I1-1,J1)=16
540 I1=I1-1
550 GOTO700
555 :
560 IF I1=4 THEN GOTO 490: REM M=2
570 B(I1,J1)=B(I1+1,J1)
580 B(I1+1,J1)=16
590 I1=I1+1
600 GOTO700
610 IF J1=1 THEN GOTO490: REM M=3
620 B(I1,J1)=B(I1,J1-1)
630 B(I1,J1-1)=16
640 J1=J1-1
650 GOTO 700
655 :
660 IF J1=4 THEN GOTO 490: REM M=4
670 B(I1,J1)=B(I1,J1+1)
680 B(J1,J1+1)=16
690 J1=J1+1
700 NEXTQ
705 :
710 REM PRINT BOARD
720 :
740 M9=M9+1
760 :
765 R=0:S=0
770 PRINT"--------------------------":REM 26'-'
775 U=0:V=0
780 FOR U=1 TO 4
790 FOR V=1 TO 4
800 PRINT":";
810 IF B(U,V)=16 THEN PRINT"    ";:GOTO840
820 IF B(U,V)(10 THEN PRINT" ";
830 PRINT B(U,V);
840 NEXT V
850 PRINT":"
860 PRINT"--------------------------"
870 NEXT U
875 :
880 FOR K=1TO1000:NEXTK
890 REM SOUND POKE 57832,#
900 REM INPUT MOVE
910 :
960 INPUT"Move which piece";P
970 I1=0:J1=0
975 H=0:G=0
977 :
980 FOR G=1TO4
990 FOR H=1 TO4
1000 IF B(G,H)=PTHEN I1=G:J1=H
1010 NEXTH
1020 NEXTG
1025 :
1030 IF I1=0 THEN PRINT"I can't find that #":GOTO960
1040 I2=0:J2=0
1050 FOR I=I1-1 TO I1+1
1060 IF I)4 THEN 1090
1070 IF I(1 THEN GOTO1090
1080 IFB(I,J1)=16 THEN I2=I:J2=J1:GOTO1170
1090 NEXT I
1095 :
1100 FOR J=J1-1 TO J1+1
1110 IF J)4 THEN GOTO 1140
1120 IF J(1 THEN GOTO 1140
1130 IF B(I1,J)=16 THEN I2=I1:J2=J:GOTO1170
1140 NEXT J
1145 :
1160 PRINT"Not a valid move!":GOTO960
1170 B(I2,J2)=P
1175 I=0:J=0:R=0:S=0
1180 B(I1,J1)=16
1190 ON T GOTO 1230,1360
1200 :
1210 REM SEQUENTIAL SOLUTION
1220 :
1230 C=0
1240 FOR R=1TO4
1250 FOR S=1TO4
1260 IF B(R,S)(C THEN GOTO 740 : REM NOT SOLVED
1270 C= B(R,S)
1280 NEXTS
1290 NEXT R
1300 PRINT"** YOU GOT IT **"
1305 PRINT"IN ";M9;" MOVES."
1310 GOTO 1450
1320 :
1330 REM MAGIC SOLUTION
1340 :
1360 FOR R=1 TO4
1370 FOR S=1 TO 4
1380 IF B(R,S)() M(R,S) THEN GOTO 740: REM NOT SOLVED
1390 NEXT S
1400 NEXT R
1410 :
1420 REM DECLARE WIN
1430 :
1440 PRINT"That is the correct solution!"
1450 INPUT"Want to play again ";A$:A=ASC(A$)
1460 IF A()78 THEN GOTO  1500
1470 POKE56832,1:END
1500 CLEAR:RESTORE:GOTO 2
1900 REM 9/30/84
1910 REM BASED ON IDEA IN KILOBAUD 2.81 PAGE 114
1920 REM BY DR. LEAVEY.
1930 REM OSI VERSION BY R. GROOME V1.0 1983
1940 REM RELEASED FROM ALL NON-COMMERCIAL USES
2000 REM   C/D,G/H
2003 REM   I/J,R/S
2004 REM   I/J,R/S FOR/NEXT COUNTERS
2005 REM   T       MAGIC OR SEQUENTIAL
2010 REM   M       MIX UP BOARD
2015 REM   Q9      HOW MUCH
2020 REM   M9      # MOVES
2030 REM   K       TIME DELAY
2035 REM   P       PIECE TO MOVE
2040 REM   C       CHECK FOR SOLUTION
2045 REM   A$      PLAY AGAIN PROMPT
2050 REM   A & B   MARKERS FOR SC ROUTINE
```

LETTERS

ED:

As you may recall, I have been working with the WP6502 word processor coding. On page 18 of the WP6502 manual there is a paragraph: "Pressing Break Key Accidentally". It gives instructions to recover, but with the BAD NEWS that the disk operations will not work! Then a note that OSI is aware of the problem. The other day, I had the misfortune of accidentally hitting the break key while typing. They are correct, the disk operations will not work.

If OSI was aware of the problem, they did not do anything about it. The problem is not in WP6502. It is in OS-65D versions V3.2 and V3.3. If you boot up the system and then hit the break key, then (M) to go to the ROM Monitor and (.2547G) to go to DOS., the A* prompt comes up on the screen. However, none of the disk operations will work.

Since my WP6502 uses V3.2 and this is the version that I disassembled, I worked with it first.

From the ROM disassembled coding, it was found that the (D) response directs the operation to coding which initializes the disk PIA and ACIA, and the Dev 1 ACIA. It then reads track zero into memory and transfers control to Coldstart at $2200. In Coldstart for some reason the disk PIA is initialized again, different than in ROM.

The ROM Monitor with (M) does none of the above done by (D), so this is part of the problem.

Because an Assembly language

program, (FIX), to initialize the disk PIA and ACIA with a jump to $2547 did not fix the problem, a more complete study of ROM was made. It was found that in the path of (.nnnnG) there was not a setting of the Drive selection and that a "push to the stack" was made without a balancing "pull from the stack". The program FIX was changed to select drive A and to do a pull from the stack. GOOD NEWS, the disk operations worked!

In DOS V3.2 there is almost one page of open coding called DOS EXTENSION. It starts at $3179. OSI put three commands in this space, so it is open from $3180. In my system I have put a subroutine in this location which moves the open space to $31A2 which is where FIX is now located. This part of DOS is on Track 1.

With FIX in the system, after a (BREAK), a (M) (.31A2G) will transfer control to DOS A* and all commands work including the disk operations.

With FIX on the WP6502 disk, after a (BREAK) do (M) then (.026BG) which brings up the WP6502 prompt A! then (GO 31A2) will reset the system. The (.026BG) transfers control to WP6502 which is necessary because of changes which WP6502 makes to DOS in order to return to WP6502 after a disk operation.

The program FIX has not been added to V3.3 because all of DOS EXT. was used on V3.3 and I do not know at this time where some open space exists to put FIX.

Some of your readers may be interested in this problem fix for DOS. A listing of FIX is shown. If anyone knows where it will fit into V3.3, I would like to know.

```
 10  *=$31A2     ;START ADDRESS
 20  LDA #$00    ;RESET DISK PIA
 30  STA $C001
 40  STA $C003
 50  LDA #$40    ;INITIALIZE PIA
 60  STA $C000
 70  LDA #$FF
 80  STA $C002
 90  LDA #$04
100  STA $C001
110  STA $C003
120  LDA #$03    ;RESET ACIA'S
130  STA $C010   ;DISK
140  STA $FC00   ;DEV #1
150  STA $FB00   ;TWO SERIAL PORTS ADDED
160  STA $FB02   ;TO MY SYSTEM
170  LDA #$58    ;INITIALIZE DISK ACIA
180  STA $C010
190  LDA #$11    ;INITIALIZE SERIAL PORTS
200  STA $FC00
210  STA $FB00
220  STA $FB02
230  LDA #$01    ;SET TO SELECT
240  JSR $29C9   ;DRIVE A
250  TSX         ;TO GET STACK IN ORDER
260  JMP $2547   ;GOTO DOS
900  .END
```

J. Edward Loeffler, Jr.
Huntsville, TX 77340

* * * * *

ED:

I have a C4PMF with OS-65D V3.2. I am working on some applications using the serial port in a character-by-character mode. The users manual indicates FC00 as the port register and FC01 as the status register and indicates how to set the baud rate by POKEing FC01. However, some additional information would be helpful:

a) I recall reading in PEEK that there is a register to be POKEd to inform the system whether the serial port is used as a modem or a printer. In combing over my past issues, I can't find that information. Could you please repeat it?

b) I have gleaned some information from articles in PEEK on the use of FC01 to report on the state of the port. What are the possible states and their meanings?

T. G. Moore
Freehold, NJ 07728

T.G.:

The serial port on the C4P-MF is a standard 6850 ACIA that is routed through either a PIA or a UART (I forget which), which in turn selects one of the two DB-25 connectors on the back of the system. The address of the PIA (or UART) is $F7D3. POKEing this location with $34 selects the modem connector and $60 selects the printer connector. $FC01 is a data register. PEEKing that location is only significant when there is an incoming piece of data ready for retrieval. $FC00 is the status register and PEEKing this location will tell you if there is any data waiting, but little other information normally available from a 6850 can be gleaned from this port due to the fact that OSI hard wired some of the other pins to always show ready.

Rick Trethewey

* * * * *

ED:

For sometime I have wished for a simple program that would automatically switch from drive A to drive B if a program was not located on drive A. The 3.3 version of OS65D has the TRAP statement and it will, when enabled, jump to a line number when an error is encountered in a program. The following short program will accomplish the function that I wished for.

When run, the program first looks at the A directory and if the file is not found will issue a #C error and also print the statement that the file is not on this drive. It will then activate drive B and search its directory and will load the requested program. If it doesn't find the file on drive B, it will then prompt for another try or load BEXEC*, as desired.

I have found this simple program to be useful and hope others find it of use also. When a program is found, several error indications will be output. The first will be SN and then US, followed by OK.

```
62000 TRAP 62050:REM DUAL DRIVE LOADER
      PROGRAM
62005 REM M.BERNSTEIN, ASBURY PARK, NJ
      10/23/84
62010 DISK! "SE A"
62020 INPUT "FILENAME";A$
62030 DISK! "LOAD "+A$"
62050 PRINT"FILE NOT FOUND ON DRIVE A"
62060 TRAP 62100
62070 DISK! "SE B"
62080 DISK! "LOAD "+A$"
62100 PRINT "FILE NOT FOUND ON DRIVE B"
62110 INPUT"TRY AGAIN (T) OR RUN BEXEC*
      (B)";B$
62120 IF B$="T"THEN 62000
62130 RUN "BEXEC*"
```

NOTE: A SPACE MUST FOLLOW THE WORD 'LOAD'.

M. Bernstein
Asbury Park, NJ 07712

* * * * *

ED:

In regard to Gary Florence's letter in the Dec '84 issue of PEEK(65), regarding tape to disk conversions, I have con-

verted the Minos (Maze) program to use on a ClP under HEXDOS. I can't remember all the details of conversion, but it does require altering and relocating the machine code portion. Perhaps I'll try to write it up someday. The Tiny Compiler is available to HEXDOS users from the HEXDOS user's library (c/o Vern Heidner, 1440 Co. Rd 110 N., Mound, MN 55364).

Jim McConkey
Rockville, MD 20855

Jim:

Don't stop there. You have just whetted our appetite. Please do tell us the details of the conversion process - hardware and software. I am sure that there are others in the same boat, but just don't know how to go about it.

Eddie

* * * * *

ED:

I have just acquired a Grafix SEB-3 80 column board for my C2-8P. Does anyone have any information about its capabilities, etc.? Please ask if any current users will write about their experiences with this board.

Thanks!

Alen Cohen
Staten Island, NY 10312

Readers:

Please help!

NEWS RELEASE

Sierra Madre, CA., January 9. -- The "Third Wave" officially arrived today with the announcement of new organization designed to support the growing number of people who work in their homes with personal computers. The newly formed Association of Electronic Cottagers will bring focus to this group, foreshadowed by Alvin Toffler in his best-selling book "The Third Wave."

"We will provide actual business services to both computer entrepreneurs and telecommuters who work at home on a salary," the group's founders, husband-and-wife team Paul and Sarah Edwards, said in announcing the group's formation.

Members of AEC can obtain marketing assistance, business consultation and other services. They can also access up-to-the minute news about local, state, national and international developments affecting their interests through a monthly newsletter, an online hotline, bulletin boards, electronic conferences and private databases available to AEC members through CompuServe Information Service. Aspiring cottagers can get help finding work at home and assistance in setting up a computer-based business.

Electronic cottage members are already mobilizing to protect their rights to work at home with a computer by opposing AFL/CIO efforts to ban telecommuting and by setting forth the Electronic Cottage Bill of Rights.

Those interested in AEC can write the Association for free information at 677 Canyon Crest Drive, Sierra Madre, CA 91024. CompuServe # 76703,242.

AD$

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

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

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

Buy one box of 6 cartridges for $8.00 and we will give you a second box FREE. That's 66.66 cents a piece or 83% off. At that rate, how can you lose? Add $3.00 for postage and handling. Make check or money order (in U.S. funds, drawn on a U.S. bank) payable to PEEK(65). P.O. Box 347, Owings Mills, Md. 21117. Order NOW, supply limited!

* * * * *

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

* * * * *

C3C 56K 2-USER OSU/OSDMS/HDM, DUAL FLOPPY, AMCAP LEVEL 3 BUSINESS SYSTEM, 2 HAZELTINE 1520's. $4000/OFFER. Paul Drummond, P. O. Box 2057, Woodland, CA 95695, 1-916-661-6600.

* * * * *

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

* * * * *

FOR SALE: OSI UTI Board with Vortrax, CBT Coupler, software & documentation. $200.00 or best offer. (Terry) 512-824-7471.

* * * * *

CUSTOM BUILT C8P. Professionally assembled with dual 8" drives, cassette interface, RS-232, parallel board, Hi-res color, ten key pad & joysticks. Works perfectly ... I need the money. Over 30 disks including MDMS, OS65D V3.3, OS65U, WP-65 Word Processor, numerous financial programs, personal accounting & games. 10" green BMC monitor. $750 firm. Phone (918) 333-5043 or 661-7998.

* * * * *

FOR SALE: C3-C, 1 MHz, 23 MByte, 3 user computer system with Dual 8" double sided floppy drives; includes three Televideo 925 Terminals and Qume Sprint 5 printer. Would consider selling this system as component pieces. Asking $250.00 each for the terminals, $300.00 for the Qume, and $1,100.00 for the C3-C-23. If purchased together, asking $1,900.00. Boards already installed in the system include 470, 510, 590/525, CA-9, CM-4, 535, 555-4 for terminals, 555/2 for serial printers, and 2 CM-20 memory boards. Includes OS-65U Ver 1.42. The system is fully operational. This is a great entry system, or could be used as spare parts for an existing installation. For more information, call (216) 743-3186 between 9:00 A.M. and 5:00 P.M. EST. Ask for Marilyn.

* * * * *

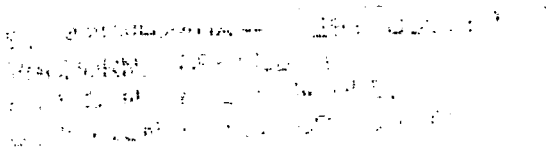
WANTED: C3-B or C3-C in good working condition. Also, tape back-up. Call Richard (201) 666-3250 (NJ).

PEEK (65)
The Unofficial OSI Users Journal

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

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

DELIVER TO:



GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

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

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

( ) Cash enclosed ( ) Master Charge ( ) VISA

Account No. _____ Expiration Date _____

Signature _____

Name _____

Street _____

City _____ State _____ Zip _____

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

POSTAGE MAY VARY FOR OVERSEAS