ERROR SIGNS - learning p12

PEEK (65)

The Unofficial OSI Users Journal

JUNE 1985
VOL.6, NO.6

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

INSIDE

Column One

Now that the dust of COMDEX is starting to settle, we can report some news, but as I said, the dust is still settling. From all reports, it was an excellent show. In spite of the fact that Isotron's booth was not exactly in the prime area, attendance was excellent as was the interest in the new 700 series of UNIX machines which were announced. In fact, orders were taken at the show for the 710 model. It is reported ready to go with delivery scheduled for the middle to end of June - just as soon as the final touches are put on the dealer training program. That is also about the same time frame for the release of the Portland Boards.

What you probably have not heard about Isotron is the partial buy-out! The story goes something like this: They had sufficient funds to get the new 700 series developed (a very large undertaking), but wanted to insure that additional funds would be available to insure a proper job of marketing. Although we don't know the exact percentage of the buy-out, it is reported to be close to the 50% mark and made up by the stock purchases of an unnamed U.S. private investor and of AHLSELL, another Swedish investment house that already had considerable involvement in the computer world. They just happened to hold the key to a version of UNIX that, unlike most others, would be compatible with most UNIX versions including ATT's v 6 & 7, ZENIX's v 7 and soon to be released v 5 and the proposed UNIX Users Group ANSI standard. The sale has reportedly put Isotron in a very healthy position.

DBI was at COMDEX too, albeit without a booth, and they report that it was a very rewarding experience. What better way to go to the show than with a record breaking April under your belt and May looking even better! There must be something to the SCSI bus and all the low cost, high density, error checking reliability that is available on the DBI boxes. Prices aren't mentioned in their ad, but from one who knows, they are worth checking into. Their 65E operating system is getting very close to an official release. The hold-up is the implementation of several important suggestions that came out of the last distributor's seminar. From what we hear, the wait will be well worth it. This operating system will be packed with far more than its 16 digit precision. Many of the proverbial shortcomings of OS-U have not only been fixed, but turned into tremendous assets.

Though we are not insurance vendors, we thought you might be interested in Data Security Ins. of Bolder, CO. They just merged with Personal Computer Ins. and now offer coverage underwritten by St. Paul that includes all software and data. Coverage can start as low as $5,000. Sounds interesting, no?

Congratulations to our writers! For months we have been harping on formats, techniques and subjects. Now it would seem that you can scan through this issue and use it as a guide. In particular, most everything here is presented in a fashion that can be understood by those new to the particular field - even though the subject may be deep and technical. When writing an article, it is so easy to assume that the reader already knows the generalities, but because so many readers are just now becoming involved with the inner workings of what they have, the extra hand holding explanations are the key factor in determining whether they will delve into these new areas. With your help, PEEK readers are gaining the confidence to explore the wondrous capabilities of these machines. On behalf of our readers, our thanks for jobs well done.

Eddie

THE LAYMAN'S GUIDE TO MACHINE CODE PROGRAMMING FOR OS-65U

By: Rick Trethewey
8 Duran Court
Pacifica, CA 94044

OS-65U is an intimidating operating system to try to write Assembly Language programs for. To begin, there's no assembler that runs under that operating system. You have to write your program and assemble it under OS-65D and port it over somehow. You have two ways of getting the machine code to 65U. First, you can use either "LOAD32"or"LOAD48" which are programs that can read OS-65D diskettes. Second, you can assemble your program and leave it in an area of memory that doesn't get overwritten when you reboot under OS-65U. Clearly, the first method is the better of the two. Still, that only solves part of the problem. You still have to get the machine code in the desired area of memory and get it safely stored on your OS-65U diskette. The complexity of this problem is largely dependent on how many bytes of object code your Assembly Language program generates and how big the BASIC application program is that will be using it.

There are two common areas of memory to store machine code programs that are executed by BASIC programs. The first is to store it in front of your BASIC program. The second is at the top of memory. Each location has advantages and disadvantages that have to be considered. To begin, let's take a look at OS-65U's useage of memory. All locations below (i.e. less than) 24576 are out of bounds, leaving the high half of memory (assuming a system with 48K of memory) to hold the BASIC program, variables, and your machine code. This free region of memory is called the workspace because its contents change with the programs as they are called into memory and executed. The low end remains largely unchanged. When a BASIC program is loaded into the workspace, it is stored in memory beginning at roughly $6000(24576 decimal). However, the actual text of the program need not begin at this location. You can reserve space between location $6000 and the start of the text to hold your machine code. When machine code is stored in this fashion, it is stored on disk with the BASIC program and is, therefore, automatically called into memory when the BASIC program is RUN. By the same token, however, the code vanishes as soon as the next program is loaded. The alternative is to have your machine code stored in its own file on disk and called into memory. If the machine code is stored at the very top of the workspace, you can do a POKE that will protect it from BASIC and other programs and it will remain there until the protection is removed. As you can see, code that may be useful to many programs might make the second alternative attractive. However, protected areas of memory are no longer available to other programs that might need the space. On the other hand, if you have to store a duplicate copy of the machine code in front of each BASIC program that might need it, you've got to use extra space on the disk. I think the old mainframe addage "Disks are cheap, core is expensive" applies very well here. Machine code routines tend to be very application-dependent, reducing the value and need for storing code in high memory.

All right, we've decided where our code will reside in memory, namely at $6000. The next step is to write and assemble our machine code program. As always with OS-65D, you must begin by creating files. One file will be required to hold the Assembly Language program and another file will be needed to hold the object code. Be generous in allocating space here, and jot down the track numbers of the object code's file.

If you're using OSI's Assembler/Editor, you will need to become familiar with two commands, "H" and "M". The reason is that under OS-65D, $6000 lies almost in the middle of the workspace, rather than at the low end. Further, the text of your Assembly Language program must also reside in memory—and may well extend from its start at $3A7E (for V3.3) to well beyond $6000. If you assemble directly to $6000, by the time the assembler gets to the end of the program, it will have overwritten the program text with machine code. Ergo, you might have to assemble with what is called an "offset." In this context, the offset is the number of bytes which are added to the origin address of the Assembly Language program at which the object code is actually stored in memory when it is assembled. However, you must also be mindful that the assembler has memory requirements of its own. Just like BASIC, the Assembler has to save the values of the variables (i.e. labels) within the workspace, and it does this by building a list of the labels and their values, beginning at the top of memory and building downward toward the program text. This leads to another possible conflict that you have to be careful to avoid.

If you're fluent in OS-65D and the Extended Monitor program, you can find out the exact memory address where your Assembly program ends. A quick alternative is to enter "S" at the Assembler's "." prompt. The Assembler will report the size of your program in number of tracks. Multiply the number of tracks by 3. If the result is less than 8, you can safely assemble your program without an offset. If it is greater than 8, divide the result by 4 and use that result for your offset using the "M" command, as in "M1000", "M2000", "M3000" ... it'll be close enough, trust me.

To protect the high end of memory, you use the "H" command. Fortunately, the Assembler has truly modest needs for storing the symbol table. If you have a 32K system, enter "H7800" and if you have a 48K system, enter "HB800". Should the Assembler issue an out of memory error with these settings, try "H7000" and "HB000" respectively. The first setting will allow for approximately 256 labels and considering the OSI Assembler's limitations, this should be sufficient in the vast majority of cases.

If you're using my assembler, ASM-Plus, just set the start of the symbol table to $7800 on 32K system or $B800 on 48K systems and assemble with a 0 offset.

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

published monthly

Editor - Eddie Gieske

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

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

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

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

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

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

Once the program has been assembled to memory, leave the Assembler/Editor with the command "EXXX". This will send you to the "A*" prompt. Now we want to save the object code on the disk. For this example, I will use "T1", "T2", and "T3" to denote the first, second, and third tracks of the object code file. The actual track numbers you will use will be the ones you wrote down when you created the object code file. Save the object code to disk with the commands:

```
SA T1,1=6000/B
SA T2,1=6B00/B
SA T3,1=7600/B
```

Of course, you may not need to issue all of these commands if your machine code program ends at memory addresses lower than the second and third commands. You might not even need the full "/B" pages in the first. Once this is done though, you're halfway home.

Re-boot your system under OS-65U. If you haven't done so already, create the file that will hold the object code and your BASIC program. Be generous here in the size of the file you create. Now run either "LOAD32" or "LOAD48" as appropriate for your system. You will see the familiar "A*" prompt. Insert your OS-65D diskette that holds the machine code file. LOAD32 and LOAD48 do not respond like the real OS-65D. They automatically insert the "=" and "," in the appropriate places. So, for our purpose, enter the following keystrokes (without a <RETURN>!);

```
C6000T11 (which will display
     "C6000=T1,1")
C6B00T21 (which will display
     "C6B00=T2,1")
C7600T31 (which will display
     "C7600=T3,1")
GBE12    (for "LOAD48" or
     "G7E12" for "LOAD32")
```

You will now see the "OK" prompt. Before you do anything else, enter "NEW" followed by the length of the object code in bytes (with a hefty fudge factor) all in one line as:

```
NEW 8100
```

This command clears the workspace of the old program and sets the start of BASIC 8100 bytes higher than normal, preserving the machine code we just called into memory. Now enter:

```
10 REM
```

and enter the SAVE command to save this program in the BASIC program file you created. You can now continue to add BASIC programming to this file. To point BASIC's "USR(X)" function to your machine code, you must include the following POKEs in your program:

```
POKE 8778,0: POKE 8779,96
```

After that, all that is required is an "X=USR(X)" to execute the program.

You now have the mechanical skills needed to get machine code interfaced to OS-65U. The next step is learning to write the Assembly Language programs. We'll cover that in the next article.

★

OS-65U SELECTIVE SEARCH AND PRINT PROGRAM

By: Raymond D. Roberts
P. O. Box 336
Ferndale, WA 98248

(Continued from last month)

ADS100 & ADS200 are what I would call "boiler-plate" or "template" programs. What I mean by this, is that both were constructed from a standard program form that can be used over and over for many different programs. This can save a lot of typing in of the same routines that most OS-65U programs can and probably will use. Side by side examination of ADS100 and ADS200 will show these same routines in each program while the application of the programs are entirely different.

These commonly used routines are:

LINE 8 Numerical variable setting.
LINE 9 Flag setting.
LINE 35,36 Hard copy printing formatting.
LINE 31,62400-63000 Julian calendar routine.
LINE 60 Printer control settings.
LINE 61-65 Output device selection.
LINE 66 Control O and Control C disabling.
LINES 70-76 Date evaluation set-up.
LINES 90-94 Data file device selection.
LINES 160-350 File opening & loading.
LINES 40000-51150 Error handling
LINE 51152 Printer form feed at end of printing session.

All of these "routines" (some of which are subroutines) are stored on a disk file called "program file." Assume that I want to "write" a program. I load the "program file" and save it to a "scratch file" for development. I then need to program only those features that I want the program to perform and some modification of the existing "routines." The extraneous lines can then be deleted if desired, or left in for future expansion as I have chosen for these programs.

Because they are not needed for the program at this time, the following can be deleted without effecting the program if saving a little space is important. LINES 7,8,P9=1 in LINE 9, 30-36, TF=1 in LINE 300, last CF=1 in LINE 391, F1=k1 in LINE 33990, Subroutine in LINES 62400-63000.

If this style of structure does nothing else, it saves a lot of typing and keeps the same "routines" in the same part of the programs for easier understanding of the programs, how they work and what they do. I do not presume to advocate this style, just share it. If you try this style, remember to leave sufficient unused line numbers between "routines" and/or have a renumberer program.

POKES & FLAGS

POKES and FLAGS used in ADS100 & ADS200 are quite standard, but here is what they do.

FLAG 6 Enables program abort and error message upon pointer (INDEX) reaching the end of data file.

FLAG 9 Enables program control retention upon disk error. (Goes to line 50000 error handling.)

FLAG 11 Enables space suppression in numeric output to files. Normally, space is reserved for + or - values. If + or - signs are not used, this space is wasted. FLAG 11 will save these spaces.

FLAG 21 Disables input escape on carriage return. This allows you to use a carriage return without causing the program to abort and go into immediate mode.

At the end of the program run, FLAGS 9, 11 & 21 are reset to the default values, namely 10, 12, & 22. See LINES 51130 to 51150.

```
1 REM ==== A D S 2 0 0 ====Copyright 1983 R.ROBERTS
2 REM 1-5-83 R.D.ROBERTS POB 336, FERNDALE,WA 98248
3 REM ----))) FLAG EXPIRED ADS FOR DELETION
4 REM - THIS PROGRAM WILL READ ENTIRE ADS FILE AND
5 REM - MARK ^P IN FIRST TWO BYTES OF A RECORD IF
6 REM - IT IS EXPIRED, BASED ON TODAY'S DATE-1
7 REM -
8 K0=0:K1=1:K2=2:K3=3:K4=4:K5=5:K6=6:K7=7:K8=8:K9=9
9 P9=1:FLAG6:FLAG9:FLAG11:FLAG21:FORX=1TO23:PRINT:NEXT
30 POKE 2976,44:REM ALLOW , TERMINATION
31 GOSUB 62400:REM FILL VARIOUS ARRAYS
35 SP$="----------":SP$=SP$+SP$+SP$+SP$+SP$+SP$+SP$
36 SQ$="          ":SQ$=SQ$+SQ$+SQ$+SQ$+SQ$+SQ$+SQ$
39 POKE 2976,13
40 CLOSE
50 PRINT"ADS200 - FLAG EXPIRED ADS FOR DELETION"
51 PRINT LEFT$(SP$,60)
53 PRINT:PRINT"THIS PROGRAM WILL FLAG EXPIRED ADS WITH A"
54 PRINT"'^P' IN FIRST TWO POSITIONS OF THE AD RECORD, SO THAT "
55 PRINT"THEY CAN BE DELETED BY THE DMS NUCLEUS"
56 PRINT LEFT$(SP$,60)
57 PRINT:PRINT"******* YOU SHOULD HAVE BACKED-UP YOUR FILE FIRST****"
58 PRINT
60 T=PEEK(14387):POKE14457,(T-6):POKE15908,(T-6):REM PRNTR CNTRL
61 INPUT"PRINT AUDIT ON CONSOLE(C) OR PRINTER(P) OR QUIT (Q)";D$
62 IF D$="P" THEN DV=5
63 IF D$="C"THEN DV=2
64 IF D$="Q" THEN 51100
65 IF DV=0 THEN PRINT"WHAT!!":GOTO61
66 POKE 14639,255:POKE2073,76
70 PRINT:INPUT"ENTER DELETION DATE (MM/DD/YY)";DT$
71 IFLEN(DT$)()8THENPRINT"*** ILLEGAL ENTRY ***":GOTO70
72 IFMID$(DT$,3,1)()"/"THENPRINT"*** ILLEGAL ENTRY ***":GOTO70
73 IFMID$(DT$,6,1)()"/"THENPRINT"*** ILLEGAL ENTRY ***":GOTO70
75 XX$=RIGHT$(DT$,1)+LEFT$(DT$,2)+MID$(DT$,4,2):REM YYMMDD
76 EX=VAL(XX$):REM VALUE OF EXPIRATION DATE YYMMDD
90 INPUT"ENTER DEVICE AD FILE IS ON";MD$
91 IF MD$()"A" AND MD$()"B" THEN GOTO90
92 DV(2)=PEEK(9832): IF DV(2))127 THEN DV(2)=DV(2)-128+4
94 DEV MD$
96 INPUT"WHAT FILE NAME ";ZZ$
110 MN$=ZZ$:MN$="PASS"
118 MN$=MN$+"0":
160 OPEN MN$,MP$,1: REM OPEN AD FILE
170 INDEX(1)=0: INPUT %1,N$:
190 INDEX(1)=6: INPUT %1,TY:
210 INDEX(1)=9: INPUT %1,EODF:
220 INDEX(1)=20: INPUT %1,BODF:
250 INDEX(1)=31: INPUT %1,RL:
260 INDEX(1)=42: INPUT %1,NR:
280 IF(EODF(=BODF)ORNR(1THENERR$="FILE EMPTY":GOTO40000
290 DIM A$(20),L$(20),FP(20):REM CONTENTS & POINTERS
300 INDEX(1)=53:N=1:NF=1:TT=0:TF=1
305 INPUT%1,T$:INPUT%1,T
310 A$(N)=T$:FP(N)=TT: REM FIELD LABELS AND DESCRIPTIONS
320 IFINDEX(1))=BODFTHEN360
330 N=N+1:NF=NF+1:REM NF IS NUMBER OF FIELDS
340 TT=TT+T:REM RECORD LENGTH
350 GOTO305
360 TY=0:REM RECORD NUMBER
700 H1$="DELETIONS FROM AD FILE, EXPIRE DATE BEFORE "+DT$
720 GOSUB30000:REM GET A REC
721 IF NN=1 THEN GOTO 950:REM EOF DONE
723 IF DV=2THEN800
725 IFPEEK(15908)(L+5THENGOSUB735
726 IFPEEK(15908)(L+5THENFORI=1TOPEEK(15908):PRINT#DV:NEXT:REMSKIPPG
727 IFPEEK(15908)((PEEK(14457)-L+5)THEN800
728 PRINT#DV,CHR$(14);TAB(10);H1$;
732 PRINT#DV,"--) PRINTED ON ";DT$
733 GOSUB 735:REM HEADING
734 GOTO 800
735 REM
770 RETURN
800 REM ----------FORMAT AUDIT
805 PRINT#DV
850 FOR I=1TONF
855 IF L$(I)()"" AND L$(I)()"0"THEN PRINT#DV,A$(I);TAB(20);L$(I)
860 NEXTI
895 PRINT#DV
920 SV$=""
940 GOTO 720
950 GOSUB 735
990 CLOSE1:GOTO51100
30000 REM----------SEARCH ADS FILE
30005 TX=(TY*RL)+BODF: REM RECORD BEGINNING ADDRESS
30010 IF TX)=EODF THEN NN=1:GOTO 33990:REM ALL DONE
33100 INDEX(1)=TX:INPUT%1,P$
33101 IF P$="^P" THENTY=TY+1:GOTO30005
33108 INDEX(1)=TX+FPTR(1):INPUT%1,EX$
33110 IF VAL(EX$))EXTHENTY=TY+1:GOTO30005
33200 INDEX(1)=TX
33236 FOR I=1TONF
33237 INPUT%1,L$(I):REM GET ALL FIELDS OF AD
33240 NEXTI
33300 REM DETERMINE NUMBER OF LINES REQUIRED THIS AD
33305 L=0
33310 FOR I=1TONF
33315 IF L$(I)()"" AND L$(I)()"0"THEN L=L+1
33320 NEXTI
33350 INDEX(1)=TX
33351 PRINT#DV,"INDEX: ";TX
33360 PRINT%1," ^P";REM FLAG IT AS GONE
33370 TY=TY+1:REM GET READY FOR NEXT RECORD
33990 F1=K1:RETURN
40000 REM- ERROR
40010 PRINT: PRINT ERR$: PRINT: CLOSE 1: GOTO 51100: REM COMMON EXIT P
50000 REMDISK ERROR HANDLER
50010 ER=PEEK(10226): EL=PEEK(11774)+PEEK(11775)*256
50025 REM CHK FOR 'CHANNEL ALREADY OPEN ERROR'
50030 IF ER=133 THEN CLOSE: GOTO EL
50040 IF ER=128 THEN ERR$="INVALID FILE NAME": GOTO 51050
50050 IF ER=132 THEN ERR$="END OF FILE ERROR": GOTO 51000
50060 IF ER=130 THEN ERR$="ACCESS RIGHTS VIOLATION":GOTO 51050
50070 IF ER=129 THEN ERR$="CANNOT ACCESS FILE ": GOTO 51050
50075 REM OTHER ERRORS ARE HARD ERRORS
50080 ERR$="DISC ERROR CODE "+STR$(ER)+" IN LINE "+STR$(EL)
50094 EA=0: FOR I=4 TO 1 STEP -1: EA=EA*256+PEEK(9889+1): NEXT I
50096 DV(3)=PEEK(9832): IF DV(3))127 THEN DV(3)=DV(3)-128+4
50098 PRINT"ERROR ON DEVICE "+CHR$(DV(3)+65)+" AT DISC ADDRESS";EA
51000 REM-RROR EXIT
51020 CLOSE 1
51040 REM ENTRY AT '51050' DOES NOT CLOSE THE CHANNEL
51050 PRINT:PRINT:PRINT"***** ERROR *****": PRINT: PRINT
51060 PRINT ERR$
51100 REM---------- COMMON EXIT
51110 DEV CHR$(DV(2)+65): REM SELECT ORGINAL DEVICE
51120 FLAG 6: REM ENABLE PROGRAM ABORT ON EOF HIT ERROR
51130 FLAG 22: REM ENABLE BASIC'S IMM. MODE
51140 FLAG 12: REM DISABLE SPACE SUPPRESSION
51150 FLAG 10: REM ENABLE PROGRAM ABORT ON DISC ERROR
51152 REM FOR1=1TOPEEK(15908):PRINT#DV:NEXTI
51155 STOP
51160 RUN"ARCSYS","PASS"
62400 REM --FILL MISC WORK ARRAYS
62410 DIM M1(13),M2(13),MM$(13)
62420 FOR M=1TO13:READ MM$(M),M1(M),M2(M):NEXTM
62430 DATA JANUARY,0,0
62431 DATA FEBRUARY,31,31
62432 DATA MARCH,59,60
62433 DATA APRIL,90,91
62434 DATA MAY,120,121
62435 DATA JUNE,151,152
62436 DATA JULY,181,182
62437 DATA AUGUST,212,213
62438 DATA SEPTEMBER,243,244
62439 DATA OCTOBER,273,274
62440 DATA NOVEMBER,304,305
62441 DATA DECEMBER,334,335
62442 DATA YEAREND,365,366
63000 RETURN
```

POKES

Certain memory locations are reserved for holding values used by either the programmer or the computer to accomplish different tasks. For a fairly comprehensive list of these reserved locations and their use, see PEEK(65) March '83, Vol. 4 No. 3, page 9.

THE DATA SYSTEM

- Stored Report Formats
- Stored Jobs, Formats, Calcs.
- Multiple Condition Reports
- Multiple File Reports
- Calc. Rules Massage Data
- Up to 100 Fields Per Record
- User Designed Entry/Edit Screens
- Powerful Editor
- Merges - Append, Overlay, Match
- Posting - Batch Input
- Nested Sorts - 6 Deep
- Abundant Utilities

HARDWARE REQUIREMENTS: 48K OSI, Hard Disk, serial system, OS-65U 1.42 or Later; Space required: 1.3 megabytes for programs and data.

PRICE: $650.00 (User Manual $35.00, credited towards TDS purchase). Michigan residents add 4% sales tax. 30 day free trial, if not satisfied, full refund upon return.

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

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

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

FINANCIAL PLANNER

- Loan/Annuity Analysis
- Annuity 'Due' Analysis
- Present/Future Value Analysis
- Sinking Fund Analysis
- Amortization Schedules
- Interest Conversions

HARDWARE REQUIREMENTS: 48K OSI, 8" floppy or hard disk, serial terminal system, OS-65U v. 1.2 or later.

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

DEALERS: Your Inquiries Most Welcome

GANDER SOFTWARE, Ltd.

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

"It Flies"

FROM THE FOLKS WHO BROUGHT YOU:
All This
THERE IS MORE COMING SOON:
Program Generator for TDS
Proposal Planner
Time and Billing A/R

The POKEs (loading these locations) and PEEKs (looking at these locations) that I used in these programs, are the following, with explanations as to their use.

LINE 30 POKE 2976,44 This causes the computer to stop its reading or printing when it encounters a comma (,). As an example, on a mailing label, Jones, Ralph would be printed as Jones.

LINE 39 POKE 2976,13 This allows commas (,) on input and, therefore, Jones, Ralph would be printed (reverse of POKE 2976,44).

LINE 60 T=PEEK(14387) Memory location 14387 holds a value (usually 66) of lines per page (device 5).

POKE 14457,(T) 14457 holds a value (in this case T (66)) of lines per page to be printed. If you wanted to type 60 lines per page, you could POKE 14457,(T-6).

POKE 15908,(T) This location holds a value of lines per page not yet printed. This value is initially set with (T), which is to say 66, and is decremented each time a line is printed and then reset to 66 at the start of the next page. You will set this value equal to the value in location 14457.

LINE 66 POKE 14639,255 & POKE 2073,76 location 14639 holds either a 0 or 255. A zero (0) in this location means that if you should input a (CNTRL) O, nothing else can be input until another (CNTRL) O is input. A 255 in this location disables the escape.

LINE 92 Location 9832 holds value of current disk drive, 0=A,1=B,2=C, etc..

LINE 50010 Location 10226 holds disk error number. 11774 & 11775 hold line number of error, (as in BS error in LINE 3010).

In LINE 725, the computer is told to look at the value in memory location 15908 (lines not yet printed) and if the value is less than L (total number of lines in current record to be printed) (see LINE 33300) less 5 (for bottom margin) then go to the subroutine at LINE 735 which prints a ===== line across the page and then returns to LINE 726.

In LINE 726 the computer is told to skip to the next page if less than enough lines remain to print the record.

In LINE 727 the computer is told that if the lines not yet printed is less than 66 minus the number of lines to be printed plus the 5 line margin, then format the record to be printed (see LINE 800).

Next time, I will go into file opening, loading, and handling under OS65-U and explain why we use one or two Data files, ADS and Class. For a better understanding of these programs and what is possible in the way of expansion, do not delete the extraneous lines.

★

BEGINNER'S CORNER

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

STOP THE DWARVE!
Part 2

Last month's article described how to create the store data efficiently, particularly for programs using graphics. A single FOR...NEXT loop was all that was required to test that the graphics printed as expected. Having established that the data behaves correctly, it is time to develop the rest of the program.

CHOICE

In the CHOICE block the decision is taken whether or not to play against the computer. If the choice is "Yes" three degrees of difficulty are offered in the next block - see lines 290-320. The program then reads in the appropriate file of words from disk. How to create the three files of words will be described next month.

GET A KEY

In DOS 3.3 the halting get-key routine is at $2336. DOS 3.2 users will need to change, in line 320, "2336" to "252B". And in line 330 change "9059" to "9815". CIP users replace 'DISK!"GO 2336"', with "POKE 11,0: POKE 12,253: X=USR(X)", and in line 330 change "9059" to "531".

When testing which key has been pressed, it is worth remembering that the VAL function sets to zero the value of all non-numeral key presses - see line 330.

"Z*" in line 340 - see last month's WAZZAT article for a full description of what "Z*" does.

READ A FILE

The TRAP command in line 350 transfers program control to line 1650 when there is a disk error. Without TRAP the program would stop, with control passing to immediate mode! A program without disk error trapping is very unfriendly.

The program will read, from disk, a file of words and store them in array W$. The trick now is to choose these words at random but not to choose the same word more than once. To do this it is necessary to understand how the RND function works.

RND FUNCTION

First of all RND(0) will always return the same number, - no use at all here. (But see final paragraph.) RND(1) will always return a different number. (Changing the "1" in RND(1) does not change anything - it is a dummy argument for RND.) The number produced by RND is always less than one.

Assume that NW=99 and V=0. A quick test reveals that the largest number returned by INT(100*(RND(1)) will be 99, i.e., one less than the 100. (The largest value that can be returned by RND(1) is 0.999999999!). It follows then that the largest value that can be returned by INT (N*(RND(1)) will always be N-1. So in line 430 a random number, at least one less than 100, is calculated and stored in "Y", i.e., Y=NW-V+1. If NW=99 and V=0 then the largest value that "Y" can take is also 99.

As an example, imagine that the number calculated by INT(100*(RND(1)) and stored in Y, is 50. Y=50. The 50th word in array W$ is now stored as the chosen word in WD$. That 50th word in array W$ is selected by the number stored in R(50), R(50)=50. ("R" is an array.) How come R(50) has a "50" stored in it? Because it was put there back in line 130 with, FOR C=0 to NW : R(C)=C M: NEXT.

What a crazy way to do things! But wait! The next step is to remove the "50" stored in R(50) and substitute for it the 99 held in R(99). Or more generally, R(Y)=R(NW-V). In our example, R(50) <-- 99,

(i.e., the 99 stored in R(99)). There is no way that the 50th word can be selected again!

But what about the "99"? It is now stored in "R(50)" and in "R(99)". True! It is a simple matter to ensure that the "Y" can never be 99. Add 1 to "V" and the maximum value that "Y" can now take from INT((NW-V+1)*RND(1)) is 98! Remember that "99" has been substituted for "50" in R(50), and will be found next time Y=50.

If you are still puzzled, try this little program,

```
10 DIM N(99)
20 FOR C=0 TO 99: N(C)=C: NEXT
30 FOR C=0 TO 99
40 X=INT((100-C)*RND(1))
50 PRINT N(X);
60 N(X)=N(99-C)
70 NEXT
```

You will see that all the numbers from 0 to 99 are chosen once only and at random. Phew!

INPUT A WORD

If the word to be guessed is input from the keyboard then its maximum length is fixed at 20 in the FOR ... NEXT loop in line 480. What follows in line 480 may be puzzling. I'm afraid it's thinking caps on again!

The keyboard-input subroutine beginning in line 990 will reject all non-alphabet characters and print an error message at the bottom of the screen. This subroutine is called in line 490. As each character is accepted it is printed on line 13. The trick here is to remember the cursor position when an error is made. The error message is printed at the bottom of the screen, but at the next correct input the cursor must return to its previous position on line 13.

This can be done in DOS 3.3 with the PRINT!(5) command, see line 480. The cursor's screen position is input to Y$ with INPUT Y$. Y$ stores two characters, one for the X coordinate and one for the Y coordinate. If "Y$=MF" then X=77-65=12 and Y=70-65=5. (ASC("M")=77 and ASC("F")=70.) A PRINT&(12,5); will return the cursor to its correct position. To avoid the cursor bobbing up and down a cursor up command is required, PRINT!(12); - see line 490. Have a look at Listing 2 if you wish to experiment with

```
220 REM ----------CHOICE----------
230 PRINT C$;&(9,4)"Hello Smiler!"&(14,8)G$
240 X=2: Y=13: PRINT &(X,Y)"Would you like the computer": Y=15
250 PRINT &(X,Y)"to choose a word ? Yes"L1$;: GOSUB 990: J$=Y$: PRINT C$
260 IF J$="n" THEN 470
270 :
280 REM ----------READ A FILE OF WORDS OFF DISK----------
290 PRINT !(11)!(11)" How difficult would you like"&(9,6)"the words ?"
300 PRINT &(14,9)G$&(3,13)"1) Kids stuff."&(3,15)"2) Average."
310 PRINT &(3,17)"3) I'm writing a dictionary."
320 PRINT &(0,20)"Which ? "L$;: DISK !"GO 2336"
330 Y=VAL(CHR$(PEEK(9059))): IF Y=0 OR Y>3 THEN PRINT C$;: GOTO 290
340 PRINT C$;&(0,22)" Reading file )"Y"( from disk ";: Z$
350 TRAP 1650: DISK OPEN,6,Y$(Y): INPUT #6,T
360 IF T>NW THEN PRINT C$;&(8,9)"File too large.": DISK CLOSE,6: GOTO 1360
370 NW=T: FOR C=0 TO T: INPUT #6,W$(C): NEXT : DISK CLOSE,6: Z$: PRINT C$
380 TRAP 1670: GOTO 430
390 :
400 REM ----------INPUT A WORD----------
410 IF DW$=" " THEN 1290
420 FOR K=0 TO 26: D(K)=0: W(K)=0: P(K)=0: NEXT : WD$="": IF J$="n" THEN 470
430 Y=INT((NW-V+1)$RND(1)): WD$=W$(R(Y)): R(Y)=R(NW-V): V=V+1: T=LEN(WD$)
440 IF V>NW THEN V=0: FOR C=0 TO NW: R(C)=C: NEXT
450 FOR C=1 TO T: W(C)=ASC(MID$(WD$,C,1)) OR 32: P(C)=C: NEXT : GOTO 560
460 :
470 POKE 13026,241: PRINT &(2,11)"What is your word please ?"!(11)!(11)R$
480 FOR C=1 TO 20: PRINT !(5): INPUT Y$: F=ASC(Y$)-65: P=ASC(RIGHT$(Y$,1))-65
490 PRINT !(12);: GOSUB 990: PRINT &(F,P);: IF Y=8 THEN GOSUB 1080: GOTO 530
500 IF Y=45 AND C<3 THEN 490
510 IF Y=45 THEN C=20: GOTO 530
520 PRINT Y$;: T=C: W(C)=Y: P(C)=C: WD$=WD$+CHR$(Y)
530 NEXT C: POKE 13026,32: IF WD$="EXIT" THEN 1360
540 :
550 REM ----------PRINT SCAFFOLD----------
560 PRINT C$: F=1: L=30: GOSUB 1050: Y$="": FOR C=1 TO T: Y$=Y$+"-": NEXT
570 PRINT &(1,14)Y$&(23,16)CHR$(222)&(0,1)DW$&(1,4)ST$: F=31: L=35
580 P=0: A=1: Z$=P3$: GOSUB 1150: GOSUB 1260
590 :
600 REM --------INPUT A CHARACTER & CHECK IT LOOP----------
610 FOR C=1 TO R
620 GOSUB 990: IF Y<97 THEN 620
630 FOR K=1 TO R: IF Y=D(K) THEN K=R: NEXT K: Z$=P$+CHR$(Y)+G$: GOSUB 1150: GOTO 620
640 NEXT K
650 FOR K=1 TO T: IF Y=W(K) THEN PRINT &(P(K),14)CHR$(W(K)): K1=K: P=P+1
660 IF P=T THEN K=T: NEXT K: C=R: NEXT C: GOSUB 1170: GOSUB 1190: GOSUB 1260: GOTO 410
670 NEXT K: D(A)=Y: A=A+1: IF Y=W(K1) THEN Z$=P1$: GOSUB 1150: GOTO 620
680 Z$=P2$: GOSUB 1150: PRINT &(26,19)R-C: IF C=R-1 THEN PRINT &(18,6)"Help!"
690 GOSUB 1050: : L=L+1: F=L: IF C=5 THEN GOSUB 1230: T1=T2
700 IF C>R-5 THEN PRINT &(M,N)F$: M=M+1: N=N-1: PRINT &(M,N)G$
710 NEXT C
720 :
730 REM ==========WORD NOT GUESSED!==========
740 PRINT &(9,23)"THAT'S IT!"R$;: IF LEN(DW$)<20 THEN DW$=DW$+G$
750 :
760 REM --PRINT DWARVE, THROW PIN, FLASH EYES, DROP TRAP DOORS-
770 GOSUB 1260: F=45: L=45: GOSUB 1050: GOSUB 1260
780 :
790 X=23: Y=16: FOR K=1 TO 4: FOR C=1 TO 4: PRINT &(X,Y)F$
800 FOR P=1 TO 50: NEXT P: X=X-1: Y=Y-1: PRINT &(X,Y)CHR$(224-C)
810 FOR P=1 TO 50: NEXT P,C,K: PRINT &(X,Y)F$&(7,0)CHR$(142)&(7,1)H$
820 :
830 X=25: Y=5: FOR C=4 TO 20 STEP 2: PRINT &(X,Y)C$(53): FOR K=1 TO 99: NEXT K
840 PRINT &(X,Y)C$(33): FOR K=2000/(C$LOG(C)) TO 1 STEP -1: NEXT K,C
850 :
860 F=46: L=52: GOSUB 1050: GOSUB 1260
870 :
880 REM ----------DROP FIGURE, MAKE DWARVE JUMP----------
890 FOR C=3 TO 6: PRINT &(25,C)C$(29): NEXT : F=31: L=44: FOR C=F TO L
900 PRINT &(X(C),Y(C)+4)C$(C): NEXT : PRINT &(25,11)C$(54)&(25,9)C$(53)
910 :
920 P=100: FOR C=1 TO 400: NEXT : FOR C=1 TO 5: PRINT &(22,16)F$: FOR K=1 TO P: NEXT K
930 PRINT &(22,14)H$: FOR K=1 TO P: NEXT K: PRINT &(22,14)F$: FOR K=1 TO P: NEXT K
940 PRINT &(22,16)G$: FOR K=1 TO P*3: NEXT K,C: FOR C=1 TO 4000: NEXT
950 GOSUB 1190: GOSUB 1260: GOTO 420
960 :
970 REM ==========SUBROUTINES==========
980 REM ----------GET A KEY----------
990 DISK !"GO 2336": Y=PEEK(9059) OR 32: Y$=CHR$(Y): IF Y=D THEN Y=8
1000 IF Y=E THEN RETURN
1010 IF Y<8 OR Y>G THEN Z$=P3$: GOSUB 1150: GOTO 990
1020 RETURN
```

```
1030 :
1040 REM ------------PRINT A PICTURE-------------------------
1050 FOR Q=F TO L: PRINT &(X(Q),Y(Q))C$(Q): NEXT Q: RETURN
1060 :
1070 REM ------------EXTENDED INPUT B/SPACE------------------
1080 IF LEN(WD$)<2 THEN WD$="": GOTO 1100
1090 WD$=LEFT$(WD$,LEN(WD$)-1)
1100 T=T-1: IF C>1 THEN PRINT L$" "L$;
1110 C=C-2: IF C<1 THEN C=0
1120 RETURN
1130 :
1140 REM ---------------PRINT A MESSAGE-----------------------
1150 PRINT &(4,23)Z$R$;: GOSUB 1260: PRINT &(0,23)!(15);: RETURN
1160 :
1170 PRINT &(18,6)"Saved!": GOSUB 1260: DW$=LEFT$(DW$,LEN(DW$)-1): RETURN
1180 :
1190 PRINT C$;&(9,2)"The word was": Y=LEN(WD$): X=(31-Y)/2
1200 PRINT &(X,4)WD$: FOR C=1 TO Y: PRINT &(X+C-1,5)"-": NEXT C: RETURN
1210 :
1220 REM ---------------DWARF PUSHES STEP--------------------
1230 H=30: N=21: FOR M=0 TO 16: PRINT &(M,N)S$: FOR K=1 TO H*2.5: NEXT K
1240 PRINT &(M,N)F$: FOR K=1 TO H: NEXT K,M: PRINT &(M,N)G$+I$: RETURN
1250 :
1260 FOR Q=1 TO T1: NEXT Q: RETURN
1270 :
1280 REM ---------------------END----------------------------
1290 GOSUB 1260: PRINT C$;&(8,7)"Congratulations!"
1300 PRINT &(1,10)"You're vocabulary and spelling"
1310 PRINT &(5,12)"are really excellent!"
1320 PRINT &(2,19)"Play again ? ";: GOSUB 990: IF Y$="n" THEN 1360
1330 PRINT C$!(17,9,11)"Please wait.": ST$=ST$+CHR$(42)
1340 FOR C=1 TO 5: DW$=DW$+G$: NEXT : Y=FRE(Y): GOTO 420
1350 :
1360 PRINT C$!(17,9,11)"Bye for now!": GOSUB 1230: GOSUB 1260: POKE 13026,171
1370 PRINT &(M,N)F$&(M+1,N)H$&(M+1,N-1)I$&(0,0);: POKE 2073,173: Z$: END
1380 :
```

```
5 REM Listing 2
10 PRINT !(28)
20 PRINT : PRINT : PRINT : PRINT : PRINT "Cursor position is ";
30 PRINT !(5): INPUT Y$: F=ASC(Y$)-65: P=ASC(RIGHT$(Y$,1))-65
40 PRINT !(12) &(F,P) Y$, F, P
```

sending the current cursor address through the keyboard driver.

The length of the input word is stored in "T" in line 520. Array "W" stores the word as lower case ASC values. Array "P" stores the position of each character of the word. Array "D" stores the values of guessed characters.

GRAPHICS PRINT

Printing of all graphics is done in line 1050. Just two values, F and L, need to be passed to this subroutine and the work is done. It is a simple matter to control the printing of the figure: set F to L and increment L by one - see line 690. Redrawing the figure lower down is straightforward too, merely add 4 to the "Y" coordinate - see line 900. Some graphics are drawn by their own little routine. For example, the dwarve jumping up and down, done by lines 920-940.

STRUCTURE

Once the preliminary procedures have been executed, the program runs in lines 410 to 950 with calls to subroutines as required. Branching is always forward unless an unavoidable loop in required. All subroutines and data follow the main body of the program, where they can be quickly identified if need be.

RND REVISITED

Try this program -

```
10 Y=RND(0)
30 PRINT Y
40 GOTO 10
```

Now insert this line

```
20 A=RND(-A) : REM reset seed value for RND function.
```

Notice how the values increase in two streams and go through zero. Anyone care to graph the results?

★

ASM-SHARED POINTERS

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

Indirect addressing is a very powerful programming tool. In this article, a comparative study will be presented illustrating the benefits of indirect addressing for a common ASM routine-clearing the video screen.

Those of you familiar with ASM (Assembly Language) programming already know that indirect addressing refers to a page zero location (called a pointer) to find the actual address used by the associated opcode. The term "two-byte" instruction is often used to refer to indirect opcodes. The first byte contains the opcode while the second byte contains the page-zero location where the actual address is contained. This is to be compared with the absolute or "three-byte" instructions that contain the actual address in the second and third bytes.

Listing 1 shows ABSCLR, a clear routine using absolute addressing. This routine appears in the C4P Operators Manual and should be familiar to many readers. In this routine, three instructions use absolute addressing. In line 90, bytes two and three contain the video address where the contents of register A are stored. Lines 130 and 180 modify the video address by changing the value of byte three in line 90.

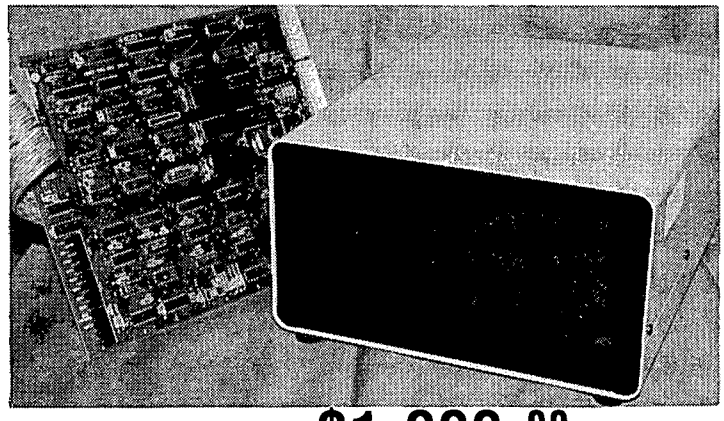
Although this routine does the job, there are two major problems, both stemming from use of absolute addressing. First, the code will not function if stored in ROM (Read-Only-Memory). It is desired that video routines be dedicated to ROM so that they are available at power-on. Second, absolute addressing does not promote "information sharing." We are interested in building a video window facility and would like to have several routines (e.g., CLEAR, SCROLL, DUMP, etc.) use a common video address base. Such an address base could be modified to create windows of arbitrary size and location on the screen.

Listing 2 shows INDCLR which uses indirect addressing to accomplish the same clearing function. All necessary video address information is stored in page zero where it is accessed via indirect instructions. For example, BEGSCR (at $68) specifies the upper-left corner of the screen. Other page-zero locations point to screen locations where output is to be printed. Also, screen size and blank character are specified at

SUPER HARD DISK Subsystem!

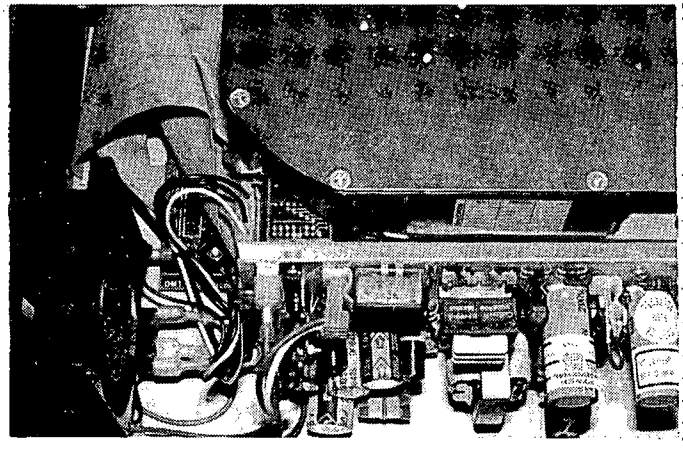
TURNS ANY FLOPPY BASED COMPUTER INTO HARD DISK BASED, INSTANTLY.

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



FROM **$1,999.00**

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



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

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

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

SPACE-COM International

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

page-zero locations $6E to $71.

In INDCLR, line 250 contains the indirect instruction which clears the screen. When this opcode is executed, the actual address is computed by adding the Y-register value to the pointer address. This is called indirect-indexed addressing since the pointer value is offset (indexed) to compute the actual address. Notice that a temporary pointer is used for the actual clearing. This is initialized to the upper-left screen address which is preserved during the clear operation.

With indirect addressing, page-zero pointers must be loaded with correct address values. Otherwise, random areas of memory may be erased! Listing 2 contains a short routine (starting at INIZ) illustrating page-zero initialization for half-screen C4P operation. Alternate values for C1P are also given. Notice that the code is unchanged (i.e., portable) with machines having different screen addresses. The code is actually functional in any 6502 machine such as APPLE II, C64, ATARI, etc., given correct page-zero initialization.

HIGH-LEVEL ACCESS

Page-zero values may also be initialized using high-level language. For example, BASIC would use a series of POKEs to the page-zero locations used by INDCLR to define a window with position and size determined by the user. Subsequent calls to the clear routine would clear only the selected window area. The window is modified by changing only page-zero values.

For the past year, I have used a programming system, called BETA/65, which has improved machine access instructions. In addition to PEEK and POKE, DPEEK and DPOKE are available, allowing pointer modification with one instruction. In addition, LINK to ASM code (such as INDCLR) is provided. This is a vast improvement over USER(X), which is used by BASIC to access machine code.

CONCLUSIONS

Check your ASM routines for absolute (three-byte) instructions. Recoding these routines to use indirect (two-byte) instructions will improve code portability and permit "information sharing" of page-zero data. (Note that three-byte opcodes JMP and JSR do not have two-byte equivalents.)

There is additional overhead with indirect code as page zero must be properly initialized. However, the benefits greatly outweigh the minor overhead penalty.

★

LISTING 1

```
10              ; ABSCLR-CLEAR VIDEO SCREEN
20              ; USING ABSOLUTE ADDRESSING
30              ;
40 A000                 *=$A000
50 A000 A920            LDA #$20      ; LOAD BLANK CHARACTER
60 A002 A008            LDY #8        ; LOAD PAGE COUNT
70 A004 A200            LDX #0        ; ZERO COLUMN COUNTER
80                      ;
90 A006 9D00D0  BLANK   STA $D000,X   ; STORE BLANK TO SCREEN
100 A009 E8             INX           ; INCREMENT COLUMN COUNT
110 A00A D0FA           BNE BLANK     ; LOOP FOR FULL PAGE
120                     ;
130 A00C EE08A0         INC BLANK+2   ; INCREMENT PAGE COUNTER
140 A00F 88             DEY           ; DECREMENT PAGE COUNT
150 A010 D0F4           BNE BLANK     ; LOOP FOR FULL SCREEN
160                     ;
170 A012 A9D0           LDA #$D0      ; RESET SCREEN ADDRESS
180 A014 8D08A0         STA BLANK+2   ; TO INITIAL VALUE
190 A017 60             RTS           ; RETURN TO CALLING PROG
```

LISTING 2

```
10              ; INDCLR-CLEAR VIDEO SCREEN
20              ; USING INDIRECT ADDRESSING
30              ;
40 0068=        BEGSCR=$68    ; UPPER-LEFT SCREEN ADDRESS
50 006A=        RESET =$6A    ; OUTPUT RESET ADDRESS
60 006C=        OUTPUT=$6C    ; OUTPUT DISPLAY ADDRESS
70              ;
80 006E=        NCOL  =$6E    ; NR OF COLUMNS LESS ONE
90 006F=        NROW  =$6F    ; NR OF ROWS LESS ONE
100 0070=       LROW  =$70    ; ROW-TO-ROW INCREMENT
110             ;
120 0071=       BLANK =$71    ; BLANK CHARACTER
130             ;
140 00AA=       TEMP  =$AA
150             ;
160 A000                *=$A000
170 A000 A568           LDA BEGSCR    ; LOAD SCREEN ADDRESS
180 A002 85AA           STA TEMP      ; INTO SCREEN POINTER
190 A004 A569           LDA BEGSCR+1  ; DITTO FOR HIGH BYTE
200 A006 85AB           STA TEMP+1
210             ;
220 A008 A66F           LDX NROW      ; INIZ ROW COUNTER
230 A00A A571   ROWCLR  LDA BLANK     ; LOAD BLANK CHARACTER
240 A00C A46E           LDY NCOL      ; INDEX TO COLUMN END
250 A00E 91AA   COLCLR  STA (TEMP),Y  ; STORE BLANK TO SCREEN
260 A010 88             DEY           ; DECREMENT COLUMN COUNT
270 A011 10FB           BPL COLCLR    ; UNTIL ALL COLUMNS CLRD
280             ;
290 A013 18             CLC           ; ADD ROWLENGTH TO SCREEN
300 A014 A570           LDA LROW      ; ADDRESS TO OBTAIN FIRST
310 A016 65AA           ADC TEMP      ; ADDRESS OF NEXT ROW
320 A018 85AA           STA TEMP      ; SAVE AT TEMP
330 A01A 9002           BCC *+4       ; TEST FOR CARRY INTO
340 A01C E6AB           INC TEMP+1    ; NEXT PAGE OF SCREEN
350             ;
360 A01E CA             DEX           ; DECREMENT ROW COUNTER
370 A01F 10E9           BPL ROWCLR    ; UNTIL ALL ROWS CLEARED
380                     ;
390 A021 60     RET     RTS           ; RETURN TO CALLING PROG
400             ;
410             ;
420 A022 A209   INIZ    LDX #BLANK-BEGSCR
430 A024 BD2DA0 LOOP    LDA H32X64,X  ; USE H12X48 FOR C1P
440 A027 9568           STA BEGSCR,X
450 A029 CA             DEX
460 A02A 10F8           BPL LOOP
470 A02C 60             RTS
480             ;
490 A02D 00D0   H32X64  .WORD $D000,$D508,$D508
490 A02F 08D5
490 A031 08D5
500 A033 3F             .BYTE 63,20,64,32
500 A034 14
500 A035 40
500 A036 20
510             ;
520 A037 8BD0   H12X48  .WORD $D08B,$D1D1,$D1D1
520 A039 D1D1
520 A03B D1D1
530 A03D 2F             .BYTE 47,5,64,32
530 A03E 05
530 A03F 40
530 A040 20
```

★

WAZZAT CORNER!

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

This month, how to make a new character generator for the Superboard, or indeed for any OSI computer.

The graphic characters in the OSI PROM character generator (CG) are great, but the ASCII characters can be improved upon. Their readability, particularly on a C1P, is not as good as it could be.

The hex dump, listed here, provides a modified character set for upper and lower case letters. The first 8 bytes of the dump, all zeros, constitute the character #32. The final character is "z", at $53DO. The advantages of the new character set are: an extra row of blank dots between rows of characters; all descenders are two (not one) dots deep; all lower case characters are now of an even height. Lines of text are now much easier to read since they are wider apart. The number of lines that can be seen on the screen is, of course, not affected.

Every character is represented by eight bytes in the character PROM. The 8 hex bytes for "z" are: 0, 3E, 10, 08, 04, 3E, 0, 0. To see how they are calculated, examine the diagram for "z".

The numbers in the horizontal row represent powers of two. From "2 to the power of zero" (=1), to "2 to the power of seven" (=128). There are no dots in row 0. Therefore, for "z", the first byte for the first row must be 0. In row 1 the dots are in columns 1 to 5. This produces the value 62 (=2+4+8+16+32), and equals $3E in hex. Row 2 is "2 to the power of 4" (=16), or $10 in hex. The next five rows produce the values of 8, 4, 62 again, then 0 and 0. All 256 characters can be calculated in the same manner.

The CG PROM occupies 2K bytes (=8*256) and so a new CG can be programmed into a 2716 EPROM. The CG PROM, in older Superboards at least, is a 2316 chip. The 2316 can be read as if it was a 2716 if its pin 18 is connected to 5v DC. (Bend pin 18 out with long-nose pliers and hook to 5v). Any EPROM programmer should then be able to read a 2316. This can be done on the OSI programmer with its supplied software. If with the OSI programmer, memory at $5000 is all $FFs, then the copy was unsuccessful. If this happens, leave the 2316 in the MASTER socket and specify copy from the COPY socket, or vice-versa. Why this works I don't know. Now save the memory to disk. Next, type in the changes as listed here and save to disk again. Now program the new CG into a 2716. If possible check that the new code is in the 2716.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
5100  00 00 00 00 00 00 00 00 08 08 08 08 00 08 00 00
5110  14 14 14 00 00 00 00 00 14 14 3E 14 3E 14 00 00
5120  08 3C 0A 1C 28 1E 08 00 00 26 10 08 04 32 00 00
5130  04 0A 04 2A 12 2C 00 00 08 08 08 00 00 00 00 00
5140  10 08 04 04 08 10 00 00 04 08 10 10 08 04 00 00
5150  08 2A 1C 08 1C 2A 08 00 00 08 08 3E 08 08 00 00
5160  00 00 00 00 08 08 04 00 00 00 00 3E 00 00 00 00
5170  00 00 00 00 00 08 00 00 00 20 10 08 04 02 00 00
5180  1C 22 32 2A 26 1C 00 00 08 0C 08 08 08 1C 00 00
5190  1C 22 20 1C 02 3E 00 00 3E 20 18 20 20 1E 00 00
51A0  10 18 14 12 3E 10 00 00 3E 02 1E 20 20 1E 00 00
51B0  3C 02 1E 22 22 1C 00 00 3E 20 10 08 04 04 00 00
51C0  1C 22 1C 22 22 1C 00 00 1C 22 22 3C 20 1E 00 00
51D0  00 00 08 00 08 00 00 00 00 00 08 00 08 08 04 00
51E0  00 08 04 02 04 08 00 00 00 00 3E 00 3E 00 00 00
51F0  00 08 10 20 10 08 00 00 1C 22 10 08 00 08 00 00
5200  1C 22 2A 3A 1A 02 3C 00 1C 22 22 3E 22 22 00 00
5210  1E 22 1E 22 22 1E 00 00 1C 22 02 02 22 1C 00 00
5220  1E 22 22 22 22 1E 00 00 3E 02 1E 02 02 3E 00 00
5230  3E 02 1E 02 02 02 00 00 3C 02 02 32 22 3C 00 00
5240  22 22 3E 22 22 22 00 00 1C 08 08 08 08 1C 00 00
5250  20 20 20 20 22 1C 00 00 12 0A 06 06 0A 12 00 00
5260  02 02 02 02 02 3E 00 00 22 36 2A 2A 22 22 00 00
5270  22 26 2A 32 22 22 00 00 1C 22 22 22 22 1C 00 00
5280  1E 22 22 1E 02 02 00 00 1C 22 22 22 2A 14 20 00
5290  1E 22 22 1E 12 22 00 00 1C 22 0C 10 22 1C 00 00
52A0  3E 08 08 08 08 08 00 00 22 22 22 22 22 1C 00 00
52B0  22 22 22 22 14 08 00 00 22 22 2A 2A 36 22 00 00
52C0  22 14 08 08 14 22 00 00 22 22 14 08 08 08 00 00
52D0  3E 10 08 04 02 3E 00 00 3E 06 06 06 06 3E 00 00
52E0  00 02 04 08 10 20 00 00 3E 30 30 30 30 3E 00 00
52F0  00 00 08 14 22 00 00 00 00 00 00 00 00 00 3E 00
5300  00 00 00 00 00 00 00 00 00 2C 32 22 32 2C 00 00
5310  02 1A 26 22 22 1C 00 00 00 3C 02 02 02 3C 00 00
5320  20 2C 32 22 22 1C 00 00 00 1C 22 1E 02 1C 00 00
5330  10 08 1C 08 08 08 00 00 00 2C 32 22 32 2C 20 1C
5340  02 1E 22 22 22 22 00 00 08 00 0C 08 08 1C 00 00
5350  10 00 10 10 10 10 10 0C 02 12 0A 0E 0A 12 00 00
5360  0C 08 08 08 08 1C 00 00 00 16 2A 2A 2A 2A 00 00
5370  00 1E 22 22 22 22 00 00 00 1C 22 22 22 1C 00 00
5380  00 1A 26 22 26 1A 02 02 00 2C 32 22 32 2C 20 20
5390  00 1A 06 02 02 02 00 00 00 3C 02 1C 20 1E 00 00
53A0  08 3E 08 08 08 08 00 00 00 22 22 22 22 3C 00 00
53B0  00 22 22 14 14 08 00 00 00 22 22 22 2A 14 00 00
53C0  00 22 14 08 14 22 00 00 00 24 24 24 24 38 20 1C
53D0  00 3E 10 08 04 3E 00 00
:
```

```
  01234567
0
1 *****
2    *
3   *
4  *
5 *****
6
7
```

To do this, cut the track coming from pin 18 of the track coming from pin 20. Now use hookup wire to connect pins 18 and 20 to Ov ("ground"). The zero volts track is on the keyboard side of the Superboard. Insert the new CG 2716 and check pin 1 orientation. Switch on, the D/C/W/M prompt should be seen as normal.

Readers may be interested in designing their own characters for a CG. The way to do this is with a program. Such a program was published in MICRO in Dec '82. As it stands the program appears to be incorrect. I have modified and expanded it and rewritten it for the C4P screen. If readers have any problems with the conversion, I would be willing to help.

★

BITS AND PIECES

How to Find the Hi Byte

By: PEEK(65) Staff

The BASIC programmer will only rarely encounter the need to understand or directly use the Byte, but because one incomprehensible encounter is enough to halt the use of a program, let's examine the principles and one such use.

As need is what usually brings things to the front, let's consider PEEK's sophisticated line resequencer, editor and variable lister program called RESEQ. This is not a sales pitch, but the result of inquiries from users who have had difficulty in using the program on hard disks. The problem is that the author pulled a clever trick to expedite RESEQ's operation. The program picks up a program from the file INFILE, resequences it and puts it out in OUTFIL. In more normal operation, the programmer would have just OPENed the file, but that's slow going considering that the operating system must first OPEN DIR to find out where the file is located. So, the trick is to do a direct disk access. That means POKEing the address of the beginning of the file to the operating system so that it will know where to find it. Unfortunately, the op. sys. wouldn't know what to do with that familiar decimal number and therein lies our problem.

As long as the original floppy disk is used, there is no problem as the files are always where the program expects them to be. But try to mount RESEQ to a hard disk, obviously at a new disk address, and it won't work until the program code is changed. In fact, if unchanged, there is a very good chance that you will wipe out whatever file resides on the hard disk at the location that OUTFIL lives on the floppy.

The op. sys. reserves locations to hold the disk address which means that the address must be expressed in the machine's language of bytes. But let's back up for a moment.

If you are a math wizard, there is not too much to it and you probably recognized it as a conversion from base 10 to base 256. For the rest of us, a little more explanation is in order.

The rudimentary language that computers speak is binary or simple "ones" and "zeros". One of these characters, the one or zero, is a "Bit". In short, something either "is" or "is not" - "true" or "false". Because our computers are "8 Bit" machines, they are capable of handling 8 bits at a time. An 8 bit chunk of data is a computer word called a "Byte". Think of it this way. Each of the 48K memory cells can hold one 8 bit byte.

If a bit can only be a zero or one, then, how big can a byte be? The largest decimal number that can be represented in 8 bits is 255. Therefore, any number larger than 255 will require the use of a second byte. If the number is equal to, or larger than, 256^2 (256x256) or 65,536, then a third byte will be required. It is therefore, not too surprising that these three bytes are referred to as Lo Byte, Mid Byte and Hi Byte.

Now, let's get to the business of figuring out the byte address of the new location of our file. Let's suppose that the decimal address of the file from the directory is 64,000. Divide (by the old long-hand method so that you will know what the remainder is) 64,000 by 256. The answer is exactly 250. This number, being less than 256 can be stored directly in the first or Lo Byte. This procedure is good to address (255x256) or 65,280.

If the disk address is higher, say 721,152, then, because it is greater than either 256 or 256^2 (65,536), we divide by the larger number and get an answer of 11 with a remainder of 1. Thus, the Mid Byte is 11 and the Lo Byte is 1. This procedure will cover disk addresses up to 16,711,680.

Those who need to represent disk addresses in excess of this will require the use of a third or Hi Byte to represent the number. Let's suppose that the address is 218,109,696. In this case, we probably will have to divide the number more than once: first by 16,777,216 (256^3) to get the answer of 13 and a remainder of 23. If the remainder was greater than 255, we would have had a Mid Byte, but because it is only 23, it must be in the Lo Byte position. The Mid Byte is a 0 and the Hi Byte is a 13. This procedure will handle addresses up to 4+ Giga Bytes.

Checking your work is easier. Just add up the products of: the Hi Byte times 16,777,216, the Mid Byte times 65,536 and the Lo Byte times 256. The total should equal the original decimal address.

```
Hi Byte *  (256^3) or 16,777,216 = _____
Mid Byte * (256^2) or     65,536 = _____
Lo Byte  * (256^1) or        256 = _____
Total = decimal disk address      _____
```

OS-U PROGRAMMING AIDS PART 1

At last the ice is broken! For years experienced OS-U programmers have carefully guarded those frequently used 'tricks' that stretch the use and flexibility of OS-U. Now Roger is sharing those tips with you in such a way that rank beginners to the experienced programmers can gain by his experiences. We challenge the rest of you to continue this effort to improve the use and quality of OS-U programming.

This month's episode is only the beginning. Future articles will detail such areas as: debugging tools, reserved words, PEEK and POKE lists plus a whole raft of frequently used subroutines: everything from sorts to printing numbers in words.

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

OHIO SCIENTIFIC ERROR MESSAGES

"?REDO FROM START" or "? Not acceptable"

usually means that the computer requires a number to be input. (With FLAG 21 and FLAG 28 on, you get this message if you just hit <RETURN>.)

"?EXTRA IGNORED" or "?Extra ignored"

means you entered more items than the computer expected. Usually you inadvertently entered a comma, and the computer mistook it for the end of the first item.

"??"

means that the computer expected more items than you entered and now wants the others, together or separately.

"BREAK IN 3210"

means that the program has

continued on page 13

RIGHT HAND JUSTIFY
PROPORTIONAL PRINT

By: Earl Morris
3200 Washington Street
Midland, MI 48640

This BASIC program will right hand justify proportional print. Put the text in the low line numbers with a quote sign after the line number. Try to make all lines about the same length then RUN 50000. The program will first scan through the text measuring each line length. The second pass will print the text. If you are not using 65D 3.2 then the value of AD in lines 50170 and 50370 must be changed. AD is the address of the pointer to the start of BASIC text. I have a parallel printer and used PRINT #4. If you have a serial printer use the correct PRINT device for your system. Line 50050 uses the control code to put my printer into proportional mode. Line 50561 uses the control code to move my printer head (FL) dot spacings. Modify these lines to suit your printer.
by Earl Morris

```
10 "This BASIC program will right hand justify proportional
20 "print. Put the text in the low line numbers with a
30 "quote sign after the line number. Try to make all lines
40 "about the same length then RUN 50000. The program will
50 "first scan through the text measuring each line length.
60 "The second pass will print the text. If you are not
70 "using 65D 3.2 then the value of AD in lines 50170 and
80 "50370 must be changed. AD is the address of the
90 "pointer to the start of BASIC text. I have a parallel
100 "printer and used PRINT #4. If you have a serial
110 "printer use the correct PRINT device for your system.
120 "Line 50050 uses the control code to put my printer
130 "into proportional mode. Line 50561 uses the control
140 "code to move my printer head (FL) dot spacings.
150 "Modify these lines to suit your printer.
160 "by Earl Morris
50000 REM RUN 50000 TO START PRINT-OUT
50010 DIMX,LL,AD,BP,LN,I,CN,S,PA,A,A(70),C(122)
50020 REM READ CHARACTER WIDTHS
50030 FOR X=32 TO 122:READ C(X):NEXT
50040 LL=10
50045 REM ENABLE PROPORTIONAL PRINT FOR MY PRINTER
50050 PRINT#4:PRINT#4,CHR$(27)CHR$(17)
50150 :
50160 REM SCAN FOR LONGEST LINE  STORE LENGTH IN LL
50165 :
50170 AD=120:REM START OF BASIC POINTER 65D 3.2
50180 AD=PEEK(AD)+256*PEEK(AD+1):REM ADDRESS NEXT LINE
50190 IF AD=0 THEN PRINT"ERROR":END
50200 BP=AD+4
50210 LN=PEEK(AD+2)+256*PEEK(AD+3):REM GET LINE NUMBER
50220 IF LN>49000 THEN 50370  : REM CHECK IF DONE
50230 IF PEEK(BP)=34 THEN BP=BP+1   :REM SKIP QUOTE SIGN
50240 CN=0:S=0
50250 A=PEEK(BP):REM GET NEXT CHARACTER
50260 IF  A=0 THEN 50300 : REM END OF LINE
50270 CN=CN+C(A)      :REM ADD UP LINE LENGTH
50280 BP=BP+1:GOTO 50250
50300 PRINTCN:IF CN>LL THEN LL=CN   :REM FIND LONGEST LINE
50310 GOTO 50180
50360 :
50370 AD=120:REM SCAN FOR PRINT OUT
50375 :
50380 AD=PEEK(AD)+256*PEEK(AD+1)
50390 IF AD=0 THEN 59000
50400 BP=AD+4
50420 LN=PEEK(AD+2)+256*PEEK(AD+3)
50430 IF LN>49000 THEN 59000
50450 IF PEEK(BP)=34 THEN BP=BP+1
50455 I=0:CN=0:S=0:FL=0
50460 A(I)=PEEK(BP)
50470 IF A(I)=0 THEN 50540   :REM END OF LINE  NOW PRINT IT
50480 IF A(I)=32 THEN S=S+1     :REM COUNT SPACES
50520 CN=CN+C(A(I))       :REM COUNT DOTS
50530 BP=BP+1:I=I+1:GOTO50460
```

Continued on next page

OS-U Programming Aids cont:

stopped running at that line number, because of Control-C or a STOP statement. STOP statements are used for debugging and are sometimes left in to catch unlikely errors. LIST 3210 (or whatever line) and phone your programmer. If you can't reach him, write down the line and RUN"MENU"

"DEV A ERROR 17 IN 3210"

The line number here is less important than the device and disk error number. Error 1 in OS-65U always means disk drive not ready. Errors between 2 and 127 have various meanings depending on drive type. The first three entries below are for floppy disks. See the OS-65U manual for hard disks.

ERROR 1, ERROR 5

disk not in drive, or sideways or upside down. Double-sided disk in single-sided drive. Drive door not closed. Drive not powered up.

ERROR 6

Write-protect notch in disk not covered.

ERRORS 2-4, 7-27

Hardware errors. May be caused by either disk or drive. If on a brand-new disk, then try initializing it once more, and if you still get the error then throw away the disk.

If, on the other hand, this is your working disk, then pray that you have an up-to-date backup. One good scheme is

Continued on next page

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

```
50535 :
50540 REM START PRINT OUT ADDING SPACE TO RIGHT JUSTIFY
50541 :
50542 REM MAKE ALL LINES EQUAL TO LONGEST LINE
50543 IF (LL-CN)>(I-1)THEN FL=1:CN=CN+I-1
50545 IF (LL-CN)>(I-1)THEN FL=2:CN=CN+I-1
50546 PRINT#4,"          ";
50548 IF CN<.7*LL THEN S=0:FL=1:REM SHORT LINE
50550 FOR X=0 TO I-1
50560 PRINT#4,CHR$(A(X));
50561 REM CONTROL CODE TO SKIP DOTS BETWEEN LETTERS
50562 IF FL>0 THEN PRINT#4,CHR$(27)CHR$(FL);
50565 IF A(X)=32 THEN GOSUB 57000  :REM PAD SPACES
50570 NEXT:PRINT#4
50580 GOTO50380
56999 :
57000 REM ADD MORE SPACE BETWEEN WORDS
57001 :
57005 IF S=0 THEN RETURN
57010 PAD=INT((LL-CN)/S)
57020 IF PA<0 THEN RETURN
57022 IF PA>12 THEN PA=12
57030 CN=CN+PA:S=S-1
57035 IF PA>6THENPRINT#4,CHR$(27)CHR$(6);:PA=PA-6
57040 PRINT#4,CHR$(27)CHR$(PAD);
57050 RETURN
57900 :
57910 :REM TABLE FOR CHARACTER WIDTHS
57920 :
58000 DATA7,7,10,15,12,16,14,7    :REM SPACE TO '
58001 DATA7,7,12,12,7,12,7,12     :REM ( TO /
58002 DATA12,12,12,12,12,12,12,12 :REM 0 TO 7
58003 DATA12,12,7,7,12,12,12,12   :REM 8 TO ?
58004 DATA14,16,15,14,16,14,14,16 :REM AT TO G
58005 DATA16,10,14,16,14,18,16,16 :REM H TO O
58006 DATA14,14,15,12,14,16,16,18 :REM P TO W
58007 DATA16,16,10,12,12,12,12,12 :REM X TO UNDERLINE
58008 DATA7,12,12,10,12,12,10,12  :REM GRAVE TO g
58009 DATA12,8,6,12,8,16,12,12    :REM h to o
58010 DATA12,12,10,12,10,12,12,16 :REM p to w
58011 DATA12,12,10                :REM x to z
59000 END
```

★ ★

OS-U Programming Aids cont:

for each working disk to have at least two backups, labeled "Even-numbered days Back-up" and "Odd-numbered days Back-up"; using these consistently should free you from worry.

Normally, the error message will specify the device. First, run COPIER and try to make a backup; if it won't copy from A to B (or C to D), try copying from B to A (or D to C). If the backup is successful, make it your new master disk and throw the old disk into your spare disk pile.

If COPIER won't go past the error, it reports the disk address of the error and returns to its menu. Write down the address, then run DIR and see what file it was on. Find a backup (not your most recent, you may need that) or else a large enough temporary file (such as SCRAT on the utility disk), and put it in device B. Insert a utility disk in A and run COPYFI. When it asks the first question ("FROM DEVICE ?") put your problem disk in A and try to copy the problem file to device B. If it stops at the error, which is likely, switch the disks and try copying from B to A (or D to C, as the case may be).

Occasionally, COPYFI works when COPIER doesn't, but whether successful or not, run COPIER and select "I" for Initialize. It asks whether it should initialize the whole disk; answer "N". It then asks "From address" and "To address"; answer both with the address you wrote down. It then asks whether a 3584-byte range is OK; answer "Y". It will initialize one track and return to its menu. Try again to make a backup; if successful, you have saved the disk except for one track. If COPYFI was successful earlier, or you have some other up-to-date backup, copy the backup onto the problem disk using COPYFI, and you should be back to normal. Otherwise, if the error was on a BASIC file, you probably have another copy of the program somewhere; find it, LOAD it, switch in your problem disk, and SAVE. If it was in a data file, list the file on the screen by the usual menu option; it may be OK, but if the error was early in the file you have probably lost 14 records. Chalk it up to experience, edit the records as best you can, and make a backup every day in future.

If you get errors frequently, your drive heads need cleaning or your drives need alignment or other maintenance.

ERROR 128

File not found. You misspelled the file name, or the wrong disk is in the drive, or the computer is looking at the wrong drive, as commonly happens after an error interrupts the normal flow. In our accounting system, typing DEV"A": RUN"MENU" will usually remedy matters.

ERROR 129

File not open. FLAG 1 is required to keep files open in the immediate mode or after an error.

ERROR 130

Wrong password. Note that a data file can be opened with a wrong password. The error comes when restricted access is attempted.

ERROR 131

Caused by trying to LOAD a data file or OPEN a non-data file, when the file has a password and you didn't give it.

ERROR 132

End of file. Usually caused by trying to input a record that's not there. Try running "FDUMP" and looking at the file. In our accounting system, check the Files Diagram for number or records. Sometimes INPUT statements go wrong because of a POKE to 2976; it normally contains 44, but must be changed to 13 to input a string containing a comma (unless the string is preceded by "). If it isn't changed back again several errors can occur. It is good practice to print " before strings containing a comma, to avoid this problem. If the problem is that you don't know the length of the file, you need FLAG 9 error trapping (see below).

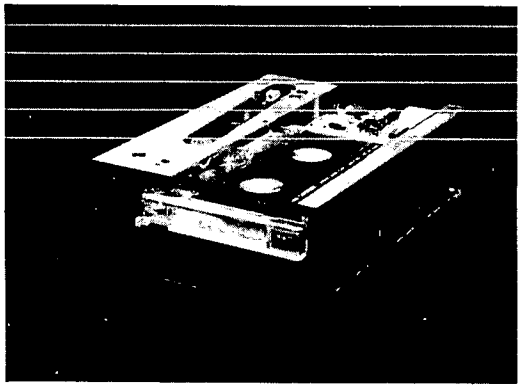
ERROR 133

Can't open file under that

continued on page 19

DBi, 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 5¼" Winchester. It's exactly the same size as a 5¼" 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 5¼" 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

A MODIFIED BUBBLE SORT/MERGE

By: George Belcher, M.D.
Columbus Clinic, P.A.
Columbus, KN 66725

Your list of article topics in the April issue prompts this information. The BUBBLE SORT/MERGE is a section of a medical billing system which I've written for a C3B with V1.2 OSU. The second listing is a Heap Sort Algorithm written by my son, Mark, on a C3OEM with OSU 1.2 which he used at college. I don't propose to discuss them as the REM's document them. Both of these are written such that they are directly workable by plugging in the right data file names.

A long list of data records is sorted by string function. New records are added and this sort/merge program is used to periodically put them in order. All records are fixed length with a 6 digit record number and to help with the "FIND" they start with "*". This is checked in Line 1032-1035-1225-1230. Bad data is printed for operator information. Deleted records are marked with 9's for sorting out in Lines 1030 & 1220.

The OSI short memory problem is handled by frequent sorts and cycling the merge records.

End index of the last sort is stored at INDEX = 0 with EOF at 20. Records start at index = 40.

This BUBBLE SORT/MERGE is relatively slow, but is easy, simple, and functions well for short sorts and this purpose.

★

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

```
10 REM  * * * A MODIFIED BUBBLE SORT / MERGE PROGRAM * * *
20 REM  USED BY G.BELCHER M.D. IN 'MEDICAL BILING PROGRAM'
40 GOTO10000
100 :                                   REM   MODIFIED BUBBLE SORT
110 D=0:M=D:SW=0
120 IFSW=0ANDD<MTHEND=M-1:              REM   JUMP TO END OF SORTED
140 D=D+1:IFD=G THEN RETURN:            REM   END SORT
150 IFA$(D)<=A$(D+1)THENSW=0:GOTO120
160 IFD>MTHENM=D
170 A$=A$(D): A$(D)=A$(D+1): A$(D+1)=A$
180 IFD=1GOTO140
190 D=D-2:SW=1:GOTO140:                 REM   BUBBLE UP TO FIT
1000 :                                  REM   FORM A$() TABLE
1010 G=0:IFE=SGOTO1200
1020 INDEX(1)=S:FORX=1TO200:IFINDEX(1)=>ETHENGOSUB100:GOTO1200
1030 INPUT%1,P1$:IFLEFT$(P1$,7)=DL$GOTO1050
1032 IFLEFT$(P1$,1)<>"*"GOTO1090:       REM   NO MARKER
1035 IFLEN(P1$)<>RLGOTO1080:            REM   WRONG LEN
1040 G=G+1:A$(G)=P1$
1050 NEXTX:STOP:REM  TABLE FULL
1080 PRINT#5,"LEN="LEN(P1$):            REM   ERROR REPORT
1090 PRINT#5,P1$:GOTO1050
1200 :                                  REM   FORM B$() TABLE
1210 X=1:INDEX(1)=40:INDEX(2)=40
1215 Y1=0:FORY=1TO200:IF INDEX(1)=>S GOTO1300
1220 INPUT%1,P1$:IFLEFT$(P1$,7)=DL$GOTO1250
1225 IFLEFT$(P1$,1)<>"*"GOTO1290
1230 IFLEN(P1$)<>RLGOTO1280
1240 Y1=Y1+1:B$(Y1)=P1$
1250 IFY1<100THENNEXTY:GOTO1300
1280 PRINT#5,"LEN"LEN(P1$)
1290 PRINT#5,P1$:NEXTY
1300 :                                  REM   MERGE A$() INTO B$()
1305 FORZ=1TOY1
1307 IF X>G GOTO1320
1310 IFA$(X)<B$(Z)THEN PRINT%2,A$(X):X=X+1:GOTO1307
1320 PRINT%2,B$(Z):IF INDEX(2)=>1E8 THEN STOP
1330 NEXTZ:IFINDEX(1)<SGOTO1215:        REM   GET MORE B$()
1340 IFINDEX(1)=>SANDX<=GTHEN PRINT%2,A$(X):X=X+1:GOTO1340
1400 :                                  REM   PRT NEW FILE LEAD
1410 INDEX<2>=0
1420 FORX=1TO13:PRINT%2,"  ":NEXTX:     REM   CLEAR LEAD SPC
1430 INDEX<2>=0:PRINT%2,E:INDEX<2>=20:PRINT%2,E:RETURN
3000 :                                  REM   CLEAN UP NEW FILE
3010 OPEN"WORK-1",2:B0$=""
3020 FORX=1TO69:B0$=B0$+"0":NEXTX
3030 FORX=1TO3000:PRINT%2,B0$:IFINDEX(2)<1E8THENNEXTX
3040 CLOSE:RETURN
10000 :                                 REM   MAIN PROGRAM
10010 DIM A$(500),B$(150):DL$="*999999":RL=69
10020 GOSUB3000:OPEN"PTFILE",1
10030 INDEX(1)=0:INPUT%1,S:             REM   END PREV.SORT
10040 INDEX(1)=20:INPUT%1,E:            REM   EOF
10050 INDEX(1)=40:FINDDL$,1:            REM   LOOK FOR DELETES
10060 IFINDEX(1)<1E8GOTO10080:          REM   DELETE FOUND
10070 IFS=ETHENCLOSE:END:               REM   NO NEW RECORDS
10080 OPEN"WORK-1",2:GOSUB1000:CLOSE:END
```

```
10 REM                  FUNCTION:  This program is an application of a
20 REM                             heap sort algorythm by Mark A. Belcher.
30 :
200 REM[------          MAIN PROGRAM
210 :
220 PRINT "HEAP SORT ALGORITHM IMPLICATION"
230 PRINT
240 DIM A$(75):                     REM     SET-UP AND INITIALIZATION
250 N=0
260 OPEN "DATA4",1:                 REM     OPEN DATA FILE
270 :
280 PRINT:PRINT
290 PRINT TAB(10)"INPUT DATA"
300 INPUT%1,A$(0):                  REM     INPUT A RECORD FROM THE FILE
310 IF A$(0)="EOF" THEN 600:        REM     IF END OF FILE THEN LEAVE
320 IF A$(0)="EOD" THEN 400:        REM     IF END OF DATA GROUP THEN SORT
330 N=N+1:A$(N)=A$(0):PRINT A$(0):GOTO 300
340 :
400 FOR I=INT(N/2) TO 1 STEP -1:REM     BUILD THE INITIAL HEAP
410 :   A=I:B=N:GOSUB 1000:NEXT I
420 :
440 FOR I=N TO 2 STEP -1:           REM     SORT THE HEAP
450 :   A$(0)=A$(1):A$(1)=A$(I):A$(I)=A$(0)
460 :   A=1:B=I-1:GOSUB 1000
470 :   NEXT I
480 :
485 PRINT TAB(10) "SORTED OUTPUT"
490 FOR I=1 TO N:                   REM     OUTPUT SORTED ARRAY
500 :   PRINT A$(I):NEXT I
510 :
520 N=0:GOTO 280                    REM     INITIALIZE AND GO FOR ANOTHER
530 :
600 REM[------   END OF PROGRAM
610 :
620 CLOSE:PRINT "END OF FILE REACHED  --  NO MORE DATA TO SORT.":END
630 :
1000 REM[\------   HEAPIFY SUBROUTINE
1010 :
1015 K=2*A:                         REM     POINT TO LEFT SON
1020 IF K > B THEN RETURN:          REM     IF SON IS OUT OF BOUND THEN BYE
1030 IF A$(K) > A$(A) THEN 1050
1035 IF A$(K+1) > A$(A) AND K+1 <=B THEN K=K+1:GOTO 1060
1040 RETURN:                        REM     BOTH SONS SMALLER SO BYE
1050 IF A$(K+1) > A$(K) AND K+1 <= B THEN K=K+1
1060 A$(0)=A$(K):A$(K)=A$(A):A$(A)=A$(0):A=K:GOTO1000
```

★ ★

ANOTHER KEYBOARD ALGORITHM

By: Mark Howell
24 Paul Avenue
Wantirna South, 3152
Victoria, Australia

Sometime ago, I started to develop an EPROM replacement for the SYN600 monitor used in the C1 and, amongst other things, wanted to improve on the keyboard decoding used by OSI.

The routine I decided upon was originally written by Rodney Eisfelder (c/o 6502 Users Group, 10 Forbes St., Essendon 3040, Australia) and has been slightly modified to separately decode RUBOUT and CONTROL/RUBOUT. The code, along with one change to the keyboard table, is the same length as the original algorithm and can be easily incorporated into an EPROM.

Some of the main advantages of the new routine are:

1. Shift lock down gives a computer terminal keyboard without generating garbage characters using the shift keys.

2. Shift lock up gives a typewriter style keyboard with special characters (CONTROL/K to CONTROL/P) available via the right shift key.

3. Line feed, Return, Rubout and all Control characters still working correctly with shift lock up or down.

This algorithm uses subroutines at $FC91, $FCBE, $FCC6, $FCCF, $FDC8 and the keyboard table at $FDCF to $FDFF for its operation. The byte at $FDF3 is changed from #$FF to #$DF for rubout key decoding and gives a final value of #$7F or #$1F when used with the control key.

```
%DFD00,FDC7
FD00 8A       TXA
FD01 48       PHA
FD02 98       TYA
FD03 48       PHA
FD04 A901     LDA  ##01
FD06 20BEFC   JSR  $FCBE
FD09 20C6FC   JSR  $FCC6
FD0C D005     BNE  $FD13
FD0E 0A       ASL  A
FD0F D0F5     BNE  $FD06
FD11 F040     BEQ  $FD53
FD13 4A       LSR  A
FD14 9009     BCC  $FD1F
FD16 2A       ROL  A
FD17 E021     CPX  ##21
FD19 D0F3     BNE  $FD0E
FD1B A91B     LDA  ##1B
FD1D D023     BNE  $FD42
FD1F 20C8FD   JSR  $FDC8
FD22 98       TYA
FD23 8D1302   STA  $0213
FD26 0A       ASL  A
FD27 0A       ASL  A
FD28 0A       ASL  A
FD29 38       SEC
FD2A ED1302   SBC  $0213
FD2D 8D1302   STA  $0213
FD30 8A       TXA
FD31 4A       LSR  A
FD32 20C8FD   JSR  $FDC8
FD35 D01C     BNE  $FD53
FD37 18       CLC
FD38 98       TYA
FD39 6D1302   ADC  $0213
FD3C A8       TAY
FD3D B9CFFD   LDA  $FDCF,Y
FD40 297F     AND  ##7F
FD42 CD1502   CMP  $0215
FD45 D011     BNE  $FD58
FD47 CE1402   DEC  $0214
FD4A F016     BEQ  $FD62
FD4C A204     LDX  ##04
FD4E 2091FC   JSR  $FC91
FD51 F0B1     BEQ  $FD04
FD53 A900     LDA  ##00
FD55 8D1602   STA  $0216
FD58 8D1502   STA  $0215
FD5B A902     LDA  ##02
FD5D 8D1402   STA  $0214
FD60 D0A2     BNE  $FD04
FD62 A296     LDX  ##96
FD64 CD1602   CMP  $0216
FD67 D002     BNE  $FD6B
FD69 A214     LDX  ##14
FD6B 8E1402   STX  $0214
FD6E 8D1602   STA  $0216
FD71 A901     LDA  ##01
FD73 20BEFC   JSR  $FCBE
FD76 20C6FC   JSR  $FCC6
FD79 AD1502   LDA  $0215
FD7C C921     CMP  ##21
FD7E 9029     BCC  $FDA9
FD80 2940     AND  ##40
FD82 F029     BEQ  $FDAD
FD84 8A       TXA
FD85 2940     AND  ##40
FD87 D02F     BNE  $FDB8
FD89 AD1502   LDA  $0215
FD8C C95F     CMP  ##5F
FD8E F025     BEQ  $FDB5
FD90 8A       TXA
FD91 2907     AND  ##07
FD93 F020     BEQ  $FDB5
FD95 AE1502   LDX  $0215
FD98 E051     CPX  ##51
FD9A B00D     BCS  $FDA9
FD9C E04B     CPX  ##4B
FD9E 9009     BCC  $FDA9
FDA0 C902     CMP  ##02
FDA2 F00E     BEQ  $FDB2
FDA4 4A       LSR  A
FDA5 F002     BEQ  $FDA9
FDA7 B009     BCS  $FDB2
FDA9 A900     LDA  ##00
FDAB F00D     BEQ  $FDBA
FDAD 8A       TXA
FDAE 2906     AND  ##06
FDB0 F0F7     BEQ  $FDA9
FDB2 A910     LDA  ##10
FDB4 2CA920   BIT  $20A9
FDB7 2CA940   BIT  $40A9
FDBA 4D1502   EOR  $0215
FDBD 8D1302   STA  $0213
FDC0 68       PLA
FDC1 A8       TAY
FDC2 68       PLA
FDC3 AA       TAX
FDC4 AD1302   LDA  $0213
FDC7 60       RTS
```

RESMON.III

RESMON.III is a variation of the DABUG III ROM with added features. The single key entry command was deleted to make room for:

(CONTROL)W - RESETS STACK AND EXITS TO $0000
(CONTROL)E - RESETS $0218 TO $0221 AND EXITS IN 24X24 MODE TO $FE00
(CONTROL)P - RESETS ACIA TO FULL SPEED AND SETS PRINTER FLAG TO 1
(CONTROL)T - RESETS ACIA TO TAPE SPEED AND SETS PRINTER FLAG TO 0
(CONTROL)V - TAPE VIEW ROUTINE WITH (SPACE BAR) TO EXIT

The existing DABUG III commands are:

(CONTROL)Q - ACTIVATES EDITOR CURSOR ETC.
(CONTROL)U - MOVES EDIT CURSOR UP
(CONTROL)D - MOVES EDIT CURSOR DOWN
(CONTROL)L - MOVES EDIT CURSOR BACKWARDS
(CONTROL)R - MOVES EDIT CURSOR FORWARDS
(CONTROL)A - FORWARD ENTRY KEY
(CONTROL)B - SWITCHES BETWEEN 24X24 AND 48X12 SCREEN FORMATS
(SHIFT)O - DESTRUCTIVE BACK-SPACE
(SHIFT)P - CANCEL LINE ENTRY
(RUBOUT) - CLEAR SCREEN
THE SCREEN FREEZE ROUTINE HAS ALSO BEEN RETAINED.

Other Features of RESMON.III are:

A new error code correction routine has been added. This corrects in 24X24 and 48X12 screen formats.

The complete character set can be printed in 48X12 mode.

The restart sequence at $FF00 prints a new menu- D/C/W/M/1/2 ? It also loads #$00 into $0217(PRINTER FLAG).

The output routine at $FF69 does a printer flag check. If this flag is set a JSR to $020F is executed. A user supplied JMP is now required at $020F or a RTS instruction.

Two other user supplied JMP's are possible. On restart: Typing 1 does a JMP to $012A Typing 2 does a JMP to $012D

The OSI 65V monitor was altered to fix a small bug. You can no longer write to non existent memory or to ROM!

Continued on next page.

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

A new keyboard algorithm at $FDØØ has been included.

References: DABUG III Manual by David Anear, A New Keyboard Algorithm by Rodney Eisfelder, KAOS Newsletter, Volume 3, Nos. 8 and 9. On Error GOTO by Earl Morris and Kerry Lourash, MICRO No. 51, also DABUG III J by John Whitehead, KAOS Newsletter, Volume 4 No. 1.

★

PROBLEM-SOLVING VS APPLICATION SOFTWARE

TWO DIFFERENT APPROACHES TO COMPUTER EDUCATION

"There is no longer a question of whether to teach computer use. The question is what and how to teach the use of the computer as a tool for everyday personal and business use."

By: Roy Agee

Microcomputers are gaining widespread acceptance in the workplace, home and school. Computers will change the way instructors teach and think according to a recent university study. The development of the microcomputer, which has brought computer power to nearly everyone, is changing the nature of everyday life. The computer has also provided the means for meeting this challenge of change. Strong demands have been placed on the educational establishment to prepare the youth of today to master the challenge and technology of tomorrow. There is no longer a question of whether to teach computer use. The question is what and how to teach the use of the computer as a tool for everyday personal and business use.

There is a growing belief among educators that the fragmented approach to computer studies must be brought under control. Obstacles to that include inadequate teacher preparation/training, a lack of direction, and availability of materials. One teacher uses a software package that seems appropriate, another uses a different package and approach. Students find the only similarity in computer classes is the computers. What is taught in one class does not relate to another class. This lack of continuity leads more to confusion than clarification.

TWO DIRECTIONS

In recent months, two distinctly different approaches have emerged for computer studies classes.

Application: This direction states that a knowledge of how to use specific application software is all that is required to master the use of the computer. This approach is generally advanced for business and/or computer literacy students and it is flawed. One glaring flaw is that such an approach severely limits the students' use of the computer. Many of the specific software applications currently used in schools (i.e., spreadsheet and data base) are management tools, not used by entry level employees.

In the case of word processing, the issue is more complex. A recent study reported that underutilization 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 a screen. Word processing, filing, mail label software relieve the secretary of many time consuming repetitive tasks and allow for taking on additional responsibilities. They are, however, severely limited without professional training in the use of the computer as a "problem-solving tool." Of course, word processors, without basic secretarial skills, is of little value to an employer.

Those who follow this "application" direction argue that not all of their students will seek computer-related employment. This short-sighted vision misses the point, which is that the computer is a tool and worthy of being taught as a problem-solving tool. Few students of driver education will become professional drivers. Most will use an automobile for personal use. Does that mean they are only taught to use the power windows and air-conditioning, or do they actually learn to drive the car? For the same reason, education must not limit computer studies to limited application software. Instead a program which provides a mastery of the thinking and problem solving skills required to use the computer must be offered.

Problem Solving: The second direction to computer studies offers many benefits to educators and to the persons they teach. A major advantage is that such an approach teaches and enhances thinking and problem solving skills, both for use with the computer and in other aspects of life. With a properly developed and sequenced curriculum, students gain the skills and knowledge to use the computer as a problem-solving tool! The capabilities acquired from this method will be used throughout their life. This is in sharp contrast with specific applications training which is limiting and soon becomes obsolete.

Using the problem-solving approach in a logical step-by-step manner, students compile a catalog of generic skills. This structure and methodology would feature such fundamental concepts as sequential and random access files, online and batch processing, ARRAYS (for spreadsheet) and data bases, program/systems design and analysis. Task problems for learning these skills should include such practical, real-life applications as accounts receivable, inventory, billing, personnel records, etc.. This will demonstrate entry level job functions, as well as prepare for further studies in computer science. In essence, students should be able to solve virtually any problem, on nearly any type of computer.

Conservative employment projections indicate that within 15 years, over 80% of the workforce will be involved in the information industry. There are still a few "buggy whip" makers around (some in education) who believe "the computer fad" will pass. It

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

won't! The Industrial Age is being replaced with a new era referred to as the Information Age. This new era will require individuals who possess innovative creative thinking and problem-solving skills. The microcomputer is a dynamic tool for teaching these skills, and its mastery as a problem-solving tool is a requirement. The computer is too valuable to be used as a typewriter with a screen!

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

★

OS-U Programming Aids cont:

channel number (1-8) because the channel is already in use.

BS ERROR

Bad subscript: index of array outside DIM range. Also caused (twice) by INDEX<9>=-1. Sometimes caused by a file being empty, causing DIM A(Ø), then trying to start at record #1.

CN ERROR

Can't continue (in response to CONT) because program was changed, or the Editor was used, or an error occurred. You can still continue by GOTO, but except in the last case, you have lost your variable values.

DD ERROR

Double dimension: array dimensioned twice. Usually caused by referring to an array (causing a default dimension of 1Ø) before the DIM statement.

FC ERROR

Function call. A number is outside the range the function or operator can handle. The following ranges are permissible:

```
1 to 8          : OPEN, CLOSE, INPUT%, PRINT%, INDEX, FIND
Ø to 21Ø        : WAIT FOR, WAIT CLEAR
Ø to 254        : INPUT[ ]
Ø to 255        : CHR$, LEFT$, MID$ (3rd arg), RIGHT$, TAB, SPC,
                : PRINT#, PRINT[ ], PRINT&, NULL, PEEK (2nd arg),
                : POKE (2nd arg), WAIT (2nd & 3rd args), FLAG, ON
1 to 255        : MID$ (2nd arg), RSEQ (3rd arg)
Ø to 32767      : DIM, A( )
-32768 to 32767: AND, OR, NOT, A%=
Ø to 63999      : NEW, RSEQ (1st & 2nd args)
Ø to 65535      : PEEK, POKE, WAIT (all 1st arg), GOTO, GOSUB, RUN
> Ø            : LOG
>=Ø            : SQR
"A" to "H"      : DEV   ("A" to "Z" in Level 3)
```

The error is also caused by A^B, where A is negative and B is not an integer, by calling USR(X) before the correct address has been POKEd in, by RSEQ NLN,OLN,INC if OLN > NLN or if the last new line number would be > 63999, and by WAIT CLEAR X when the semaphore X is not set. If necessary one can define DEF FNSM(X) = -((PEEK(55333+X/8) AND 2^(X AND 7))=Ø) AND THEN CHECK FIRST: IF FNSM(X) THEN WAIT CLEAR X

FS ERROR

Full stack. The stack will hold 26 GOSUBs or 11 FOR ... NEXT loops. The error may occur during evaluation of a complicated formula as that also uses the stack, but the usual cause is repeatedly failing to return from a subroutine. If you are using

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

BTI 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

a subroutine that calls itself (Quicksort is the best known), your only recourse is to replace GOSUB by GOTO and keep the count yourself.

LS ERROR

Long string. Usually, the file name had more than 6 letters. Also caused by trying to assemble a string longer than 255 bytes. With INP$ enabled, PRINT [LN,"R"] A$ and PRINT [LN,"L"] A$ give LS ERROR if A$ is longer than the length LN.

NF ERROR

Next without FOR. May be programming error, usually caused by having more than one NEXT in a loop; all but the last NEXT must be followed on the same line by a GOTO (out of loop) to prevent this error. Is also caused by, for example, jumping out of an I-loop, starting a J-loop, and then starting another I-loop. BASIC assumes you're restarting the I-loop, and throws out the J-loop too, causing an error with the NEXT J. This can be prevented by cleaning the first loop off the stack by FOR I=Ø to Ø: NEXT. Note that returning from a subroutine cleans out all the loops started by the subroutine. If you interrrupt a program, list some lines, and wish to CONTinue, don't interrupt the listing with a Control-C, as it can cause a later NF ERROR.

NR ERROR

Printer not ready. The message appears only in response to Control-C; any other key forces a retry.

OD ERROR

Out of Data. More READs than DATA items. Check that PEEK(2976)=44. Also note that a DATA statement must be first on a line. This error sometimes results from having two or more lists of DATA and changing the length of one of them. You can avoid this problem by searching for the data you need: RESTORE: FOR I=1 TO 1000: READ X$: IF X$<>"January" THEN NEXT.

OM ERROR

Out of memory. This is sometimes a hardware problem; if you think you really shouldn't be out of memory, save your program if you've been working on it, and type NEW, then PRINT FRE(X). The standard OS-65U system should give 24572, or 23547 with COMKIL enabled. If the answer is correct, then your program has to be cut. First save it into INFILE on a D.P.M. Utility disk, then run RESEQ and delete comments and/or spaces. If caused by large arrays, you may be able to make them integer arrays, taking 60% less room. With multi-dimensional arrays, using the Øth elements saves a surprising amount: DIM A(1,3,7) uses less than half as much space as DIM A(2,4,8). There are often hundreds of unnecessary semicolons in PRINT statements. Combining lines saves four bytes per line (regardless of line number). The final solution is to split the program in two, using COMKIL if necessary.

OS-65U uses OM ERROR in response to a SAVE command if the disk file is too small for the program. You should always have a large scratch file available for such emergencies.

OV ERROR

Number >= 170,141,183,460,000,000,000,000,000,000,000,000,000,000,000,000. (Approximately 1.7E38). Entering 1E99 in response to an input request is often useful for breaking into programs. The EXP (or ANTILOG) function gives OV ERROR over 88.0296919. FLAG 30 gives OV ERROR over 4,294,967,295, the largest number storable without loss of accuracy. There is no underflow error in BASIC: a number smaller than 2.9387358754E-39 is stored as zero. The RSEQ command uses OV ERROR to mean that there is not enough room to make its line number tables.

RG ERROR

RETURN without GOSUB. Usually caused by "falling through" into an unintended subroutine; check the preceding lines for a missing END or GOTO, or run again under FLAG 7.

SN ERROR

Syntax error. This covers a wide variety of errors:

Unmatched number of parentheses (brackets).

An illegal variable name containing a reserved word, particularly ON, OR and TO.

Misspelled reserved word.

Incorrect punctuation.

Line number > 63999, or a number directly following line number.

READing a number when the next DATA item is a string.

Integer or subscripted variable in a loop, e.g. FOR I%=1 TO 10 INPUT or DEF in the direct mode, without a line number.

NEW followed by anything except a carriage return.

SQR, LOG, EXP, ^, RND, SIN, COS, TAN or ATN with INP$ enabled.

DEF or FN with COMKIL enabled.

! without the Editor enabled.

KILL, RSEQ, SWAP or PNTR without COMKIL, RSEQ, etc. enabled.

NULL if any replacement is enabled. (Use POKE 21,X instead.)

LIST if access is restricted and you didn't give the password.

A POKE into the reserved word list, (e.g. POKE 9057,1 for LIST).

The more puzzling syntax errors are generally caused by BASIC's simple minded routine for recognizing reserved words. For example, X=T AND 127 will give a syntax error because BASIC sees the word TAN there (ignoring spaces as always). A way to check for this kind of thing is to reenter the line with a space after every character, then list it; BASIC will remove spaces from the words it recognizes. A little rearrangement or insertion of parentheses should then fix the problem.

SS ERROR

Semaphore stack overflow. In level 3 time-sharing, a maximum of 16 files or other resources can be locked by each user.

ST ERROR

There are 3 string temporaries used to point at temporary strings and literals. It is barely possible to overload them, as in 75 SS$="S.S.#"+(" "+X$+("-"+Y$+("-"+Z$))). The error can also be caused by uncompleted comparisons, as in 75 IF "A" THEN 75, which gives ST ERROR after three loops.

TM ERROR

Type mismatch. Number found

where string required, or vice versa.

UF ERROR

Undefined function. In OS-65U this is misprinted as NF ERROR.

US ERROR

Undefined statement -- no such line number. List the program; sometimes a disk error loses part of it. If caused by a statement in PGM1, 5010 RUN "PGM2",63000 and there is no line 63000 in PGM2, you will get the message "?US ERROR IN 5010", but the program in memory will be PGM2, not containing line 5010, which may be puzzling. This error also appears when a different error occurs under FLAG 9 or FLAG 23 and there is no line 50000 in the program. A very fast check for bad line numbers is made by the RSEQ command.

/0 ERROR

Division by zero. You need an extra line to catch zeros and bypass the calculation. Also caused by TAN(PI/2) if you specify PI to ten digits.

No Error Given

A POKE statement cannot contain a PEEK from a different location. The POKE is not made, but no message is given and the program continues.

OS-65U ERROR TRAPPING

Trapping of disk (numeric) errors is enabled by FLAG 9, which send the errors to line 50000 until disabled by FLAG 10. (If line 50000 is missing, a US ERROR results.)

If the error trapping is confined to one place in the program, the routine at 50000 can be very simple. For example:

```
40  INPUT"PASSWORD";R$
50 FLAG 9: OPEN"#PR",R$,1: INPUT %1,X$: FLAG 10
...
50000 FLAG 10: RUN"MENU": REM Exit on wrong password
```

A more complicated example is a read-write test which needs to check for end-of-file (ERROR 132) on both operations:

```
100 DEV DV$: FLAG 9: INDEX<1>=0:PRINT"WRITING ..."
110 FOR I=1 TO 1E4: PRINT%1,TEST$: NEXT
120 INDEX<1>=0: PRINT"READING ..."
130 FOR I=1 TO 1E4: INPUT%1,X$: IF X$=TEST$ THEN NEXT
...
50000 ERR=PEEK(10226): LN=PEEK(11774)+256*PEEK(11775)
50010 IF ERR=132 AND LN=110 GOTO 120
50020 IF ERR=132 THEN PRINT"TEST COMPLETE": GOTO 170
50030 PRINT"DISK ERROR"ERR"AT INDEX"INDEX(1): GOTO 160
```

The only disk errors commonly worth trapping are 1 (drive not ready), 128 (file not found), 130 (wrong password), and 132 (end of file).

BASIC language errors (two-letter codes) can be similarly trapped by FLAG 23, which is turned off by FLAG 24. If your application can give OV ERROR (number too large), that is probably worth trapping. The other BASIC errors should normally be fixed by changing the program. The error code can be obtained as follows:

```
50000 X=PEEK(18176): if X=23 GOTO 50100: REM Numeric error code

50010   ERR$=CHR$(PEEK(X+867) AND 127)+CHR$(PEEK(X+868) AND 127)
```

It is important to turn off error-trapping as soon as possible, and in any event by the end of the program or before entering the direct mode, otherwise subsequent programs will give "US ERROR" instead of the correct error code.

There is nothing to stop you making up your own error codes for bad data etc., as in the larger versions of BASIC:

```
8330 IF NAME$$="" THEN ERR=201: LN=8330: GOSUB 50000
```

The error-handing routine could print an error message, or store the bad record in an error file for printing after the good data.

MORE NEXT MONTH!

LETTERS

ED:

1. I have a Superboard II Rev D 1980 and a 610 board. The Sams Servicing Data which I have is dated 1979. The pictorials in this manual do not agree with my Superboard. Is there a later issue of the Sams Manual which agrees with the hardware?

2. Basically, what is covered in the OSI Small Systems Journals? I'm trying to determine if they would be of value to me.

3. The last dealer where I bought my 610 board was Cleveland Consumer Computer Components, Cleveland, OH. My letter recently to them was returned. Do you know an OSI dealer that is near this area?

4. Do you know a source for the connectors that fit J-3 on the 610 board and J-2 on the Superboard?

5. I am in the process of adapting a TEAC 55B disk drive to operate with my computer. The article by Joe Ennis in the April PEEK is quite helpful. This is the first time I have ever attempted any of the modifications which have been published in PEEK. But if I am to make the disk drive work, I am going to have to.

6. I would like to make one comment that I have observed. Some of the diagrams printed in PEEK are not too clear (legible), i.e., the Lines and IDs are rather dim. Otherwise, I enjoy reading the articles.

Robert L. Dingle
Dayton, OH 45429

Robert:

1. Regrettably, Sams is what it is, even so, much is still of value. If you are in a bind, call OSI Tech Support (216) 562-2020.

2. The SSJ is some 95 pages summarizing SSJ issues from July 77 through April 78 (when it appeared in Kilobaud Microcomputing). The index contains 110 references to articles, notes, bugs, fixes, games, ASM, mem tests, theory, track 0 writer, etc. They are available only from PEEK(65) for $15.00.

3. CCCC is no more, but as it was a part of OSI, functions returned to Isotron, Aurora.

Again, call Tech Support and ask for Bill Thompson, or call Isotron (203) 255-7443 for the dealer nearest you.

4. Connectors: not specifically, but there are a number of mail order houses like Jameco, and certainly Dayton must have a radio/electric parts house somewhere.

5. Good luck with the TEAC. Let us know how you fare. Others will be interested in your experience.

6. Re printing text is one thing and redrawing schematics is another. We have redrawn too many (you know the hours it takes). So this is a good time to implore those of you submitting drawings to make them clear, black on white and no smaller than a publishable size (blow - ups get fuzzy), taking into account that it will be reduced 30% during the process of printing PEEK.

Eddie

* * * * *

A listing in standard mode.

```
100 DV=1:REM Change DV to whatever device the printer is assigned.
110 PRINT:PRINTTAB(12)"Boldface print switch for EPSON printers.":PRINT
120 INPUT"Do you want to Enable or Disable the boldface modes";A$
130 IF LEFT$(A$,1)="E" THEN GOTO 200
140 IF LEFT$(A$,1)="D" THEN GOTO 250
150 GOTO 100
170 :
180 :
190 REM Transmit appropriate enable codes & suppress <CR>.
200 PRINT#DV,CHR$(27);CHR$(71);:REM enable the double strike mode.
210 PRINT#DV,CHR$(27);CHR$(69);:REM enable the emphasized mode.
220 GOTO 999
230 :
240 REM Transmit the appropriate disable codes & suppress <CR>.
250 PRINT#DV,CHR$(27);CHR$(72);:REM disable the double strike mode.
260 PRINT#DV,CHR$(27);CHR$(70);:REM disable the emphasized mode.
999 END
```

And now a listing in boldface mode.

```
100 DV=1:REM Change DV to whatever device the printer is assigned.
110 PRINT:PRINTTAB(12)"Boldface print switch for EPSON printers.":PRINT
120 INPUT"Do you want to Enable or Disable the boldface modes";A$
130 IF LEFT$(A$,1)="E" THEN GOTO 200
140 IF LEFT$(A$,1)="D" THEN GOTO 250
150 GOTO 100
170 :
180 :
190 REM Transmit appropriate enable codes & suppress <CR>.
200 PRINT#DV,CHR$(27);CHR$(71);:REM enable the double strike mode.
210 PRINT#DV,CHR$(27);CHR$(69);:REM enable the emphasized mode.
220 GOTO 999
230 :
240 REM Transmit the appropriate disable codes & suppress <CR>.
250 PRINT#DV,CHR$(27);CHR$(72);:REM disable the double strike mode.
260 PRINT#DV,CHR$(27);CHR$(70);:REM disable the emphasized mode.
999 END
```

This was reduced 0.75 before printing.

ED:

Thank you for the personal "call for papers." I hope to be able to get time to write several hardware related articles in the coming months. It is actually your writers guidelines which spurred me to write. Your guidelines say to be sure to use a fresh ribbon when generating a listing of a program, but I think that more should be done if a dot matrix printer is being used.

Even with a fresh ribbon, most listings from a DMP look very "spotty" and hard to read after the photocopy process and this gets worse if the copy is reduced. Since most DMP's have a boldface mode of some type, this is an easy problem to correct. If you'll look at the two sample listings, you'll see just how readable a listing can be. While I didn't use a new ribbon on either listing, I think that you'll agree that the boldface version will stand up to photocopying better than most of the listings that you receive.

Of course, the sample listings are of the routine which I used with my EPSON MX-80 to generate the boldface print. I kept the routine as simple as possible (no print formatting, allowances for null input, etc.) to allow for as wide a range of OSI machines as possible. If someone has a different printer, they should consult their printer manual to determine the appropriate control codes required to accomplish similar modes of operation and adapt this routine to use them. I hope that the contributors to PEEK will utilize the boldface modes of their printers in their future articles.

Ray Hackney
Dallas, TX 75253

Readers:

Yes, please!

Eddie

* *

ED:

My expanded system (SBII, 32k, dual floppies) is up and running (V3.2 and V3.3). I have mainly PEEK(65) and its readers to thank for that. Also, Daryl Blair of MPI was very helpful in getting my B drive going - I recommend contacting him if your MPI drives are acting up.

I am now thinking of modem usage and note that CompuServe, at least relative to the Radio Shack Model 100, will only accept one stop bit at 300 baud. Yet, via telephone, OSI has told me that my sigs contain 2 stop bits. How are OSI users addressing CompuServe? Where can I find a detailed discussion of OSI word structure?

Lastly, my 600 board has 2 wires (of a few CM length each) on the underside of the board. I suggest placing tape under those wires to prevent shorting out of nearby components.

Paul Harris
Morristown, NJ 07960

Paul:

The answer is OSI systems can alter the word length and stop bit settings via software control. In most cases, the 8th bit is masked off anyway so the problem sorta goes away. If you want the real nitty-gritty of this, I recommend the data sheet on the M68850 available from Motorola.

Rick Trethewey, Sys Operator
OSI SIG on CompuServe

* * * * *

ED:

Below are two tips I have for users of The Data System.

Printing Labels: Instead of building a key of zip codes and accessing the name/address master file by means of a key file, rebuild your key file to

include all the fields you wish to put on a label. You then sort the key on zip, and print labels directly from the key file.

Search for duplicates before entry: build a key file on the field of interest. If you go into a field, key in ESC 7 to search in a particular field. TDS will search any key containing this field, which is faster than searching the master file. Even faster is this .....instead of doing a regular search "S", key in Sn, where n is the number of the key file containing the field of interest. TDS searches that file instead of the master file, -- again, much quicker.

Tom McGourin
Ft. Wayne, IN 46818

* * * * *

ED:

Here's a real good one for all of your hacker type readers: I recently found a "good deal" on two double sided disk drives, and for $85.00 a pair (count that two drives), I just couldn't pass it up. Well, a few weeks later a parcel arrived via UPS, and inside were two brand new, unused CANNON double-sided 2/3 height (yes, 2/3) single or double density drives. By this time I was quite anxious to quadruple my on-line storage capacity. I hooked both of the new drives up to my disk controller, hit the power switch on the external power strip, and booted up OS-65D. That part completed, I moved on to the rest of my testing by successfully switching from drive A to drive D. The next step involved using OSI's copy routine to make copies of some of my disks, and here is where the problem started. Yes, there is a fly in the ointment! The copy routine would hang up after the first track was copied, and then return an Error #9 for the source disk. Further inspection revealed that the source track had indeed been scrambled, and was no longer readable. Luckily, my source disk was itself a copy, so no real damage was done, but the strange part is that it was a write-protected disk! Several phone calls and a visit to the local floppy repair shop later I am still at a loss as to why I am having this problem. I can say that if I use my original Tandon 100-1 as drive A, and one of the Cannons as drive B and D, I can make as many copies of a disk as I have blanks with no problems.

One thing that I did discover about the Cannon drives is that side one of each drive has a double stepmode. This means that the drive head steps twice for each step pulse that the controller sends out, thereby reducing the disk space from 40 to 20 tracks. I found this out by trying to find out why the green activity LED would not light up when alternate sides of the drive were selected. It seems that Cannon has manufactured these drives to conform to a particular computer's disk controller, while also maintaining some form of compatibility with a "standard" interface scheme. Oh, what a tangled web we weave..!

So there you have it. My Tandon drive is still my workhorse, one new drive is serving as drives B and D, while its companion sits in its box on my workbench, waiting for me to solve this puzzle.

C. J. Hipsher
Virginia Beach, VA 23456

Faces for Familiar Names or East Meets West
David Tasker (L) on recent visit from Australia with Earl Morris at a well known location in Ohio.

AD$

WANTED: Documentation for the Intelligent Terminal Program written by Rodney Trugman. Please contact J. W. Taylor, 3992 No. Juniper Lane, Eden, UT 84310.

* * * * *

FOR SALE: C28P, light use, software. Make offer. (703) 642-0818.

* * * * *

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

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

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

* * * * *

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

* * * * *

48K C4P Dual Floppy for sale. The system includes the following software: WP6502 word processor with disk operating system enhancements, Planner Plus spread sheet with graphics, Term+ and several other terminal editors, full database management system, home device control software, music generator software, Plot BASIC and more. Best offer! Carl M. Good, 560 Longley Rd., Groton, MA 01450, (617) 448-2563.

*** * * * * 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!

PEEK (65)
The Unofficial OSI Users Journal

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

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

DELIVER TO:

GOODIES for OSI Users!

PEEK (65)
The Unofficial OSI Users Journal

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

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

( ) Cash enclosed ( ) Master Charge ( ) VISA

Account No. _____ Expiration Date _____

Signature _____

Name _____

Street _____

City _____ State _____ Zip _____

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

POSTAGE MAY VARY FOR OVERSEAS