

# PEEK (C5)

JULY 1985  
VOL.6, NO.7

The Unofficial OSI Users Journal

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

## Column One

Here it is July again - time of national celebration and vacations. A time to sit back and ponder where it is all leading and where we each fit in.

While you are vacationing, take PEEK along. This issue is full of things that will make you think about significant improvements to your hardware and software, but none of them are one-line quick fixes or a simple cut jumper.

Just contemplate how an 80 fold speed increase might influence the operation of your software package by implementing Dave Livesay's 68000 co-processor. For those who would rather use their heads instead of wallets, consider rewriting your GOTOS and GO-SUBs to take advantage of how BASIC really finds that line, as related by TOSIE.

For those new (and not so new) to OS65-U, and that in particular means you business users who are not getting every ounce out of your machines, John Whitehead continues his explanations and Roger Clegg shares very useful programmer tools.

Had you considered the FORTH Op. Sys., but never really knew what it was all about? You will if you read what Charles Curley has to say!

Roy Agee is back at it again with another thought provoking piece on how to train computer users so that you won't have to train them again ... again ... again.

Dan Sweger has been advertising his SCRIBE word processor for some time. Now we have a review and example of how its versatility is being utilized in the real world. And Tom McGourin sheds more light on WP-2/3's DIR.

Who says we have forgotten the cassette? We strike twice with the cassette assembler bug fix and token load and save.

For all "P" users, Dave Pompea's piece on how to save your MF disks by unloading the head and turning the drive off is a must. This is a mod we feel should be implemented on all MF machines. If you are not up to making the little board, wait until next month for the simple mod that just lifts the head, but why not go all the way? While you are at it, read how Jankowski, true to form, speeds up the screen dump.

That leaves us with Dennis Shoulder's description of how yet another unique real world problem was solved by a little creative thinking. As we have said before, Dennis is not the only one to find a unique use for our machines, our problem is in getting you to follow his lead in sharing the experience.

Even if you don't have a unique situation to write about, please do drop us a line or two (we'll forward them) about the articles you read. A little positive feed-

## INSIDE

OS-U PROGRAMMING AIDS PART 2	2
THE OS1-68000 SYSTEM	3
THE COMPUTER AS A TOOL	6
SORTED DIRECTORY FOR WP-3/2	8
GO FORTH AND MULTIPLY	8
ASSEMBLER/EDITOR BUG & FIX	12
FILE HANDLING OS65-U & OS-DMS	12
SCRIBE WORD PROCESSOR REVIEW	14
SILENCE YOUR DISK DRIVE	16
SCREEN DUMP/DISK BASED SYSTEMS	17
TOKEN LOAD/SAVE PROGRAM	18
BASIC SPEED & LINE NUMBERS	19
DEALER PROFILE	20

back does great things for the author's ego and encourages more articles.

On a more somber note, we've received word that Ian Eyles, who has done so much for Australia's OSI newsletter is quite ill. If you can find time, I am sure that he would appreciate a "get well" note addressed to him c/o KAOS, 10 Forbes St. Essenden, VIC 3040, Australia.

On the manufacturing front, we have received, at last, but too late for this issue, all the "poop" on the 700 series machines. It will be in next month for sure. We are told that ISOTRON is accepting orders on the 710 which is ready to go except for the documentation which is due this month and will probably coincide with a dealer training program to bring dealers up to speed on the machine and UNIX. Meanwhile, over at DBI, things are rolling full steam ahead and they have just announced a new leasing policy suitable for the purchasers of smaller machines.

So there you have it. Be a patriot and enjoy the 4th.



## OS-U PROGRAMMING AIDS PART 2

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

### OS-65U DEBUGGING TOOLS

(1) Typing FLAG 7 in the direct mode (that is, when no program is running), then RUN, will display each line on the screen before it executes. The only tricky points are that a line with a GOSUB or FOR in the middle is not displayed again when the program returns, and that the lines may be deleted by a screen clear or cursor up before you can read them. The second problem can be fixed on the Hazeltine 1420 by moving dip switch #6 on the left, which changes the lead-in code for screen clear, etc.

The display runs too quick to read. You can use Control-S to pause and Control-Q to resume, or Control-C to break and CONT to resume, but the best tools are NULL 255 (or POKE 21,255), which slows down the screen display, and POKE 11686,17, which duplicates the display on printer #5. POKE 11686, 1+2^(D-1) for printer #D.

These tools can be inserted as program statements if you want a fast run up to the critical routine. They are turned off by FLAG 8, NULL 0 (or POKE 21,0), and POKE 11686,1.

(2) Insert STOP statements at various points. When the program breaks, you can tell BASIC to print the values of suspicious variables, change their values, or list some lines, then you can CONT to the next STOP, RUN again, or GOTO a different line. You can't change a line or use the

Editor without losing the variables, and if you interrupt a listing by Control-C and then CONT, you may get a "?NF ERROR". Remember to remove the STOP statements after debugging.

(3) If using FLAG 7, you may prefer Control-C to the STOP statement as it has more flexibility. The same cautions apply.

(4) If variable X is acquiring a strange value, you can insert lines like PRINT CHR\$(13) X; at various points. (The CHR\$(13) and ; prevent the screen from scrolling.)

(5) To debug a seldom-used routine, you can go there from the direct mode, for example:

DV\$="A": D=1: NA\$="TESTCASE":  
TY=3: GOTO 18000

or you may prefer a debugging line early in the program, or just before the menu:

98 NA\$="TESTCASE": FOR TY=1 TO  
6: GOSUB 18000: PRINT TY,X\$:  
NEXT: STOP

(6) Even when debugging is apparently complete, it is good practice to leave in place lines to catch "impossible" errors.

For example:

65 IF DV\$>"E" THEN STOP  
A STOP statement like this uses little space, and if it is ever hit, the message "BREAK IN 65" immediately directs you to the problem without the necessity of a specific error message.

### HOW TO TRANSFER LINES FROM ONE PROGRAM TO ANOTHER

Say lines 7100-7190 in "PGM1" to become lines 550-595 in "PGM2".

A "SCRAT" file is needed. There should be one on your Utility disk.

(1) If you have fewer than 10 lines to move, LOAD"PGM1", renumber them one by one using the Editor, swap in the Utilities disk, and go to (10).

(2) Insert Utilities disk, RUN"RSEQ".

(3) Swap in the "PGM1" disk, LOAD"PGM1", swap back the Utilities.

(4) Type: RSEQ 1,,1 This gives the lowest possible numbering.

(5) LIST. If the lines you need are now numbered lower than 550-595, say 413-422, then RSEQ 550,413,5 and go to (10).

(6) Suppose the lines required are numbered 608-617.

Type: CLOSE: OPEN"SCRAT",1:  
LIST#1,608-617: PRINT#1,"OK":  
NEW

(7) Type: INDEX<1>=0: FLAG 13:  
INPUT#1, (notice the comma). There should be a couple of seconds delay and then "?SN ERROR".

(8) LIST and check lines 608-617 are there.

(9) Type: RSEQ 550,,5 (renumber from 550, interval of 5).

(10) Type: CLOSE:OPEN"SCRAT",1:  
LIST#1,550-595: PRINT#1,"OK":  
NEW.

(11) Swap in the "PGM2" disk, LOAD"PGM2", swap back the Utilities.

(12) Type: INDEX<1>=0: FLAG 13:  
INPUT#1, (notice the comma). There should be a couple of seconds delay and then "?SN ERROR".

(13) LIST and check lines 550-595 are there.

(14) Swap in the "PGM2" disk and SAVE.

(15) LIST 550-595. Look out for truncation of extra-long lines which were originally entered by using "?" as an abbreviation for "PRINT".

(16) If any lines are truncated, reboot with the Utilities disk or whatever disk you usually use, LOAD"PGM2", fix the truncated lines, and SAVE.

### HOW TO DELETE A BLOCK OF LINES

Say from 6000 to 8990, a "SCRAT" file is needed. There should be one on your Utility disk.

(1) LOAD your program, swap in the Utilities disk.

(2) CLOSE: OPEN"SCRAT",1  
FOR I=6000 TO 8990 STEP 10:  
PRINT#1,I: NEXT: PRINT#1,"OK"  
This puts blank lines into "SCRAT" for merging with your program. If the lines are not numbered in 10s, omit the STEP 10.

(3) INDEX<1>=0: FLAG 13:  
INPUT#1, There should be a delay and then "?SN ERROR".

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

Editor - Eddie Gieske

Technical Editor - Brian Harston  
Circulation & Advertising Mgr. - Karin Q. Gieske

Production Dept. - A. Fusselbaugh, Ginny Mays

Subscription Rates

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

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

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

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

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

(4) LIST and check the lines are gone, then swap in the right disk and SAVE.

#### OS-65U RESERVED WORDS

ABS	FN	NEXT	RUN
AND	FOR	NOT	SAVE
ASC	FRE	NULL	SGN
ATN	GOSUB	ON	SIN
CHR\$	GOTO	OPEN	SPC(
CLEAR	IF	OR	SQR
CLOSE	INDEX	PEEK	STEP
CONT	INPUT	POKE	STOP
COS	INT	POS	STR\$
DATA	LEFT\$	PRINT	TAB(
DEF	LEN	READ	TAN
DEV	LET	REM	THEN
DIM	LIST	RESTORE	TO
END	LOAD	RETURN	USR
EXP	LOG	RIGHT\$	VAL
FIND	MID\$	RDN	WAIT
FLAG	NEW		

The word NULL is sometimes replaced by RSEQ, SWAP, KILL or PNTR.

A variable name cannot contain a reserved word. Beware particularly of ON, OR and TO.

The only words that cannot be used in the immediate mode are INPUT, DEF and DATA.

The only word that cannot be used in a program is CONT, although LOAD, SAVE, LIST and RSEQ would not normally be in a program.

#### DRESSING UP THE PROGRAM

Blank lines and indentation can be achieved by entering the program using colons, and then inserting these temporary lines:

```
7 FOR I=24576 TO PEEK(122)+256
 * PEEK(123)-5:IF PEEK(I)
 THEN NEXT
8 I=I+5: IF PEEK(I)=58 THEN
 POKE I,32
9 NEXT: STOP
```

Running the program will replace leading colons with spaces.

Then delete the temporary lines.

If there is machine code ahead of the program then 24576 in line 7 must be replaced by 24576+PEEK(24572)+256\*PEEK(24573).



#### THE OSI-68000 SYSTEM

By: David M. Livesay  
Ave de la Resistance No. 6  
B-4920 Embourg, Belgium

At the request of PEEK(65), I've decided to take pen in

hand, or should I say computer, and tell the story of how a Motorola 68000 became attached to the OSI. This article describes the history of the 68000 Attached Processor built by Digital Acoustics and my work in building an interface and developing the software to use it with the OSI. The combination results is a relatively low cost but very high performance 68000 system for the OSI user. This system enables you to learn to program the 68000, speed up BASIC, use it as a RAM disk and run complete languages and operating systems in the 68000 and still retain your OSI hardware.

I will go back aways to about 1980. At that time a company called Digital Acoustics had for some time been in the environmental noise analyzer business. They were using the 6502 and were looking for something more powerful. This was about the time the 68000, Intel 8086 (and other Intel), T.I. 9900 series, National 16032 and various Zilog processors had just been announced or were coming on the market. Their investigations showed that the Motorola 68000 with its 15 general purpose 32 bit registers, high speed, large linear address space of 16+ megabytes and powerful instruction set was the best candidate for the job. They also felt that the 68000 had the best long term potential. At that time the 68000 looked like it would be around for at least 10 years. It now looks like it and its family members will be with us for at least 15 more years.

Back then and even to some extent today, Motorola promotes the 68000 as being a microprocessor for expensive complicated systems. Digital Acoustics was not interested in tying 16 terminals to one microprocessor. They just wanted to have a very powerful processor doing one job. So after studying the data sheets and doing a little experimenting they concluded that yes indeed you could use the 68000 in a relatively simple system.

About the time they decided that the 68000 was the processor for their new generation noise analyzer, the market for noise monitors dried up. So they came up with the idea of building a 68000 board to attach to a microcomputer such as the PET or the Apple. Both the PET and the Apple version became available at the end of 1981. The boards were avail-

able with 4K to 92K of static RAM with the 68000 running at 8, 11 or 12.5 MHz. During the next year, they came out with a 128K expansion board and more software. Due to the lack of an Assembler in those days, all of the early code was produced using a program called the Hand Assemblers Helper. In 1982 a company called Phase Zero came out with a cross assembler that would run on the Apple. The Digital Acoustics software at the end of that year consisted of several impressive graphics demonstrations including a 3-D airplane that would rotate and move in 3 axes, a Microsoft compatible floating point package which hooked into BASIC, a monitor program and several demonstration programs. Since that time other people have developed PASCAL, FORTH, COMPILED BASIC and other software to run on the Digital Acoustics boards.

Now, this brings us to the end of 1982. As we were making our annual trip home to California, I was reading an issue of Micro perhaps the September 82 issue. Anyway, I read an advertisement for the Digital Acoustics board which could be attached to the Apple. I had been a long time OSI and 6502 user and I was also beginning to think that I wanted something faster and more powerful. After looking at several of the new processors that were coming on the market, I had decided that the 68000 was the processor on which I would focus my attention. My impression was that just as the 6502 was much easier to program than the 8080, the 68000 would be much easier to program than the 6502. So right there on the plane, I decided that I would purchase the Digital Acoustics board and attach it to the OSI.

Right after Christmas I drove up from San Diego to Santa Ana to visit Digital Acoustics. It turned out that nobody but the owner was there. So he gave me a demonstration of the board on the Apple. They didn't have any spare boards to sell but after explaining that I was living in Belgium and would have trouble getting the board at a later date, he not too reluctantly parted with one board with 4K of memory on it. I then walked out with one largish board and quite a bit less money than I had walked in with. At that time a board with 4K cost \$683 plus tax!

In late January 1983 upon our arrival back in Belgium, I unpacked my board, set it next to the OSI and left it there for a few months while I tried to catch up on some of my real work. About May I noticed that the 68000 board was not going to mate itself to the OSI without some help. I had purchased the Apple version of the board which includes a small interface board to plug into one of the Apple expansion slots. I obtained a schematic of the Apple so that I could determine how the Apple interface worked. After about a week or so of work in the evenings, I had an interface board that plugged into the OSI and the Apple interface board in turn plugged into it. I then wrote a short program that would allow transferring data back and forth between the OSI and the 68000 so that I could test out my interface. Everything worked as it should so I proceeded on to the software.

Now this is when the fun really started. I wanted to start with hooking the 68000 floating point routines into OSI BASIC. I didn't know where to look for the math routines in BASIC so I started to disassemble BASIC. At one point I thought that I could change the function jump table to jump to the code that sends the data to the 68000 and waits for the results. I manually entered the 68000 code and the OSI utilities code and stored it on disk. When I tried to run BASIC I found that it sort of worked. I then discovered that some of the math routines called other math routines. I decided that I would need to place the hooks into the BASIC math routines themselves. So I went back to disassembling BASIC. Not too long after this I received a copy of a commented disassembly of OSI Microsoft BASIC (OS65D version) copyright by M. K. Miller. I don't know if it's still available, but I found it to be very complete and very useful. With the use of this document I was able to finish getting the 68000 floating point package to work after a few weeks. All in all, I spent about 4 months to do what could have been accomplished in about 1 month if I had had the correct information to begin with.

When this was done, I decided to build a printed circuit board to replace my wire wrap interface board. I layed this out and included an OSI bus interface which includes all

of the main signals plus some additional ones to make it easy to interface some types of hardware. About this time I also thought I might be able to sell some of this hardware and software. This brought me to the first part of 1984. While I was waiting for the prototype boards to be finished, I continued with the software and got the Hand Assemblers Helper running on the OSI plus got a monitor program running and wrote a few brief demonstration programs.

With that not too brief background, I will describe the status of the hardware today. The current hardware consists of the Digital Acoustics dynamic RAM CPU board with 128K to 1024K (1 megabyte!) running at 12.5 MHz with one wait state all connected to the OSI with an interface board that plugs into the OSI 48 pin bus. For those with a C4 machine with all slots filled, the 68000 can also be interfaced through the 16 pin expansion bus. The 68000 system sits outside the OSI and connects with a 40 conductor ribbon cable. The 68000 board is about 15" X 6.5", and Digital Acoustics sells a stainless steel case designed to hold about four boards and a 10 amp power supply. This case is 4 inches wide by 8 inches high and about 19 inches deep. The signal lines consist of two 8 bit data paths (input and output) plus 7 other signal lines used for status, reset and the handshake signals. The OSI can write and read data, write and read status and reset the 68000. The 68000 can read and write data and read status. Data under full handshake control can be transferred at a rate of 45000 bytes per second with a 1 MHz OSI system. With a 2 MHz system the transfer rate will be about 90000 bytes per second. Since this is significantly above the average disk transfer rate, it isn't a system bottleneck. In order for all this data transfer to take place, both microprocessors must have routines to handle the transfer. The 68000 has a built in monitor PROM that allows it to communicate right after reset. On the OSI side the routines must be supplied in either machine code or BASIC. Normally, this will be in machine code for reasons of speed. In addition to the 68000 board, Digital Acoustics builds two high resolution graphics controller boards which can be attached to the 68000 board and a math chip board for the National Semiconductor math chip. The

two graphics boards are now used in combination with the 68000 boards in a CAD system sold by Cascade Graphics. Digital Acoustics is now working on a new graphics controller board which will use a second generation graphics chip. This I suspect will be supported with software and should be of interest to the serious hobbyist.

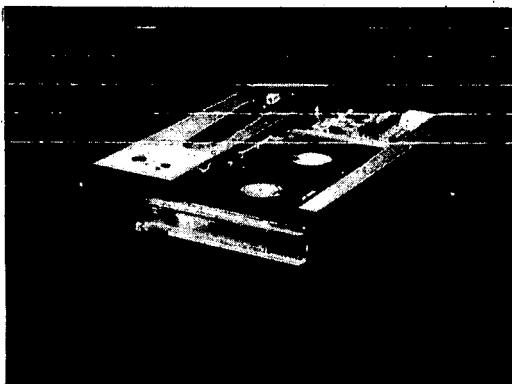
So now what can we do with all of this hardware? One of the ways that we can use it is to treat the 68000 as a peripheral to the OSI. In this mode we place some code into the 68000 which will upon command from the OSI do something for us. One example of this is using the 68000 to perform the math routines in BASIC as was already described. In this case the 68000 just sits around waiting for the OSI to send it a command plus the arguments of a math function. While the 68000 performs the math function, the OSI waits for the results and then returns to BASIC. Other examples of uses where the 68000 would be a peripheral are: As a RAM disk, storage of variables for BASIC, routines to perform sorting of lists, etc. There is in fact another way in which we can use this OSI-68000 combination. We can place the 68000 in charge and the OSI becomes an input/output processor. At first sight this may sound as if I said I will hook up a printer to my OSI and the printer will be in charge. But what happens is this; we place a program into the 68000 which could, for example, be a language and operating system, and then start that program running. We then place the OSI into a machine language program which waits for commands from the 68000. These commands can be such things as read the keyboard, write to the display, read a disk, send data, etc.. So it's possible to have a real 68000 system with the 68000 in total control. I have one example of a language running in the 68000 which I will describe later.

Now that I've described the hardware and a little about how we can use software with it, you might ask what software is available. At this time, the OSI interface board comes with the following software:

1. Utilities code for data transfer between the 68000 and the OSI.
2. A floating point math

# D.B.I., inc.

p.o. box 21146 • denver, co 80221  
phone [303] 428-0222



Wangtek sets the industry's standard for excellence in 1/4-inch streamer technology because its tape drives are all created with an uncompromising dedication to the highest possible quality in design, engineering and manufacturing. These factors combine to give the Wangtek 5000E tape drive a level of performance and reliability that is unexcelled in today's marketplace.

The Wangtek 5000E is uniquely suited to meet the backup demands of today's smaller size, higher capacity Winchester-based computer systems—it packs up to 60 MBytes of data storage in a compact, half-high form factor only 1.625 inches tall. For added user convenience, the drive accepts and automatically adjusts gains for either standard 45 MByte tape cartridges (450-foot cartridge) or high-capacity 60 MByte cartridges (600-foot cartridge).

## WHAT'S NEW AT D.B.I. ???

What's the answer? The DMA 360 removable 5 1/4" Winchester. It's exactly the same size as a 5 1/4" half-height floppy drive—but that's where the similarity stops.

The DMA 360 gives you hard-disk reliability. Floppies don't.

The DMA 360 protects your data in a totally sealed cartridge. Floppies don't.

The DMA 360 packs 13 megabytes (10 formatted) on a single ANSI-standard cartridge. It takes up to 30 floppy disks to achieve an equal capacity.

The DMA 360 even has a lower cost-per-megabyte than a floppy. But it gives you so much more.

Like an average access time of 98 milliseconds. A transfer rate of 625 kilobytes per second. And an error rate on par with the most reliable conventional Winchester disk drives.

### DMA Systems half-height removable 5 1/4" Winchester.



**FOR PRICING AND DELIVERY CONTACT YOUR NEAREST D.B.I. DEALER!!!**

\*WANGTEK 5000E is a registered trademark of WANGTEK CORPORATION

\*DMA 360 is a registered trademark of DMA SYSTEMS

package that hooks into BASIC.

3. A 68000 monitor written in a combination of BASIC and Assembly. The monitor allows for sending code to and from the 68000, changing data in the 68000, saving to disk, dumping the registers and dumping memory from the 68000.

4. A memory test program.

5. Some demonstration programs.

6. The Hand Assemblers Helpers.

The Hand Assemblers Helper is a menu driven program which will walk you through the 68000 instruction set and help you to hand assemble 68000 code. Since a real cross assembler will be complete in a few weeks, the real use for this program is as an aid in learning 68000 Assembly language.

In addition to the above software, a 68000 cross assembler is in the final stages of being debugged. This assembler will store the object files on disk and will, therefore, not be limited by the memory size of the OSI. A very fast BASIC like language is now up and running in a preliminary form.

The BASIC like language is called HALGOL. This is a development of Digital Acoustics and is designed to take full advantage of the power of the 68000. It is a language which is both interactive like Microsoft BASIC and compiled. Each time you enter a line, the syntax is checked and if all is well that line is compiled. If there is some error, an understandable error message is issued. When you have finished entering the program, it is already compiled and when you enter run, it will run right now and very fast. How fast? Well consider this line:

```
FOR I = 1 TO 5000: A=LOG(I):  
NEXT I
```

With a 1 MHz OSI system under OS65D3.3, it will run in 128 seconds. In HALGOL it will run in 8.47 seconds. (With the 68000 hooked into BASIC it will run in about 23 seconds.) In general the math functions will run between 15 and 25 times faster in HALGOL than in Microsoft BASIC. Also available for use with the Digital Acoustics boards is a math board which holds the National Semiconductor math chip. With this math board, the HALGOL

math routines will run about 60 to 80 times faster than Microsoft BASIC. HALGOL unlike Microsoft BASIC doesn't suffer from the search for line number problem that causes long programs to run progressively slower.

Other than speed, what are the features of HALGOL? HALGOL includes a screen editor which allows editing anywhere on the screen without entering an edit mode. Instead of entering EDIT 130, we just move the cursor up to line 130 and make the changes that we desire. The input buffer holds 255 characters which is about four times what we have on the OSI. Another important feature is that variable names can be up to 255 characters long. This means that unlike Microsoft BASIC which uses only the first two letters of the variable name, you can have names such as LENGTH1, LENGTH2, etc. and each one will be a unique variable. Variables can be either "local" to a subroutine or "global." How many of you have wished to be able to delete blocks of lines with one instruction when building a new program based on an old one? With HALGOL you can do it. A resequencing command is also resident in the language. This along with an APPEND command allows quick and easy building of programs using subroutines or modules from other programs. Another feature of HALGOL is that you use labels for subroutines. Instead of GOTO 5000 we might have something like GOTO "INPUT" or GOSUB "INPUT". The first line of the subroutine INPUT will look like this -100 "INPUT". This avoids the problem of changing all of the line numbers in the subroutine calling statements when you start changing your program. This language is still in its development stage but is now about 80% complete and you can already write useful programs with it.

At this time, all of the software runs under OS65D. HALGOL at this time can only be used on a video system, but I will have a version running on a serial system in a few months. Potential uses of interest to the business users community would be to use it as a RAM disk, to do sorting and searching of lists, to store the BASIC variables and to speed up some of the math routines. At this time, those are only suggestions because I haven't written the software for any of the above mentioned uses except the hooks into BASIC

for the math routines.

For those who are interested in a very high performance 68000 system, learning 68000 Assembly language, or need high speed computation power for either a scientific/engineering project or even a new business application, this system may be the answer that you're looking for.

For those who are interested, Digital Acoustics publishes a 68000 newsletter which is \$15 for ten issues. You can write to them at:

Digital Acoustics  
1415 E. McFadden, Ste. F  
Santa Ana, CA 92705

Now what's the cost of all of this? The price of a minimum configuration system with 128K of RAM, the OSI interface board and software is about \$800. A 1 megabyte system with case, power supply and OSI interface board with software will be about \$1400.



#### THE COMPUTER AS A TOOL A PROBLEM-SOLVING APPROACH

By: Roy Agee

What's the best way to teach people how to use computers? An "application" approach, based on the premise that mastery of specific application software is equivalent to mastery of the computer? Or a "problem-solving" approach, which provides mastery of the thinking skills required to use the computer as a problem-solving tool? Computer training has branched off in each of these distinctly different directions. This article will examine the methods and effectiveness of them both, making a case for the problem-solving approach.

The applications approach, generally advanced for business students and for in-house training, seems to have some flaws. Most glaring is the way in which it limits the use of the computer. Many of the specific software applications currently taught in schools -- spreadsheet and data base packages -- are management tools, rarely used by entry level employees.

In the case of word processing, the issue is more complex. A recent study reported

that under-use of computers is a common problem in offices where secretaries, the primary users of word processing software, have been inadequately trained in computer use. As a result, they are using expensive computer equipment merely as typewriters with screens. Along with word processing, filing and mailing software can relieve secretaries of time-consuming, repetitive tasks and allow them to take on additional responsibilities. But this can only happen if secretaries are trained, not just in word processing software, but in the use of the computer as a problem-solving tool. And of course, like a typist, a word processor without basic secretarial skill is of limited value to an employer.

For managers as well, the application approach is inadequate. Proponents of this approach argue that it's not necessary to know everything about a computer to perform a few tasks. But this view neglects to consider the computer as a tool for problem-solving. Just as a new driver learns how to drive the car and not just operate a few of its isolated features, like air conditioning or power windows, the new computer user should learn how to use the machine and not just a few specific applications. And a program which teaches the thinking and problem-solving skills required in using the machine is an essential to this learning.

The problem-solving approach to computer training benefits trainers and employers, along with trainees. Fiscal officers can appreciate the fact that personnel who can use the computer as a problem-solving tool will not need retraining with the addition of new responsibilities or the acquisition of more sophisticated application software.

The recent experience of a medium-sized company illustrates this point. The company was expanding and acquired several new, more sophisticated software packages. The software was allegedly "user friendly." The cost of the software was about \$2,000. The cost to retrain the employees was over \$7,000. Having learned the fundamental concepts of computer use and applications initially, these employees would have been able to decipher and use those new software packages without expensive additional training.

The first step in teaching problem-solving is to have a well thought out, fully developed and proven course of study. This course should be competency-based to allow the entry of people with varying computer skills. For beginners, the course should begin at the beginning: learning how to turn the computer on, and learning the functions of operating components.

The next logical step is to acquire a basic understanding of what makes a computer perform specific tasks. This function, known as programming, is the skill of writing a set of instructions which the computer can understand and follow. Nearly all micros have the BASIC language built in. Consequently, it is most practical to learn to write these instructions in BASIC. Using other languages would generally require that expensive additions be made to the microcomputer.

Once trainees have learned the fundamentals of BASIC, they can progress in a logical, sequential manner to acquire and improve problem-solving skills. This is accomplished while mastering the use of the microcomputer as a problem-solving tool.

A recommended sequence begins with a simple file procedure involving formatting output. This would be followed by such functions and procedures as: the computer as a calculator, disk and random access files, top-down design, subroutines, array processing, spreadsheet concepts, and batch processing. These applications and functions can be taught using a series of projects or problems with which the trainees are already familiar.

Using the foregoing procedure and sequence, trainees will compile a catalog of problem-solving techniques. Generic in nature, these skills will enable trainees to solve most any problem on nearly any type of computer; and with experience in most computer languages. Their only real limitations will be imagination, creativity, and the restrictions of a specific computer.

By learning these fundamental concepts of computer and software use, trainees will also be able to decipher and use most commercially-produced materials, design software to meet specific needs, and maintain and with source codes, modify privately and commer-

cially-developed software packages to meet specific needs. A properly structured and sequenced curriculum will enable trainees to use a wide variety of applications software packages, such as accounts receivable, inventory, billing, data base management, spreadsheets, using arrays, and so forth.

Additionally, trainees will have acquired and enhanced the kind of creative and innovative skills which are becoming increasingly more important for our changing society. The new era referred to as the "Information Age" will require more innovation and creativity than was needed or even desired during the Industrial Age. Many companies now have professional and technical

#### MEDIA CONVERSION

- 9 TRACK 1600 BPI TAPE
- 8 INCH FLOPPY  
(OSI 65U)
- 5 1/4 INCH FLOPPY  
(DBI FORMAT)
- IOMEGA CARTRIDGE  
(DBI FORMAT)

MED-DATA MIDWEST, INC.  
246 Grand  
St. Louis, MO 63122  
314-965-4160

#### OSI/ISOTRON

##### MICRO COMPUTER SYSTEM SERVICE

- \*C2 AND C3 SERIES
- \*200 AND 300 SERIES
- \*FLOPPY DISK DRIVES
- \*HARD DISK DRIVES  
CD 7/23/36/74
- \*TERMINALS, PRINTERS, MODEMS
- \*BOARD SWAPS
- \*CUSTOM CONFIGURATIONS
- \*CUSTOM CABLES
- \*SERVICE CONTRACTS

PHONE (616) 451-3778

COMPUTERLAB, INC.  
307 MICHIGAN ST. N.E.  
GRAND RAPIDS, MI. 49503

personnel working out of their homes. The computer, coupled with innovative, creative, critically thinking people, is vital to the success and continued growth of this phenomenon.

"Matters are at a crisis point in computing for many corporations," wrote Dr. Brandt Allen in the Harvard Business Review in 1982. Whether this crisis point becomes a major business problem depends on whether the challenge of computing technology is met and mastered by the training personnel. Fundamental changes are occurring at a more rapid rate than ever. The microcomputer has decentralized computer functions and operations. With the large, central mainframe, a relatively small, highly skilled group of people provides the computing services. The microcomputer is altering the structure of companies. Personnel in nearly all departments and levels of management are required to use a computer daily, in order to be efficient and effective in the performance of their duties. To keep the crisis point from becoming a business problem, these individuals require training in using the computer as more than a calculator or typewriter with a screen. The workforce of the "Information Age" must have the ability to use the computer with the same competence it now has with the telephone.

If the current trend of fractured and fragmented computer training continues, we will be retraining the workforce every time a new piece of software is developed. But people can be trained in the generic use of the computer as a tool in the same time or less time than that required to teach several specific and limited application software operations. Doing it right the first time can save time and money. Will the crisis continue, or will we meet and master the challenge? The choice should be clear.

Roy Agee is a computer education consultant for Career Publishing, Inc., Orange, CA. He is an author, lecturer, and educator who has been involved with the development of computer education since 1959.

#### SORTED DIRECTORY FOR WP-3 (and WP-2)

By: Tom McGourin  
216 West Michigan Ave.  
Kalamazoo, MI 49007

I happened to look at the directory program in WP-3 one day, and discovered that it has two sort routines--to sort the directory by name and by track. The program evidently came intact from WP-2 (the first REM says it's a WP-2 directory). The directory routine only invokes the sort-by-track routine, but with a little effort I was able to

have it run both sort routines, write the results into an array, and then print both to the screen.

I'm no expert in OS-65DA, so I don't know if there's an easy way to make the edits. I could use only the BASIC line editor--in other words, re-key each line, rather than editing the lines or line numbers.

There are four areas to be modified: declare the array, change the display header, fix the portion of the code that handles the UNUSED tracks in the table, and build the array.

The first:

60 DIM NM\$(64),TO%(64),T9%(64), NA\$(64),TI%(64),T8%(64)

The second is simple (there are three spaces between titles), and 34 and 31 dashes:

```
10026 PRINT DV,TAB(22);"OHIO SCIENTIFIC, INC."  
10030 RPINT DV,TAB(22);"WORD PROCESSOR WP-3-2":PRINT DV  
10035 PRINT DV,TAB(25);"-- DIRECTORY --":PRINT DV  
10040 PRINT DV,"FILE NAME TRACK RANGE PAGES ";  
10045 PRINT DV,"FILE NAME TRACK RANGE PAGES"  
10050 PRINT DV,"-----";  
10055 PRINT DV,"-----"
```

Now fix the UNUSED and put the track sort into the array:

```
delete 10111-10140  
10097 GOSUB21000:J=0:CT=AV  
10106 NM$="ZZZZZZZ"+STR$(J)  
10120 NA$(J)=NM$:T1%(J)=ST:T8%(J)=ET:J=J+1  
10130 NEXTI  
10140 ST=77:IFLE<76THENFORI=2T01:GOT010104  
10150 AV=J:FORI==OTOAV-1  
10160 NM$(I)=NA$(I):TO%(I)=T1%(I):T9$(I)=TI%(I):NEXTI  
10170 GOSUB20010  
10300 FORI=OTOAV  
10310 IFLEFT$(NA$(I),7)="ZZZZZZZ"THENNA$(I)="UNUSED ..."  
10320 IFLEFT$(NM$(I),7)="ZZZZZZZ"THENNM$(I)="UNUSED ..."  
10330 PRINT DV,TAB(1);NA$(I);TAB(13);T1%(I);TAB(17);"-";T8%(I):  
TAB(27);  
10340 SZ=T8%(I)-T1%(I)+1:A$="":IFSZ<1OTHENA$=""  
10370 PRINT DV,T9$(I);TAB(61);A$;SZ  
10380 NEXTI10400 PRINT DV:PRINT DV,CT;" OF 58 FILES DEFINED"  
10410 PRINT DV,  
PU;" OF 62 PAGES USED"  
10420 DISK!AS
```

Both directories are printed side-by-side, and fit on normal 8 1/2" wide paper.



#### GO FORTH AND MULTIPLY

By: Charles Curley  
5595 E. 7th St., #285  
Long Beach, CA 90804

Over the years, those in the know have sung the praises of FORTH while the rest of us have had to be content with mere crumbs of information. Thanks to Charles Curley's article, somewhat dated, but very viable, PEEK readers will

get a peek into the wonders of FORTH. This article originally appeared in the OSI Users Independent Newsletter.

I did my own implementation of FORTH for the simple reason that, at the time I started, there were no other implementations on the market. Obviously, a lot of other people had the same idea. Still, I found the experiment worthwhile, since I now know the



guts of fig-FORTH in a way very few people ever will. This knowledge is not entirely useless, as I have just started a job at FORTH, INC. as a programmer.

What is FORTH, anyway? Is it a programming language, an interpreter, a compiler, a disk operating system, an assembler, a way of thinking, or a way to warp your mind? The answer is, of course, yes! It certainly is a way of thinking, since it lets you do things that no other language will permit. Where BASIC, say, prohibits all sorts of things, FORTH permits not only many unusual operations, but also permits additions to the compiler to allow regular use of these unusual operations.

A case in point: FORTH does not ordinarily have a CASE statement. However, in the "Blue Sky Products" implementation and several others, you will find a case statement readily available. This is done by adding to the compiler. BASIC has an ON ... GOTO construct, which allows conditional execution. This is rarely used, but it is convenient. FORTH doesn't have one built in, but, because FORTH can be extended, one can be constructed. An execution table can easily be built,

because the programmer has immediate access to the guts of the compiler. Here, you do not need to modify the compiler, just know how to use it.

FORTH is heavily stack oriented. Almost all procedures look for their parameters on the stack, and a number of stack manipulation procedures are available in FORTH. Also, FORTH does not use the infix notation (e.g. PRINT 2 + 3) of BASIC, but the simpler reverse Polish notation (2 3 + . ). In my two examples, the operation is the same, print the sum of two and three. The mechanism is different.

In FORTH, everything is grist for the mill of the interpreter. Typing the character "2" tells the interpreter to place the value 2 on the parameter stack. Typing "3" places the value 3 on the stack. Typing the character "+" causes the system to add the two top values on the stack. "." causes the system to print the value currently on the stack to the console.

FORTH is an integer arithmetic language, although one can implement floating point. The obvious advantage of fixed point arithmetic is speed. Precision is another. With 32 bit fixed point operations,

one can exceed the precision of OSI's 9 digit BASIC and have far greater speed. With the assembler handy, one can readily code 64 bit precision operators if one really needs to express over 2.3E18 precise to the last bit. Another advantage is that one can easily and simply do operations which are extremely complex in floating point. The operator /MOD does a 16 bit division, returning both a remainder and a quotient. In BASIC, you would need to code a subroutine to get the same results.

There is a lot going on here which I have skipped. All of this is potentially useful. There is a very useful system variable called BASE. All internal operations are in binary. The value in BASE is the base to which the interpreter refers when converting values for the console. Thus, the following are two identical ways to place the same value on the stack:

HEX 20

or

DECIMAL 32

If you made both of these entries, you could then type "... " and you would see the two top values on the stack:

## HAS YOUR HARD DISK GONE S-O-F-F-T?

**BTI is your Authorized Service Agent for:  
Okidata, OSI and DTO 14-inch disk drives.**

**BTI service includes:**

- Maintenance contracts
- On-site service
- Product exchange
- Depot repair

Over 15 years' computer systems maintenance experience.  
More than 5000 disk drives currently supported in the field.

For information or service, contact:

**U.S. and Canada**

Greg De Bord  
Sunnyvale, California  
408-733-1122

**Europe**

Victor Whitehead  
Birmingham, England  
021-449-8000



**COMPUTER SYSTEMS**

870 W. Maude Avenue, Box 3428, Sunnyvale, CA 94088-3428 (408) 733-1122  
Regional offices in Minneapolis, MN; Ramsey, NJ; Atlanta, GA; Dayton, OH

32 and 32 (remember we left the variable BASE set to 10).

Memory access in FORTH is fairly simple. Place a value on the stack (which is always sixteen bits wide), and then say @ (which is pronounced "fetch"). The sixteen bit value in that location and the one next to it in memory will be placed on the stack. If you have a byte addressed machine (any OSI machine), then the operation C@ (pronounced "C-fetch") will get the value in that address only. For example, if you wished to know how much memory 65D found at boot time (assuming you had a 65D based FORTH, an assumption I shall make for the rest of the article), you could type,

HEX 2300 C@ .

on a 48K machine. This would return the value BF, which is the most significant byte of the value BFFF, the highest address of the user RAM. Now, if you should wish to change that value, you could type,

B0 2300 C!

which will place the value B0 in the location 2300. This operator for store is called "C-store".

The FORTH operating system is fairly simple to grasp at first. The guts of it is a procedure called BLOCK. The bulk storage (tape, disk, whatever) is organized in 1K chunks, called blocks. Thus, to enter 5 BLOCK is to cause the system to get the sixth block from the disk, and bring it into memory, if it isn't already there. BLOCK is somewhat intelligent, as it provides two services automatically. One service is that it will select the disk buffer which was referenced the longest time ago, and use it for the new block. The other is that, if that buffer was flagged as having been updated, its contents are first written out to disk. In addition, BLOCK leaves on the stack a very useful datum, the address of the buffer in which the block now resides.

I have now explained enough (just barely) to describe the beginnings of virtual memory operations in FORTH. If I have a block, say block 15, which I am using to store data from an experiment, I might wish to access the 234th byte in that block. I can type,

15 BLOCK 234 + C@ .

15 BLOCK forces the block into memory, if it isn't already there, and leaves the starting address of the block buffer on the stack. 234 is the offset into the block, and + adds the two values of the stack to produce the address in memory of the byte desired.

I mentioned that FORTH operates in 16 bit chunks. This is the default data size, but there are operations designed for 32, and even 64 bit operations. For example, \* ("star") is the 16 bit multiplication operation, and D+ ("D-plus") will add two 32 bit values. In both cases, a value of the same precision will be returned.

There are also mixed precision operators. If you want to multiply two 16 bit values to produce a 32 bit value, you can use M\* ("M-star"), which would be very handy to convert a 16 bit value to a 32 bit one. Simply type,

1 M\*

similarly, to convert from a double precision value to a single precision value, type,

1 M/

/ ("slash") being, of course, the single precision division operator.

Now, suppose in our experiment you wished to work with a lot of data, more than you could save in one block. You could allocate several blocks and still access each byte individually. To continue our example, if you wished to access the 4567th byte from the beginning of block 15, you could type,

4567 1024 /MOD 15 +  
BLOCK + C@ .

here, 4567 if first divided by the number of bytes in a block, 1024. The quotient and remainder are returned to the stack. Then 15 is added to the quotient, and that block is called from the disk. The address of the block is now left on the stack, and the remainder from the /MOD operation is added to it, giving us the address in memory of the desired byte. The C@ and . get the byte and cause it to be printed.

The above example would suggest that file handling is a normal part of FORTH. In fact, it isn't. FORTH has a philosophy which may be summed up as, Keep It Simple, Stupid

(KISS). FORTH allows you to do all sorts of neat things, but there are enough different ways to go on a particular type of thing that FORTH lets you decide how to do it. Directories, data structures, data bases, and so on, all should be designed to a particular job. So you get to implement them yourself. As I have suggested in my example, they aren't hard to do. So great variety will proliferate.

This is part of a more fundamental philosophical point of FORTH. FORTH is a language for people who would like to (or need to) muck with the guts of the system. I usually write printer drivers in assembler whenever I need one. FORTH is the only language besides assembler in which I would even consider writing one, and I have done so. This does not mean that you have to get down to the guts of a system in order to write your application, only that the capability is there if you wish to do so.

All of the examples that I have given have been immediate entry of commands. FORTH provides an interpreter to execute procedures previously defined. When you boot the FORTH system, enough of FORTH is pre-defined to allow you to go on to load your application. Loading consists of compiling new procedures. FORTH doesn't care whether its source code comes from the disk or the console.

Let us take a simple example of expanding FORTH. Suppose you have a lot of numbers in hexadecimal and you want them in decimal. You could type,

HEX 234 DECIMAL .

Or you could first use the compiler. For example,

: H>D DECIMAL . HEX ;

the colon is the word that calls forth the compiler. Hence, a new entry to the dictionary may be called a colon definition. H>D is the name of the new word. H>D is a simple, descriptive name for the word. This word assumes that the value already on the stack is the value in which we are interested. First it sets the base to ten. Then it prints the value at the console. Since the idea was to convert a lot of numbers, the next step is to set the base to hex, in preparation for the next value.

Once this is done, we can use the new word just as we can use any other word in the dictionary:

234 H>D

which will give the answer desired: 564.

FORTH's compiler produces a series of linked lists. Each list is a colon (or other) definition. A header contains the name of the procedure, so that the interpreter can find it for later execution, and the compiler can find it to incorporate into higher code. It also contains a link pointer to the next word down in the dictionary. The next entry is a pointer to the code required to interpret the rest of the word. In a colon definition, this field points to the run time code in the word : . The rest of the word consists of pointers to the code to be executed. In our example of H>D , the parameter field points in sequence to DECIMAL , , and HEX. Then a pointer is compiled to a point to ; , which ends the execution of the word at hand.

The code thus compiled is now available to use just like any of the pre-defined code in the kernel. In fact, FORTH doesn't know the difference. Most systems have a kernel in pre-compiled object form on the disk which is brought into memory at boot time. Then, different additional code is compiled by the user depending on the application. This points out one of the unusual features of FORTH: the compiler is so fast that recompilation of source code is faster than a relocatable object code module loader would be.

One application which every programmer uses is the FORTH editor. In some versions of FORTH, this is a simple TTY editor, line oriented, but with fairly sophisticated search and delete functions. I wrote a full screen editor because I wanted one, and indeed most FORTH programmers customize the editor to suit their own predilections.

FORTH usually comes with its own assembler. This is so CPU-specific that no standard can define what should be in a FORTH assembler. Most FORTH assemblers are table driven. They compile object code directly into the dictionary, just as the FORTH compiler compiles FORTH code. Since assembly code goes directly

into the dictionary, there is no need for a link editor or a linker, or for elaborate pointers and fiddling with memory. In addition, once a code definition is in place, it can be called at any time just as any other FORTH word.

The practical result of this is that if the programmer wishes to code in assembler, he need only compile the assembler, and LOAD his code from disk. However, since FORTH is so fast, it is customary for FORTH applications to be written in high level first, and then if the speed of assembler is still desired, to code the slowest functions in assembler.

FORTH's assemblers are structured assemblers. One can code BEGIN ... END structures, IF ... ELSE ... THEN , whatever one wants. This is because the assembler is itself extensible. Since it is written in FORTH, you can add to it yourself. I have added several items to the "Blue Sky Products" assembler, and they have made life much easier for me. (Incidentally, this ought to indicate what a bargain a good FORTH package is. How often do you get a structured macro assembler for \$75, never mind its source code and a high level language thrown in?)

Interrupt handling is usually fairly simple under FORTH. One writes a normal code definition, except for two features. One is that an interrupt handler is headless. This is because it is never executed by the interpreter, or compiled into higher level definitions. The other is that it begins with the word BEGIN , which is nothing more than a word to place the address of the start of the routine on the stack. It ends with the word INTERRUPT, which is a CPU-specific word to compile the start address into the appropriate interrupt vector, and to put a RTI instruction at the end of the interrupt code.

I said that interrupt handling is usually very simple under FORTH. One exception to this rule is that the OSI computers, except for the C3, are hard coded to have their interrupt vectors in 01 page. The problem with this is that CPU stack (FORTH's return stack) is hard coded to live in 01 page also. PUTTING A 6502'S INTERRUPT VECTORS IN 01 PAGE IS ONE OF THE DUMBEST THINGS DONE TO A COMPUTER

SINCE BABBAGE!!!! And OSI did it!

However, there are ways to get around this problem, and I am plodding along on a fix for it for Blue Sky Products' fig-FORTH. For the moment, you have to see to it that your return stack doesn't go deep enough to write over your interrupt vector at \$01C0 (IRQ). Of course, you can't use the NMI because the DOS is entirely in software.

These problems aside, fig-FORTH is a very good language for a number of real time applications. I have written a OS-65D directory handler in FORTH which is much faster than OSI's in BASIC, a colour graphics demo which for sheer speed beats OSI's 65D 3.1 colour demo disks. I have also implemented full cursor addressing on the video console, which is fast enough to allow a full screen editor on the Blue Sky Products' fig-FORTH. I also have, as I mentioned, written a NEC Spinwriter driver for the Diablo interface, all in high level FORTH.

I have not covered the language in its entirety here. That could be the subject of several books. One such book I will suggest to you here is Starting FORTH, by Mr. Leo Brodie of FORTH, Inc. This book is published by Prentice-Hall, and copies may be ordered through your local book seller.

\* We tried to call 'Blue Sky Products,' 729 East Willow, Signal Hill, CA 90806, and were unsuccessful. They either have an unlisted number,

## DISK DRIVE RECONDITIONING WINCHESTER DRIVES

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

Shugart	SA4008	23meg	\$550.00
Shugart	SA1004	10meg	\$450.00
Seagate	ST412	10meg	\$350.00

### FLOPPY DRIVE FLAT RATES

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

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

Phone: (417) 485-2501

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

moved out of the area, or are no longer in business.

PEEK Staff.



#### AN OSI CASSETTE BASED 6500 ASSEMBLER/EDITOR BUG & FIX

By: David W. Adams  
407 Rollcrest Ct.  
Midland, MI 48640

OSI's cassette based 6500 Assembler has a little known Bug.

The Bug occurs when the Assembler is expected to assign the Low-Byte value of a label's 2-Byte machine code address to a Register via the Immediate addressing mode (Example 1).

```
2; Example 1
5      *=$7000
10     LDA #MESS ;Get LO-
          Byte of MESS Address
20 MESS  .BYTE 'MESSAGE'
```

Line #10 should work and does work on OSI's disk based Assemblers, but refuses to work on the cassette based Assemblers as it will report 2 error conditions; Error #20 in line #10 and Error #12 in line #20 in program Example 1.

Take note that in Example 1 if we changed line #5 to:

```
5      *=$0000
```

the Bug will not occur as the "MESS" label's address will equate to a 1-Byte machine code address, the Bug will only appear when the label's address equates to a 2-Byte machine code address (Example 1).

Illustrated below is a quick and dirty way to get the cassette Assembler to assign the Low-Byte value of a label's 2-Byte machine code address to a Register via the Immediate addressing mode (Example 2).

```
2; Example 2
5      *=$7000
10     LDA #MESS*256/256 ;
          Get LO-Byte of MESS
          Address
20 MESS  .BYTE 'MESSAGE'
```

The Fix in Line #10 (Example 2) works fine, but at the cost of extra confusion and bytes in your Assembler source text.

In comparing the Assembler source code of the disk based vs the cassette based Assemblers I have found that the

Assemblers are approximately 99% the same. The exception being that the disk based Assembler has a few table code value changes, as well as the location of the "Fill the line buffer" routine, now at the start of the Assembler's machine code and some additional disk I/O routines at the end of the Assembler's machine code.

After several hours of evaluation, I found the corrupt machine code in the cassette based Assembler, the following address locations code in the cassette Assembler must be changed as follows:

Location	Present Code	New Code
(Hex)		
08ED	\$09	\$14
08EE	\$6A	\$4C
08EF	\$04	\$1F
08FO	\$41	\$05

To verify that the code on the disk based Assembler was correct for the cassette based Assembler, I plugged the 4-Bytes from the disk Assembler into the cassette Assembler (same memory locations).

Voila, it eliminated the Bug from the cassette based Assembler!

Now to verify that the cassette based Assembler code that was suspect was really at fault, I plugged the suspect (corrupt) code values from the original cassette Assembler into the disk Assembler. Now the Bug which had occurred in the original cassette Assembler was now plaguing the modified disk Assembler.

I can confidently conclude that this Fix is tested and a permanent solution to the elimination of the Bug in the cassette based Assembler/Editor.



#### FILE HANDLING UNDER OS65-U & OS-DMS

By: Raymond D. Roberts  
5996 Longdin Rd.  
Ferndale, WA 98248

Every time a file is created with OS65-U or OS-DMS, a file header is created. This header contains information essential for file handling by the computer or programmer. While the header is transparent, it is physically (and always)

laid out on the disk in the following manner.

NAME#TYPE#EODF#BODF#REC  
LENGTH#NUMBER OF RECORDS#

The first 52 characters of the file are reserved for this information, allocated as follows:

NAME-5 (0-4)CR (carriage return) 1 (5), TYPE-2 (6,7) CR-1 (8), EODF-10 (9-18) (end of data file), CR-1(19), BODF-10 (20-29) (beginning of data file), CR-1 (30), RECORD LENGTH-10 (31-40), CR-1 (41), NUMBER OF RECORDS -10 (42-51), CR-1 (52).

Immediately following this is Field names-Field parameters, (length of field holding data, not length of field name).

Using for an example, the first three field names in "ADS", we would see the following:

CLASSIFICATION#4#ITEM#21#DESC-1#21#DE---  
53 68 75 85 88

NOTICE!! The 4 represents 3 spaces for data and 1 CR, the 21 means 20 data and 1 CR.

The first character of the fourth field name (DESC-2) would start at character (byte?) 88 on the disk data file ADS. This would continue on until all field names and field parameters are entered. The computer then computes the number of characters (bytes) used, and the beginning of data file (BODF) is the next physical character on the disk. In the file "ADS" (see PEEK May '85 listing), you will note that BODF is at 171. EODF (end of data file) will naturally be 171 plus (240 \* NR), 240 being length of a record in ADS. NOTE!! If you have 950 available records in the file, but only the first 30 have data in them, then the EODF is at the end of the 30th record.

Next, we need to visualize two things. 1. The absolute address of data in the file (constant). 2. The offset address of data in the file (relative). Also, we need a name for the pointer that locates (points to) a specific address, (location). Let's call this pointer an INDEX (because that's what OSI calls it).

Let's look at record #3 of our ADS data file. If we set the INDEX at 651, we find that this is the first CHR in record three.

171-411 411-651 651-891 or  
 171 + 240 + 240 = 651 or  
 BODF R#2 R#3 INDEX

Our absolute address is 651.

Our offset address is 0.

Since field #1 is 3 CHR's long, the absolute address of field #2 would be 655 (651,2,3 for data 654 for CR), its offset address would be 4. Field #2 being 30 CHR's long, the absolute address of field #3 would be 686 (655+30+CR). The offset address from BE (beginning of record) would be 35 (3+CR+30+CR).

Visualizing this, we can see that any particular field will have an absolute address from BODF, an offset address from BE or a relative address from its last occurrence or its next occurrence.

Now, let's see how we can use this in opening, loading, or handling data files. Let's use the data file ADS, and the program ADS100.

First, let's open and load the ADS data file.

In LINE 170 we set the INDEX to location 0 and input the name.

In LINE 190 we set the INDEX to 6 and input (load) the TYPE. We can have 10 for Master or 20 for Key.

In LINE 210 we load EODF.

LINE 220 sets the INDEX at 20 and inputs the BODF which in this case is 171. LINE 250 inputs the record length (240) at location 31, and LINE 260 loads the number of records (950) at location 42.

If you could see this as it actually appears, it would look like this: (using OS-65U "FDUMP", you can)

```
ADS #10#17211#-----171#-----  

-240#-----950#-----CLASSIF  

ICATION#4#ITEM#31#DESC-1#31#DE  

SC-2#31#DESC-3#31#DESC-4#31#PR  

ICE#21#PHONE#31#CITY#16#STATE#  

4#EXP DATE#6#CODES#3#133#
```

followed by the first CHR in the first record. The -- are nulls, the #'s are carriage returns. Now we dimension for 20 field "contents", (L\$), 20 field parameters "lengths", (FP), and 20 field labels (A\$).

LINE 300 sets the INDEX to 53 (BODF), then sets a field label counter to 1 (N=1) and a field parameter counter to 1

(NF=1). Total record length is set at 0 (TT=0). Ignore TF=1. LINE 305 inputs the first field table and length.

T\$ and T are "dummy" variables. LINE 330 increments field name and length on each pass. LINE 340 keeps a running total of field lengths so that after 12 passes, (12 fields in ADS) the total record length will be 240. LINE 350 says go back to LINE 305 and input the next field name and length. Our INDEX <1> is now at address 171.

If we are using two files, LINES 360-397 would input (load) the second file header in the same manner as the first one was done.

Now that the file(s) headers are loaded into memory, let's assume that we wish to find a particular record. Assume further that we have a record whose classification is 120. LINE 505 asks what classification we wish to find (or what classification number). Here is where we will use the FIND command. If we entered "120" then the program fails thru to LINE 720: GOSUB30005.

Remember that our INDEX is now at 171 (BODF). CX\$ was 120 not "ALL", so we GOTO 30005, which says FIND 120 (CX\$).

The computer will now sequentially search our data fields for the first occurrence of 120. LINE 30010 says if we reach the end of the data file without finding an occurrence of 120 then we are done, GOTO 33990.

LINE 33021 sets up variable TY to equal the number of the record in which 120 occurs if we find one. It states that TY will equal the integer of what address the INDEX (on channel 1) is pointing to, minus 171 (BODF) divided by 240 (RL). For example, TY=651 -171/240=2 (record #2).

LINE 33021 checks to make sure that we did not FIND a 120 in the middle of a record. We only want to find it if it is in the first field (classification number). TX (651) = 2\*240+171.

LINE 33035 says that if TX (651) does not equal the address that the INDEX is pointing to, then increment the INDEX by 1. This will continue until a 120 is found in the first field if it occurs (loops back to LINE 30005).

In this, remember that INDEX <1> is the absolute address that the pointer is pointing to (the only way to SET a pointer) and INDEX (1) is an abstract reference to INDEX <1>.

If we had wanted to FIND an occurrence of 120 in the third field of a record, we could have set the INDEX to BODF plus 35 (INDEX <1> = BODF+35). BODF (171) + 35 = 206. The pointer would be set at address 206, the first CHR in the third field. 53 and 171 are absolute addresses; 35 is an offset. 206 is another absolute address and 240 is an offset from 171 or 206.

Continued

FILE: ADS 0  
 NUMBER OF RECORDS: 950

CLASSIFICATION	---
ITEM	-----
DESC-1	-----
DESC-2	-----
DESC-3	-----
DESC-4	-----
PRICE	-----
PHONE	-----
CITY	-----
STATE	---
EXP DATE	----
CODES	--

RECORD: 1	INDEX: 171
CLASSIFICATION	133
ITEM	HAY-FEED
DESC-1	--- FAYNE FARMS ---
DESC-2	DAIRY QUALITY ALFALFA-
DESC-3	CALL FOR DELIVERED PRICES-
DESC-4	
PRICE	
PHONE	509-266-4619
CITY	PASCO
STATE	WA
EXP DATE	40310
CODES	

Of course, this would not work here because we set up in LINES 33021-33035 a bypass for those occurrences anywhere but in the first field. More detail on this next time when we get into data file "CLASS", LINES 360-8060.

I hope I haven't confused you.



SCRIBE  
65-U WORD PROCESSING  
IN BASIC

By: Richard E Reed  
Oak Creek Securities  
411 N. Mill St.  
Tehachapi, CA 93561  
(805) 822-7952

Oak Creek Securities, a wind power developer, uses a C3-B with dual-sided floppy drives and 4 Denver boards. Over the course of the past two years it was upgraded from a C2-D through a multi-user C3-B to the current configuration. During the first transition we purchased General Ledger, VMDBMS, and Scribe, all by IHS Computer Services, through Space-Com in Laguna Hills, both advertisers in Peek (65). After encountering several problems with the software as delivered, we were informed by IHS that the disks we received from Space-Com were a demo set and not the full blown system, that some of the programs were currently being re-written, and that IHS would furnish us a new set of disks. They did, and these too had annoying bugs here and there throughout the programs. This is to be expected of software that is in a state of flux when delivered on an emergency basis, and IHS helped us work through the difficulties. Our only complaint with the company is that there is seldom anyone there, so it is difficult to get a problem resolved.

Space-Com has been responsible for all of our hardware support and upgrading. We have found them to be knowledgeable, reliable, and eager to satisfy their customers. Our machine was purchased used, and was originally sold as a real estate package through RealStar of Colorado. It had annoying hardware bugs that persisted through the upgrades. Space-Com went out of their way to help us track down and eliminate the problems, and we now have a system that operates error free. Because we have frequent power outages, and because we often print mailings all night long, we recently installed a Topaz uninterruptible power supply, something we now highly recommend.

Our primary use of SCRIBE is for mass mailings using the Mail Merge

feature with our data base manager. But it is also used for generating offerings and sales literature, and for manuscript preparation of books and screen plays. It is a versatile package and has given us good service. One small annoyance is that our typists can often get ahead of its ability to capture key strokes, and we experience some errors based on that problem.

The fact that SCRIBE is written in Basic has made it easy to modify for our own special needs here. That has made an already versatile and functional set of programs even more valuable. We have made 9 major modifications which I will describe.

1. As delivered, except when loading a file, the only way to clear the work space was to exit out of Scribe. We added a clear function so that one could type a letter, save it, and then proceed to type another.

2. The programs, as delivered had no way to join files smoothly, either in printing or in the edit mode. We disabled the memory clear function on the file-load command. This permits files to be continuously loaded until one runs out of user memory. Thus many small files of standard paragraphs can be quickly concatenated to produce larger documents. This change was one of the reasons for change # one, since we often wish to load new files without adding them to the end of our current text.

3. Since we produce books and screen plays on our computer, we also needed the capability of printing multiple files without having to worry about page breaks and pagination from one file to the next. To accommodate this need we added the capability of printing any number of files one after the other without disturbing the page/line count or page numbers and footers. We can now print a whole book as one smooth continuous process.

4. Sometimes it is nice to send out mail-merged letters in justified format. But because of justification calculations it is a slow process, since each replication of the letter performs all the calculations. To speed things up we created a facility to save the letter in its justified format except for the mail-merge functions. You then run the letter as a flush left product. The outcome is in justified type with the merged features at over 4 times the speed.

5-7. We also wanted to be able to change the parameters for margins, tabs, and justification during the body of the text. We found that adding these facilities was trivial since the source was in Basic, and now we have full control of these functions, turning justification on and off, moving margins all across the page, and re-positioning tabs as

needed in the body of long texts. The programming of this word processor is quite straight-forward, and we had little trouble locating the appropriate routines, understanding the code, and making the modifications.

8. The programs had a facility to stop between pages by putting in a halt command to accommodate single sheet feeding. However, in printing long texts there was no way, except by trial and error, to tell where a page would divide. To alleviate this problem we expanded the halt command to require it to be given only once in order to permit stopping whenever the computer decided to start a new page.

9. Recently we acquired an Epson LQ-1500 printer, which supports proportional spacing. We decided to add justified, proportionally-spaced type to the output options. In the process we now have truly centered text,

\*  
AA  
AAA  
AAAA  
AAAAAA  
H  
WWW

ragged left,

\*  
AA  
AAA  
AAAA  
AAAAAA  
H  
WW

as well as ragged right,

\*  
AA  
AAA  
AAAA  
AAAAAA  
H  
WW

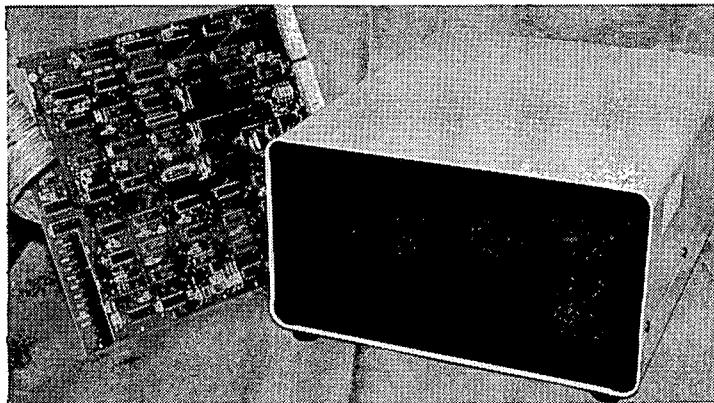
and justified output that looks much better than spacing-between-words print. This article was set proportionally. The look-up tables can be modified for any printer with proportional modes, and multiple fonts are accommodated.

There are other features already in SCRIBE that make it a good choice for mail-merge applications with DMS structured files. You can pause during the printing function to enter data from the keyboard, but more importantly, that can be the method

# **SUPER HARD DISK Subsystem!**

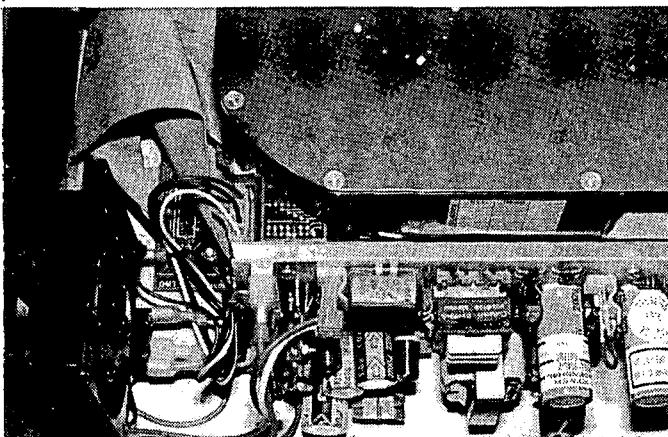
**TURNS ANY FLOPPY BASED COMPUTER INTO HARD DISK BASED, INSTANTLY.**

- 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 5 $\frac{1}{4}$ " 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**

14661A Myford Road, Tustin, CA 92680 (714) 731-6502

of putting the same data into the DMS files as well.

We highly recommend SCRIBE as a versatile word processing package for 65-U, especially for those users who may find it necessary to change virtually any or all of the functions and features to suit their peculiar needs.

**EDITOR'S FOOTNOTE:** The above changes have been made to SCRIBE and this updated version is currently available for an additional \$50.00.



### SILENCE YOUR DISK DRIVE

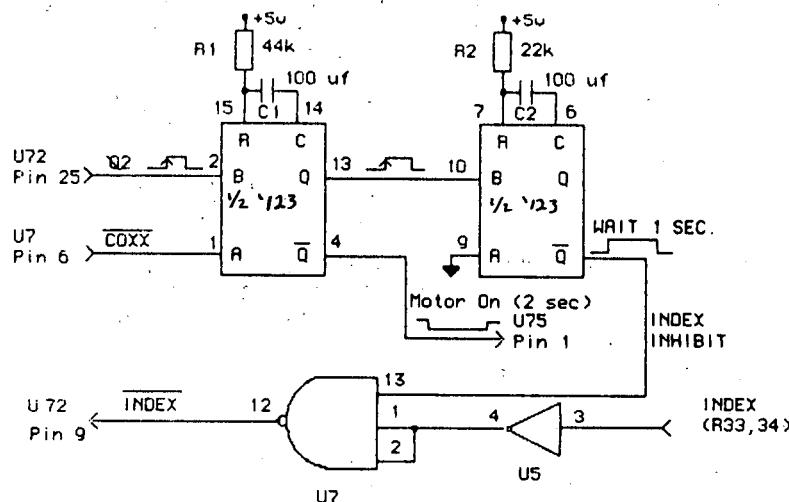
By: David Pompea  
319 Hampton Blvd.  
Rochester, NY 14612

As I sit here the red light on the front of my drive is on, but the disk is quiet and the head is not loaded. Your disk can be resting too.

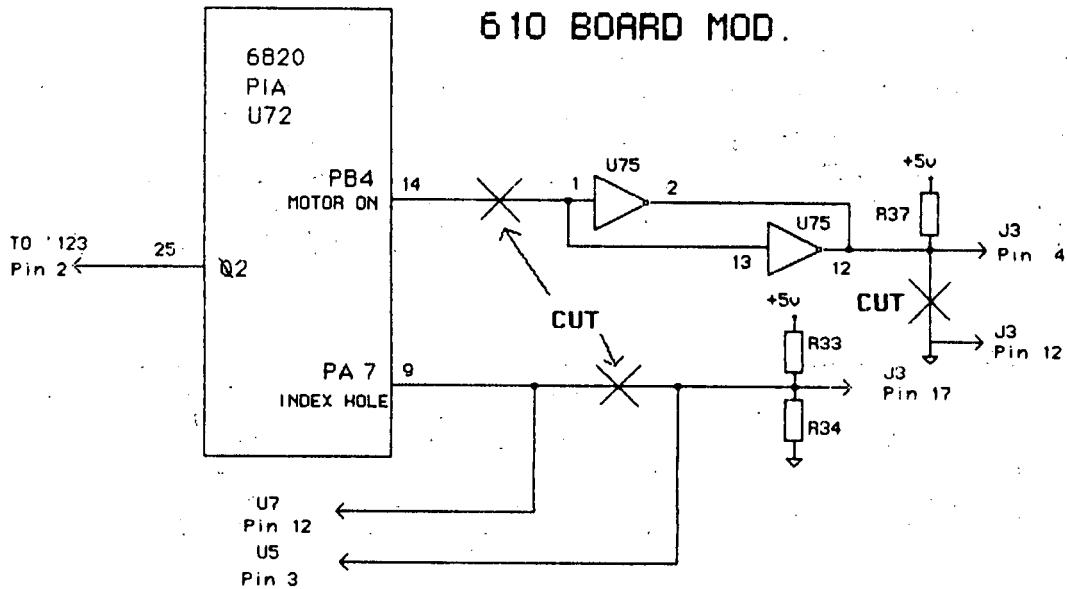
The normal way DOS reads data goes like this: load the r/w head; seek the track; wait for the index hole; wait for the correct sector; and then read the data. All these operations involve accessing the PIA and ACIA at the address

C0xx and using the phase 2 clock line to tell us that the address is valid.

Here is how my modification works: When the phase 2 line goes high and the not C0xx line is low, a one shot is triggered (and can be retriggered) and starts timing. During its timing, the motor on line is activated to get (and keep) the disk spinning. The leading edge of the motor on pulse is used to trigger the other one shot. This one blocks the index hole pulse to the computer for one second to allow the disk to get up to speed before reading or



### 610 BOARD MOD.



# C1P/MF DRIVE FIX

©

1985 DAVE POMPEA

writing. The entire circuit is on a 2 x 2 perf board, wrapped in tape and hidden inside the computer under the 620 board. I've used several "spare" gates on the board, so if you don't have them or are modifying anything other than a CLP/MF, you'll need to mount them on the perf board.

#### MODIFYING YOUR CLP/MF

1. On a small perf board, mount and wire the 74123, R1&2, C1&2.

2. Open your CLP to expose the bottom of the 610 board.

3. Using wire-wrap wire (or other small ga. wire) connect the spare 3-input nand gate (U7) pins 1&2 to the spare inverter (U5) pin 4.

4. Pull off the disk interface board (the one with the ribbon cable) and look at position 4. That is the motor on signal pin. It's jumpered to ground at position 12. Cut that jumper.

5. Cut the trace from U72 pin 9 to R33&34 (bias resistors).

6. On top of the 610 board, cut the trace from U72, pin 14.

7. Connect pin 2 on the 74123 to U72, pin 25 (the phase 2 line).

8. Connect pin 1 on the 74123 to U7, pin 6 (the not C0xx line).

9. Connect pin 4 on the 74123 to U75, pin 1 (motor on line).

10. Connect pin 12 on the 74123 to U7, pin 13 (index inhibit).

11. Connect U7, pin 12 to U72, pin 9 (index to PIA).

12. Connect U5, pin 3 to the index signal bias resistors, R33&34 (the other side of the U72, pin 9 trace cut).

13. Connect power (+5V) and ground to the 74123 and add a luf bypass cap. on the perf board for noise filtering.

That's it! When DOS wants to use the drive, the motor will come on and after a one second delay (to let the motor come up to speed) the index hole signal will be let through. The motor will turn off after 2 seconds of non-use.

You can also change the jumpers inside the drive to load the head upon motor on

(it's already set for head load on drive select) by removing the pin 1-14 shunt on the programming socket inside the drive and inserting a jumper for pins 7-8.

Mods for the 4P coming soon.



#### SCREEN DUMP FOR DISK BASED SYSTEMS

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

OS65D 3.3 provides a screen-dump, so why another one? Well, this program dumps 30 lines in 14 seconds, as opposed to 104 seconds, (to device 4). This is at 2MHz. With a 1 MHz CPU clock, 208 seconds is an absurdly long time to wait for a screen dump.

The program does not print a box round the printout, but if you want everything squashed together type this first:

```
PRINT#4, CHR$(27);CHR$(49);
For printers at DV#1 ($FC00) try PRINT = $24CD.
For DV#8 ($CF00) try PRINT = $24BD.
```

```

10 ; SCREEN DUMP FOR OS65D 3.2 & 3.3
20 ; WARNING! SELF-MODIFYING CODE!
30 ;
40 F425 * = $F425
50 249F= PRINT = $249F ; for DV #4
60 ;
70 F425 48 PHA
80 F426 8A TXA
90 F427 48 PHA
100 F428 98 TYA
110 F429 48 PHA
120 F42A A200 LDX #$00
130 F42C A000 LDY #$00
140 ;
150 F42E BD00D0 COUNT LDA $D000,X
160 F431 209F24 JSR PRINT
170 F434 C8 INY
180 F435 C040 CPY #$40
190 F437 D00C BNE NEXT
200 ;
210 F439 A90A LDA #$0A
220 F43B 209F24 JSR PRINT
230 F43E A90D LDA #$0D
240 F440 209F24 JSR PRINT
250 F443 A000 LDY #$00
260 ;
270 F445 E8 NEXT INX
280 F446 D0E6 BNE COUNT
290 F448 EE30F4 INC COUNT+2
300 F44B AD30F4 LDA COUNT+2
310 F44E C9D7 CMP #$D7
320 F450 D0DC BNE COUNT
330 ;
340 F452 68 PLA
350 F453 A8 TAY
360 F454 68 PLA
370 F455 AA TAX
380 F456 68 PLA
390 F457 A9D0 LDA #$D0
400 F459 8D30F4 STA COUNT+2
410 F45C 60 RTS

```



# computer repair

#### Board level service on:

- OSI / Isotron
- TeleVideo
- IBM pc/xt

#### Floppy drive alignment:

- Siemens
- Shugart
- Teac

#### Terminal repair:

- TeleVideo
- Micro-Term

(1 week turnaround)

Sokol Electronics Inc.  
474 N. Potomac St.  
Hagerstown, Md. 21740  
(301) 791-2562



**Sei**

## TOKEN LOAD/SAVE PROGRAM

This program allows you to make copies of your M/Code programs, especially those that you have modified, even if you don't have the extended monitor in EPROM.

By: John Whitehead  
17 Frudal Crescent  
Knoxfield 318, Australia

At 300 baud, it will load three times faster than OSI Checksum. It can save up to four blocks of code of any

length and auto start the program.

This was originally written by Alan Cashin. I have modified it so that it can put directions on the tape being made. Also, to allow one line of POKEs which can be used to restrict BASIC to below the M/Code program. It allows a M/Code program to be moved to higher memory, e.g., \$2300. So that this BASIC program can be loaded in at \$0300, an offset of \$2000 is then entered in, so that when the tape just

made is loaded, the M/Code is loaded back in at \$0300.

To allow return to BASIC after loading the tape just made, answer \$0000 to the "GO AT" address. The PRINTCHR\$(217) is a screen clear command.

An alternative method is to relocate the BASIC program above the M/Code, the one to be copied, as described in my article in last month's PEEK. If you want to save BASIC programs in token format, relocating this BASIC program is the only way.

```

30 REMARKABLE TOKEN LOAD BY ALAN CASHIN. MODS BY JOHN WHITEHEAD
32 POKE13,20:POKE517,0:DIMD(3,3):DIMOF(3):AC=61440:FF=65535
34 PRINTCHR$(127)"TOKEN CHECKSUM LOAD/SAVE
36 PRINT:PRINT"CAN SAVE UP TO 4 AREAS OF MEMORY WITH
38 PRINT" OFFSETS OF $0000, $2000, $3000 OR $4000
40 PRINT:PRINT"THE 1ST ONE CAN BE BASIC STARTING AT $0300
42 PRINT" ANSWER B TO START ADDRESS FOR BASIC.
44 GOTO110
46 WAITAC,2:POKEAC+1,E:RETURN
48 FORA=$TO25000:NEXT
50 FORA=$TO100:PRINTCHR$(0)::NEXT:PRINT:RETURN
52 DATAORIGINAL START ADDRESS , "ORIGINAL END ADDRESS "
54 DATA"GO AT OR MORE "
56 DATA.0013/95*5B*2C*A2*C8*AD*00*F0*4A*90*FA*AD*01*F0*E8*30*EF*.0016G
58 DATAA206814F18755395539002F654A901CACAD0F1A2FC202901E830FAE004F00B
60 DATAB557F0F5E00290CD6CFCCF85FBA2F74C18002901E9FFC1F660130655265690
100 DATA02E6552A90F5C93A900269069DA9CF6018002901FFFF57FFA9X
110 PRINT:INPUT:ENTER PROGRAM NAME ";PS
120 PRINT"INPUT 4 SINGLE LINE MESSAGES, A SPACE = NONE
130 INPUTCS$,DS$,ES$,FS
140 PRINT"ENTER A LINE OF BASIC COMMANDS BEGINNING WITH A ";CHR$(34)
150 INPUTGS
160 PRINT:PRINT"SELECT OFFSET OF $0000, $2000, $3000 OR $4000":PRINT
170 RESTORE:PRINT"AREA"N". ENTER OFFSET ":"INPUTOF
180 IFOF<>0ANDOF<>2000ANDOF<>3000ANDOF<>4000THEN160
190 OF(N)=OF*4096/1000
200 FORA=1TO3:READBS
210 IFX$="B"THENX$="X":NEXTA
220 PRINT"AREA"N". ;BS::INPUTXS:IFLEN(X$)=4THEN270
230 IFX$="M"ANDN=3THENPRINT:PRINT"LAST AREA.":GOTO220
240 IFX$="M"ANDA=3THENE=24:GOTO320
250 IFX$="B"ANDA=1ANDOF(0)=0ANDN=0THEN560
260 PRINT"NOT 4 HEX":GOTO220
270 E=0:FORC=1TO4:F=ASC(MIDS(X$,C,1))
280 IFF>47ANDF<58THENF=F-48:GOTO310
290 IFF>64ANDF<71THENF=F-55:GOTO310
300 GOTO260
310 E=E*16+F:NEXTC
320 D(N,A)=E:NEXTA:IFD(N,1)>D(N,2)THENPRINT"START > END":GOTO170
330 PRINT:PRINT"WAIT FOR THE NEXT INPUT MESSAGE":PRINT
340 C=0:FORA=D(N,1)TOD(N,2):C=C-PEEK(A+OF(N)):IFC<0THENC=C+FF+1
350 NEXTA:D(N,0)=D(N,3):D(N,3)=C:D(N,2)=FF-D(N,2)+D(N,1)
360 IFD(N,0)=24THENN=N+1:GOTO170
370 INPUT" READY TO RECORD ";Z$:PRINT
380 PRINT" START TAPE NOW":SAVE:GOSUB48
390 PRINT" 20 PRINTCHR$(127)"CHR$(34);PS
400 PRINT" 21 ";GS
410 PRINT" 22 PRINT"CHR$(34)"TOKEN CHECKSUM LOAD"CHR$(34)"
420 PRINT" 23 PRINT:PRINT:PRINT:PRINT"CHR$(34)CS
430 PRINT" 24 PRINT"CHR$(34)DS
440 PRINT" 25 PRINT"CHR$(34)E$
450 PRINT" 26 PRINT"CHR$(34)FS
460 PRINT" 27 IFPEEK(251)=0THENPOKE251,2
470 PRINT" 28 POKE123,"PEEK(123)":POKE124,"PEEK(124)
480 PRINT" 29 POKE11,67:POKE12,254:X=USR(X):END
490 PRINT"POKE515,0:RUN20";CHR$(13):GOSUB50
500 READBS:FORA=1TOLEN(B$):E=ASC(MIDS(B$,A,1)):IFE=42THENE=13
510 GOSUB46:NEXTA:C=500:F=C
520 C=C+1:IFC>LEN(B$)THENREADB$:C=1

```

LISTING CONTINUED ON NEXT PAGE

```

530 E=F:F=ASC(MID$(B$,C,1))-48:IFF>9THENF=F-7:IFF>15GOTO600
540 IFE<17THEN=E*16+F:GOSUB46:F=500
550 GOTO520
560 D(0,1)=768
570 D(0,2)=PEEK(123)+(PEEK(124)*256)
580 IFPEEK(124)<>3THENPRINT"BASIC POINTERS INCORRECT":STOP
590 NEXTA
600 FORA=0TON:FORC=0TO3:F=INT(D(A,C)/256):E=D(A,C)-F*256:GOSUB46:E=F
610 GOSUB46:NEXTC:FORC=D(A,1)TOFF-D(A,2)+D(A,1):E=PEEK(C+OF(A))
620 GOSUB46:NEXTC,A:POKE517,0:PRINT"FINISHED SAVEING":NULL0:END
630 :
640 THIS PROGRAM IS STORED AT $0300 TO $1000. IT CAN BE RELOCATED.

```



#### BASIC SPEED AND LINE NUMBERS

Courtesy of TOSIE  
 Toronto Ohio Scientific Idea  
 Exchange  
 P. O. Box 29  
 Streetsville, Ont.  
 Canada L5M 2B7

Key in the two little programs that follow. Note that they are identical, except for one digit, yet their running speeds are vastly different.

A note in the May 83 issue of Compute! helped explain the difference, and once again stresses the close relationship of PET BASIC and the OSI ROM version.

When BASIC encounters a statement that changes control to another line, e.g. GOSUBXXXX

or GOTOXXXX, then it must somehow find that line of code. I have had the impression that BASIC began the search at the lowest line number, continuing line by line until the proper line was found. Thus authors have expounded the virtues of putting subroutines, especially the critical ones, at the top of the program. They were to be executed faster because of the lower line numbers.

This proves to be not always true. In the demonstration programs line 256 is found faster 255, even though they are the same number of lines from the beginning of the program.

It turns out that the BASIC routines do search from the

beginning if the high byte of the line number is the same or lower than the high byte of the line to be found.

However, if the high byte is greater, then the search continues from the present line, without backtracking to the beginning.

Calculating the numbers to be used to take advantage is simple. Divide the present line number by 256, then add one to the integer, and multiply by 256, e.g. from line 40, type ?40/256. The computer responds with 0.15625. The integer part is 0. Add one equals 1. 1\*256=256.

Once again:  
 Your current line number is 1250. Key in ?1250/256. The

*From Gander Software, Ltd.*

*The Ultimate Personal Planner*

## TIME & TASK PLANNER

30 DAY FREE TRIAL — IF NOT SATISFIED, FULL REFUND UPON RETURN

- "Daily Appointment Schedule"
- "Future Planning List" - sorted
- "To Do List" - by rank or date
- Work Sheets for all Aspects
- Year & Month Printed Calendar
- Transfers to Daily Schedule

A SIMPLE BUT POWERFUL TOOL FOR SUCCESS

Put the two most effective success techniques to work for you — every day of every year. Just five to ten minutes a day allows your mind and dreams to take charge of your life.

**Set Your Goals:** To reach a goal, you have to know where you are going. Just enter your goals or future appointments and let your computer remind you.

**Set Your Priorities:** Success depends upon doing first things first. Assign priorities (1-99) to your "To Do" list, let the computer keep them ranked by date or priority, and then get to work. When the time comes, the computer will help you transfer items to your choice of time on the daily Appointment Scheduler.

**Technicalities** - Appointment Scheduler: 18 time slots per day (you define) for 60 days. To Do List: 60 items ranked by date or priority. Future Planning: 60 long range items, date sorted; days to event or days overdue. Transfer to Scheduler: just tell it the date and time. Printed Calendars: Year on a page and one month box planning; any month, any year. System uses both Julian and Gregorian calendars to handle dates from 1910-2399 and produce day of the week. Screen and menu driven; DMS Keybase compatible files. Detailed 38 page manual. Simple installation; FD to Multi HD. Files for 5 users=5,400 appointments. Unlimited Warranty.

**HARDWARE:** 48K OSI, 8" floppy or hard disk, serial terminal system, OS-65U v. 1.3 or later.

**FEATURES:** package allows configuration to ANSI standard and almost all non-ANSI terminals, AND user specification of printer port.

**PRICE:** \$300.00 (User Manual, \$25.00, credited toward TTP purchase). Michigan residents add 4% sales tax.

**DEALERS:** Your inquiries are invited. This program should be on every 65U machine, including your own. At dealer prices, you could bundle this superior package as a sales incentive.

**GANDER SOFTWARE, Ltd.**

3223 Gross Road  
 "The Ponds"  
 Hastings, MI 49058  
 (616) 945-2821



"It Flies"

computer responds with 4.8828, thus the high byte is 4. We use at least 5 as the high byte, or a line number 5\*256 i.e. 1280 or greater.

Thus an opportunity presents

itself to speed up BASIC just by a proper choice of line numbers. Hopefully, this will add a little to our understanding of BASIC, and maybe help speed up a critical section of your program.

```
10 REM EXECUTES IN 19.8 SECONDS
15 REM AT 1 MHZ CLOCK
20 REM SIMULATE PROGRAM LINES
21 REM
22 REM
23 REM
24 REM
25 REM
26 REM
27 REM
28 REM
29 REM
30 REM
40 FORI=1TO6000:GOSUB255:NEXT
50 PRINT"DONE":STOP
255 RETURN
```

```
10 REM EXECUTES IN 14.2 SECONDS
15 REM AT 1 MHZ CLOCK
20 REM SIMULATE PROGRAM LINES
21 REM
22 REM
23 REM
24 REM
25 REM
26 REM
27 REM
28 REM
29 REM
30 REM
40 FORI=1TO6000:GOSUB256:NEXT
50 PRINT"DONE":STOP
256 RETURN
```



#### DEALER PROFILE

By: Dennis Shoulders,  
SoftTouch, Inc.  
2 Eagle Drive  
Dayton, OH 45431

In response to PEEK(65)'s request for articles, here is how we got started as an OSI dealer in 1981. While we were at Ohio State working on our engineering degrees, we happened to meet a frantic salesman from a local independent telephone supplier. He told us a salesman from the company had sold a telephone switchboard to a large hospital by overstating its capabilities. The customer was told the switchboard would accept numeric messages from push-button telephone and distribute them to selected offices. This capability would allow the hospital to eliminate their slow and expensive pipe vacuum mail system, similar to those used at the drive-in teller windows at most banks. However, the switchboard was not capable of such a function nor did the company have the expertise to develop it at the time.

We knew this problem was close to our chosen areas, so we told the salesman not to worry. Three days later, after burning the midnight oil and using a friend's OSI C-3, a calendar-clock card, and a UTI card, we wrote a 1000 line BASIC program that clumsily worked. The program alternately scanned the console port and UTI card, which was hooked to the switchboard, to buffer incoming messages. When

the message was terminated, either by a carriage return or star symbol from the telephone keypad, the message was sent out to 4 remote serial printers along with the date and time. The user defines the meaning of the numeric telephone messages.

The telephone people were amazed with our work! But, much work was left ahead. The CRT on the console port was used in the admitting and discharge office to send alpha messages to the remote printers, and a fast typist could outrun our scanning rate. We took 3 more months to rewrite the program in 6502 Assembly Language. There is no one alive who can out-type our scanning rate now!

We burned the code into EPROM and set it up to run 24 hours per day, 365 days per year. The system is still running today, with the only down time related to a failed memory chip. This inexpensive system was named "Telephone Command System" and has saved the hospital a fortune in maintenance and operating costs. By the way, we advertised this program in PEEK(65), in the free software listings. This adventure is just one of many in SoftTouch's three year corporate history in working with Ohio Scientific products. More articles will follow.

EDITOR'S FOOTNOTE: In our "Reader Survey" many of our subscribers expressed a wish for articles such as this. Please do send more!

## LETTERS

### Ed:

I have put together a new memory map of OS-65U from the three you have published (Feb '80, Mar '83, and Jun '83) and other sources. These sources include information in letters and articles in PEEK[65], OSI Tech Notes, other unofficial OSI publications, and my own delvings into various programs and U itself. I started with Wallis' format (Feb '80 PEEK [65]), expanded the description field, and added columns for range and source of information. My map currently has over 700 entries, is 132 columns long, and is printed on 8 1/2 by 11 paper with margins for binding.

I'd like to provide it to your readers, with an incentive to help me update it, because we all could use as complete a map as possible. I hope that everyone who reads it will add to it all of the locations they have found, correct any mistakes or omissions in my version, and send this information back to me. I'll incorporate all of the updates and release it again. So I'm offering either edition alone for \$3, or both editions for \$5.

Various and sundry notes from OS-65U:

### FLAG 13 CAUTION

When using FLAG 13 to write BASIC programs off to a file in ASCII and then back again, watch out! Sometimes long lines are truncated--you'll lose closing parens, last digits of GOTO statements, etc.

### FLAG 16 ERROR

BEXEC\* 6.9 line 1080 says that FLAG 16 (and by implication 15) allow or disallow colons, commas, and ampersands on input. But if you look at the flag area in the OS, you find only POKEs for colon and comma. No POKE for an ampersand.

### COPYFI BUG

There is a bug in the OSI COPYFI routine--the new fast one that uses all available memory as a buffer--version 2.xx. Try this. Load it, and start a copy from A to B. During the copy, open a disk drive door. You get a DEVICE E ERROR 1! Happens for both devices A and B.

Took a few minutes to find



this one. During normal disk I/O, you are supposed to set the appropriate device before calling USR. The I/O routine takes the byte at 9832 (device number) and puts it in 9889, the first byte of the disk I/O buffer. Then it does the I/O. But COPYFI sets up a more complex buffer at \$6000 for the I/O. I traced it and its hooks into the OS and could not find it taking the device number from \$6000 or \$6009 and putting it to 9889, but it does. 9832 is not touched at all. It remains the number of the program device. Anyway, if you error out to line 50000, COPYFI takes the device number from 9832 when it prepares the error message.

To fix, change the PEEK(9832) in line 50000 to PEEK(9889).

#### THE BINARY SEARCH ADVANTAGE

David Weigle had an excellent article on binary searching in the December 1984 issue, but I feel that maybe less experienced people might have missed the real speed of a binary search in a long sorted file. Let's take, for example, a file of 60,000 sorted records, and do a binary search. We assume that we don't find the record until the last look.

Divide 60,000 by two and discard the wrong half. This leaves 30,000. Divide by two and discard the wrong half again. Now you have 15,000. A third time leaves 7500, a fourth 3750, a fifth 1875, a sixth 938 (let's do it the hard way and keep the odd record). Continuing, 469 (move 7), 235 (8), 118 (9), 59 (10), 30 (11), 15 (12), 8 (13), 4 (14), 2 (15), and 1 (16).

Look at this: search 60,000 records and find in 16 or less moves!

The rule of thumb is this: How many bits does it take to represent the number of records? That's the maximum number of moves it will take to find your record in a binary search. 60,000 takes 16 bits to represent, so it will take a maximum of 16 moves to find a given record.

Compare this to a sequential search. In general, half the time the record you want will be in the second half of the file, so half the time you will have to search through at least 30,000 records. That's a few more than 16.

#### HELP!

I have programs which write parameters to a scratch file and then RUN another program without keeping common variables. The second program reads the parameters from the file and executes. Sometimes, when I have EXITed to the OS and done things (haven't pinned down exactly what) and then rerun the programs, the first file PRINTS to the scratch file with double carriage returns. Not for every PRINT statement, nor with any pattern I see. Needless to say, the second program doesn't get its parameters straight with all the extra CRs. It takes a reboot to clear this, but the reboot always works. At least one other OS-65U programmer has seen this. Any guesses, readers?

Recently, I tried POKE X,PEEK (Y). No error message, but nothing was poked. Does anyone know why?

How many Tech Notes did OSI issue? I have only seen numbers 1-27, although several PEEK[65] correspondents mention number 28.

In his memory map info (March '83 PEEK[65]), Roger Clegg cites TI 1000 and others as sources. What are these?

In the May '80 PEEK[65] someone asked about ERROR 17 under OS-65U and Al Peabody in essence said "Panic!" Then Jim Sanders in August '80 had a couple of hardware-oriented causes/fixes. Under newer OS-65U versions, is ERROR 17 still a problem? If perchance the file with the error is the only copy, is there a POKE to permit the INPUT or LOAD to continue? Actually, if you look at the 6850 ACIA status register and set the 16 and 1 bits you get FRAMING ERROR and NO CHARACTER READY TO READ, so I suppose the 17 comes directly from the port at \$C000.

Does anyone out there know why the 535 dynamic RAM board will not run 2 MHz or with the Z-80?

Does anyone in the Kalamazoo, Michigan or Ft. Wayne, Indiana area have a maintenance manual for a Hazeltine 1420 that I could borrow?

Tom McGourin  
216 West Michigan Avenue  
Kalamazoo, MI 49007  
(616) 388-5955  
Fort Wayne, IN:  
(219) 429-4160 (Office)  
(219) 489-6001 (Home)

#### ED:

For those of your readers that are considering adding double sided disk drives to their machines, I feel it would be wise for them to be wary of ads for Cannon 2/3 height drives at very low prices (I purchased two for \$85.00 plus shipping).

After the drives arrived, I encountered some major difficulties, and spent several days trying to figure out what was going on. I talked with the local floppy disk drive repair shop and found out that these particular drives were well known for their erratic behavior. It seems that the drives were made by Cannon as an OEM part, and ended up on the surplus market because of defects in either the design of the electronics, or bad components.

Of the two drives that I purchased, I was able to get one of them working after I discovered that it was "late" in putting data on the disk. This problem was solved by using my "RPM" program to set the disk speed at 302.2 RPM (that seems high, but it works).

## NEW!

USE LOTUS 1-2-3 WITH OS-DMS DATA!

DOWLOAD DMS FILES (TYPE 10)  
TO  
MS AND PC-DOS COMPUTERS  
AND  
INTERFACE TO LOTUS 1-2-3  
OR  
GENERATE YOUR OWN PROGRAMS.

COMPLETE DATABASE ON PC.  
OS-65U PROGRAMS PREPARE AND  
SEND DATA.

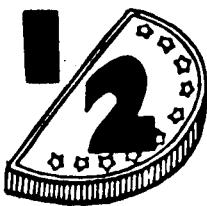
With manual \$195.00  
Manual only \$ 35.00

#### SPECIAL!

IHS SCRIBE  
WORD PROCESSOR

Was \$195.00  
Now \$150.00

IHS Computer Services  
2515C East Market Street  
Harrisonburg, Va 22801  
(703) 434-4177



# PRICE

# INVENTORY SALE

OUR STOCK ROOM IS OVERFLOWING!

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

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

## GOODIES for OSI Users!

### PEEK (65)

The Unofficial OSI Users Journal

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

- ( ) **C1P Sams Photo-Facts Manual.** Complete schematics, scope waveforms and board photos. All you need to be a C1P or SII Wizard, just \$7.95
- ( ) **C4P Sams Photo-Facts Manual.** Includes pinouts, photos, schematics for the 502, 505, 527, 540 and 542 boards. A bargain at \$15.00
- ( ) **C2/C3 Sams Photo-Facts Manual.** The facts you need to repair the larger OSI computers. Fat with useful information, but just \$30.00
- ( ) **OSI's Small Systems Journals.** The complete set, July 1977 through April 1978, bound and reproduced by PEEK (65). Full set only \$15.00
- ( ) **Terminal Extensions Package** - lets you program like the mini-users do, with direct cursor positioning, mnemonics and a number formatting function much more powerful than a mere "print using." Requires 65U. \$50.00
- ( ) **RESEQ.** - BASIC program resequencer plus much more. Global changes, tables of bad references, GOSUBs & GOTOs, variables by line number, resequences parts of programs or entire programs, handles line 50000 trap. Best debug tool I've seen. MACHINE LANGUAGE - VERY FAST! Requires 65U. Manual & samples only, \$5.00 Everything for \$50.00
- ( ) **Sanders Machine Language Sort/Merge** for OS-65U. Complete disk sort and merge, documentation shows you how to call from any BASIC program on any disk and return it or any other BASIC program on any disk, floppy or hard. Most versatile disk sort yet: Will run under LEVEL I, II, or III. It should cost more but Sanders says, "...sell it for just..." \$50.00
- ( ) **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. \$89.00
- ( ) **Assembler Editor & Extended Monitor Reference Manual** (C1P, C4P & C8P) \$100.00
- ( ) **65V Primer.** Introduces machine language programming. \$6.95
- ( ) **C1P, C1P MF, C4P, C4P DF, C4P MF, C8P DF Introductory Manuals** (\$5.95 each, please specify) \$4.95
- ( ) **Basic Reference Manual** — (ROM, 65D and 65U) \$5.95
- ( ) **C1P, C4P, C8P Users Manuals** — (\$7.95 each, please specify) \$5.95
- ( ) **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
<b>TOTAL DUE</b>	\$ _____
POSTAGE MAY VARY FOR OVERSEAS	

The second drive is beyond my scope to repair, primarily because there is a "controller" chip (FUJITSU MB14036) that I cannot obtain a pinout for because it may be "X-Rated." This means that data sheets are not available to the hobby user in order to provide authorized dealers a market for repair work. The major problem with this drive is that it likes to "re-write" tracks on the source disk.

Another strange thing about these drives is a "double step" made for the "B" side. In this mode the "B" side of the drive steps twice for each step pulse from the controller. I found this mode by accident while trying to figure out why the Red/Green activity LED was not turning on the green when side "B" was in use.

Were it not for these problems, the Cannon drive would be an excellent buy. Some of the pluses are:

1. Direct spindle drive motor (no belts to wear out).
2. Door lock, - prevents opening the door during disk access.
3. Head Load, - good for saving media and heads from wear.
4. Door open switch for disk ready status (shades of 8"!).

Some of the minuses are:

1. 2/3 height, - difficult to find a case for, but mounting holes match full height cases.
2. Appear to be OEM REJECTS and may need repairs.
3. Will not work with Hexdos!

This last problem has really puzzled me and here are the symptoms:

1. Hexdos will boot, but will display a DT error when a program is selected from the menu.

2. LOAD! Causes the head carriage to "Buzz" against the mechanical stop. Seems as if Hexdos does not "see" track 0 pulse right away.

3. COPY and Test Disk (from Hexasm 1.0) Run, but copied disk shows DT errors.

Well, that's pretty much it. I now have a Tandom 100-1 as drive A, and one Cannon drive as B & D. The second Cannon is in the box waiting for me

to come up with enough to cover the repair costs.

One last thing for your die-hard hacker readers. If you feel up to the challenge, check into these drives, but don't do any testing with a master disk!

C. J. Hipsher  
Virginia Beach, VA 23456.

\* \* \* \* \*

ED:

We have been a dealer of Ohio Scientific Computers since 1979.

Here in Trinidad and Tobago, we have several of the old Diablo HiType II letter quality printers. Do you know how we can interface these printers to Apple//e, IBM PC or Macintosh? Our clients out there are hollering!!

If any of your readers has the answer, please write to us.

George Martin  
Computer Systems Ltd.,  
3 Rust Street  
St. Clair  
Trinidad and Tobago

## AD\$

FOR SALE: C2-OEM with 48k RAM. Single density 8 inch floppy drives. Upgraded with 2 megahertz speed enhancements. Runs on OS-65U Operating System. Hazeltine 1420 terminal. \$500 or best offer. Call Frank Sullivan, 303-482-7777.

\* \* \* \* \*

WANTED: Print (schematic) of a 530 board. Bob Groome, 824 W. Main St., Richmond, IN 47374.

\* \* \* \* \*

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

\* \* \* \* \*

FOR SALE: OSI 2+2 64K 6502/65U/65D with Intertec "Inter-Tube CRT" and TI810 (fully loaded) printer and Rixon 212A modem. (\$7800 new) \$2800 or best offer, must sell. Dallas Porter (301)-937-1363.

\* \* \* \* \*

FOR SALE: OSI Model C3-B-55 Computer, Installed 1982, 74 Meg. Winchester Disk, 2 each

Floppy Disk Drives, 5 CRT Multi-User, OS-65U Level III, with Bus. II, HDE, HDM and OSDMX Software. For information, call (219)-423-9461 Mr. Ron Lerch.

\* \* \* \* \*

FOR SALE: OSI C2D-52K with 10 Megabyte Winchester Disk and 8" Floppy OS-65U Level 1 Operating System with Hazeltine 1420 CRT. All in good working condition, no reasonable offer refused (315)-733-2803 weekdays.

\* \* \* \* \* 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!

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

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

\* \* \* \* \*

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

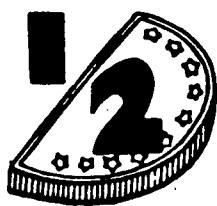
# PEEK [65]

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:



## PRICE INVENTORY SALE

OUR SHELVES ARE BULGING!

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

Get a one year volume set (12 back issues) for \$15.00 and we will pay UPS.

Get one back issue of the OSIO newsletter free with order.

Foreign orders by VISA/MASTER/CHOICE only, plus postage.

Orders must be postmarked not later than August 31, 1985.

Orders can not be sent to P.O. Boxes.

NAME.....STREET.....

CITY.....STATE .....

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

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

JAN #1 FEB #2 MAR #3 APR # 4 MAY # 5 JUN # 6  
JUL #7 AUG #8 SEP #9 OCT #10 NOV #11 DEC #12

Vol 2. 1981 ( )  
JAN #1 FEB #2 MAR #3 APR # 4 MAY # 5 JUN # 6  
JUL #7 AUG #8 SEP #9 OCT #10 NOV #11 DEC #12

Vol 3. 1982 ( )  
JAN #1 FEB #2 MAR #3 APR # 4 MAY # 5 JUN # 6  
JUL #7 AUG #8 SEP #9 OCT #10 NOV #11 DEC #12

Vol 4. 1983 ( )  
JAN #1 FEB #2 MAR #3 APR # 4 MAY # 5 JUN # 6  
JUL #7 AUG #8 SEP #9 OCT #10 NOV #11 DEC #12

Vol 5. 1984 ( )  
JAN #1 FEB #2 MAR #3 APR # 4 MAY # 5 JUN # 6  
JUL #7 AUG #8 SEP #9 OCT #10 NOV #11 DEC #12