SEPTEMBER 1985
VOL.6, NO. 9

PEEK (65)

The Unofficial OSI Users Journal

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

INSIDE

Column One

Lest I forget, I'll begin by announcing PEEK's upcoming software listing. New subscribers have been calling and writing regularly wondering where they can find software for their machines. For the past several years, PEEK has given authors a chance to list their software absolutely free of charge. Well, we are going to do it again. The demand is there and it all sounds just great, but it won't be unless you support us by filling in the form on page 23 ASAP. Just because you sent in a listing last year, don't think that it will automatically reappear! Send in a form!

Here's a second piece of good news. PEEK's half price specials have been extended - "a little." In our efforts to make sure that there would always be a supply of back issues and other manuals, we must have gone overboard. Now we are paying for it in the lack of storage room. It doesn't take a computer to figure out how much room 5.5 years will take up! Besides, this is a golden opportunity to fill out your library.

Another piece of good news. Thanks to the efforts of Leo Jankowski, there is, at last, a single disk copier for OSU. That should be music to the ears of all the newer OSI machine users - you know the ones with only one floppy disk drive. In the past you have either had to forego copying disks or devise tricks with the hard disk to get a copy. Leo's disk is now added to the PEEK "stable" and is available for only $25.50, which includes domestic postage and handling. We will have more details next month, but do feel free to call or write us.

Even more good news. We are told that the latest update version of The Data System (TDS), the "U" based DBMS produced by Gander Software, Ltd. and sold by them and ISOTRON, is in final testing and expected to be released very soon. Improvements include: editing of Defined File editors that may now be 3 pages deep, stored label formats, stored calculations and postings including math and logic rules, plus vastly spedup labels and reports. The Program Generator has also been upgraded to literally write BASIC code to execute programs stored in TDS jobs - and it is even faster. More next month, when it is officially out.

Now to the issue at hand. For the last several months, Roy Agee has been trying to tweak your conscience. Let us hear your thoughts and counter thoughts. On a more specific plain, Bryson's NETAL (circuit analyzer) and Johansen's Data Recorder are guaranteed to make you read carefully, the latter giving some good practical insight into BETA/65.

Programmer's heaven can be found in Rick Trethewey's latest effort to help laymen get better use of the USR function with OSU and Roger Clegg's contribution on sort programs, round out the "U" world.

For the "D" programmer, Leo Jankowski is at it again with more on the sequential file and a new trick with Rick Trethewey's HOOKS. What a job these two have done and what a shame they are half a world apart!

Then there is one to make everyone feel good and at the same time offer a challenging contest. It's Joseph Ennis's OSI Beats IBM PC.

Ed Richardson puts the cap on head loading by addressing twin drives of the 5" variety.

That leaves us with what readers are requesting and hence our request for your articles. BUSINESS RELATED ARTICLES. What we have printed in the past has been great, but we need more. Whether it be a narrative on how you got to where you are, some useful routines that you have developed or need, or a description of your unique application, just jot it down and send it to us. Here are two other areas readers want covered. BSR control continues to be an area that is little understood, and thus is rarely used. If only users knew more about the power that lurks under the hood! Although covered in this issue, USR functions continue to be an area of interest, (particularly by those who don't understand them very well, but yearn for the speed and power they hold).

Once again, don't forget to get your software listings in to us, fast. Not many things are free in this world!

Eddie

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

PART II

By: Rick Trethewey
8 Duran Court
Pacifica, CA 94044

As you begin to program in machine code for OS-65U, you quickly become aware of how completely BASIC has been merged into the operating system. It is not at all like OS-65D in this respect. Therefore, you must respect BASIC's uses of memory when you write your machine code programs. The primary concerns that arise are in the use of page zero (i.e. memory addresses $ØØ through $FF). While the top 32 to 48 bytes of page zero are untouched by BASIC, this is often not enough for major applications. The easiest solution to this problem is to copy the contents of page zero into a buffer as soon as your machine code program is executed, and then restore page zero when your program has finished. This technique costs only a small overhead in memory and will insure that all will be well when control is returned to BASIC. By the same token, there are times when leaving BASIC alone has advantages.

The nice thing about having BASIC remain in memory is that it allows you to use several of its functions for your own routines. For example, if you use BASIC's text output routine at $ØAEE, you can route the output to either the console or to a printer by setting the output flag at 11686. This further allows you to check the position of the cursor, because BASIC counts the number of characters printed on the current line and saves the count in location $16, allowing you to set up columns in your output. Another very useful tool built into BASIC is the ability to pass data between BASIC and your machine code programs. Without modifications, the range of values that can be passed between BASIC and your machine code programs is -32768 and +32767. This is because of the format in which the values are passed, namely 16-bit signed integers in two's compliment held in two 8-bit bytes. Books that teach 6502 Assembly Language programming will discuss two's compliment math. For this discussion, we need only deal with positive values for now.

In a program that will want to pass values, our BASIC program will contain a line of code much like;

X=USR(Y)

where "X" is to be given the result of the operation of the function "USR" on the value "Y". The following program is a trivial application, yet it demonstrates the essentials required to pass parameters between BASIC and machine code routines. Consider the following;

```
10; BUMP IT ONE
20;
30 GETINT = $0006  MAKE F.P. ACCUM. CONTENTS AN INTEGER
40 GIVJMP = $0008  GIVE CONTENTS OF ACCUM. & Y REG. TO
                                                  BASIC
50 FACMLO = $0081  FLOATING POINT ACCUMULATOR MLSB
60 FACLO  = $00B2  FLOATING POINT ACCUMULATOR LSB
70;
80         * = $6000
90;
100 START  JSR GETVAL     MAKE ARGUMENT AN INTEGER
110        LDA FACLO      GET LSB
120        CLC
130        ADC #$01       ADD 1 TO IT
140        TAY            SAVE RESULT LSB IN Y REG.
150        LDA FACMLO     GET MSB
160        ADC #$00       ADD IN ANY CARRY
170        JMP (GIVJMP)   JUMP/EXIT TO GIVAYF
180;
190 GETVAL JMP (GETINT)   JUMP TO INTEGER CONVERT
200;
210        .END
```

The BASIC program to support this code would be:

```
10 POKE 8778,0:POKE 8779,96:
   REM-SET USR VECTOR TO $6000
20 INPUT "YOUR NUMBER "; Y
30 X = USR(Y):
   REM-PERFORM FUNCTION ON "Y"
40 PRINT X:
   REM-DISPLAY RESULT & QUIT
```

All this program does is to take the number you enter and add 1 to it, displaying the result. Let's again examine the way this program is processed by BASIC. When BASIC sees "X=" in line 30 of the BASIC program, it automatically evaluates the expression on the right side of the "=". Upon encountering the "USR", BASIC knows USR is a function and so it continues by evaluating the argument contained in the parenthesis (i.e. "(Y)"), and jumps to the appropriate code pointed to by locations 8778 and 8779. That's where our machine code takes over. The first thing we have to do is to convert the value held in BASIC's Floating Point Accumulator from floating point format into integer (again, in two's compliment form). The conversion routine is pointed to by a vector held in BASIC's page zero contents at location $ØØØ6. The 65Ø2's instruction set provides a JMP command that uses such a page zero vector, but unfortunately, not a JSR. This forces us to set up a subroutine whose only instruction is the zero page JMP (i.e. "JMP ($ØØØ6)"). Next, we pick up the result in the F.P. Accumulator and add one to it, storing the least significant byte in the Y register and the most significant byte in the Accumulator. Finally, we return to BASIC by doing another zero page JMP through a vector that points to a routine in BASIC which gives the contents of the Accumulator and the Y register to BASIC and makes it a proper floating point value. At this point, BASIC again has control and passes the result we have obtained to the variable "X".

So far - so good, but we need a practical job to perform. Since BASIC is replete with abilities to deal with numerical values, a common use of machine code is to deal with strings. As I noted in an earlier article, the USR function is designed to allow only numerical values to be passed back to BASIC. However, this is not to say that USR is limited to numerical values for its argument (i.e. the "(Y)"). We can validly use the expression "X=USR(Y$)". Further, with some added effort, USR need not be limited to a single argument. We could construct a function that works with two arguments, such as BASIC's MID$ function that has a syntax similar to;

A$ = MID$(B$,X,Y)

A common problem BASIC programmers have to overcome is to find the occurrence (if any) of one string within another, and to locate the position of the substring within the string being searched. It can take an awful lot of program space and execution time for this task, yet it is a relatively trivial task in machine code, as I hope to demonstrate. Simply stated, our job is to see if a one string can be found within another string and if so, at what position is the substring located - giving that position

published monthly

Editor - Eddie Gieske

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

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

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

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

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

value to a variable in BASIC. Well, the idea is simple anyway.

To begin, we need to discuss three routines built into BASIC that we can use within our machine code routine. The first is called "FRMEVL" and this routine is, in my opinion, the very heart of BASIC because its task is to evaluate expressions within BASIC programs and determine the result. If you consider the rules of presidence and the complexities of floating point math and string functions, I think you'll join me in admiring this powerful routine. The second routine takes the information FRMEVL found out about a string and gives it to us in a convenient form. This routine is called "FREFAC" and it tells us the length of the string evaluated and the memory address where the string is stored in memory by BASIC. Fabulous stuff! Last, there is a routine called "GIVAYF" which passes the value of the Accumulator and the Y register and gives it to BASIC in a form the language understands. The page zero jump vector at $0008 we used in the first program points to GIVAYF.

Let's begin to write the assembly language program to perform our desired task. The routine is small, and should only require one track to store the program's source code and one additional track to store the object code. (Remember, after the machine code has been generated, it must be stored on disk and transferred to OS-65U with LOAD48 or LOAD32.) Enter and assemble the following program:

installed WRBPR3

```
 10; STRING SEARCH ROUTINE
 20;
 30 INDEX  = $006F  0-PAGE POINTER USED BY FREFAC
 40 CHKSTR = $0CBE  CHECK FOR STRING EXPRESSION
 50 FRMEVL = $0CCD  FORMULA EVALUATOR
 60 CHKCLS = $0E0D  CHECK FOR ")" IN PROGRAM
 70 CHKCOM = $0E13  CHECK FOR COMMA IN PROGRAM
 80 FCERR  = $1000  FUNCTION CALL ERROR
 90 GIVAYF = $1218  GIVE A/Y PAIR TO F.P. ACCUM.
100 FREFAC = $1520  FIND STRING LENGTH AND ADDRESS
110      * = $6000
120;
130 START  JSR $15FC      GET MATCH STRING INFO
140        BEQ BADSTR     LENGTH 0? ==> BADSTR!
150        STA S1LEN      OK. SAVE AS 1ST STRING LENGTH
160        LDA INDEX      GET MATCH STRING LOC. LSB
170        STA P3+1       SAVE IT BELOW
180        LDA INDEX+1    GET LOCATION MSB
190        STA P3+2       SAVE IT TOO
200        JSR CHKCOM     FIND SEPARATING COMMA
210        JSR FRMEVL     EVALUATE 2ND EXPRESSION
220        JSR CHKSTR     MAKE SURE IT'S A STRING
230        JSR FREFAC     FIND IT
240        CMP S1LEN      SMALLER THAN SEARCH STRING?
250        BCC NOTF       YES! DEFAULT TO NOT FOUND!
260        LDY #$00       INIZ OBJECT POINTER
270        STY S2IND      SAVE AS SEARCH START INDEX
280        STA S2LEN      SAVE LENGTH OF 2ND STRING
290        JSR CHKCLS     MAKE SURE OF ")" IN TEXT!
300 P1     LDY S2IND      FETCH STRING 2 INDEX
310        LDX #$00       INIZ MATCH POINTER/COUNT
320 P2     LDA (INDEX),Y  LOOK AT A CHARACTER
330 P3     CMP $FFFF,X    SAME AS MATCH STRING?
340        BNE P4         NO! RESET & RETRY!
350        INX            YES! BUMP MATCH COUNTER
360        CPX S1LEN      MATCHED ALL CHARACTERS?
370        BEQ FOUND      YES! ==>
380        INY            NO, BUMP SEARCH POINTER
390        CPY S2LEN      LOOKED AT WHOLE STRING?
400        BNE P2         NO! LOOP!
410 P4     INC S2IND      YES! INC SEARCH START INDEX
420        LDA S2IND      SEE WHERE WE ARE
430        CMP S2LEN      AT END OF STRING?
440        BNE P1         NO! RETRY!
450        BEQ NOTF       YES! SHOW NO MATCH! ==>
460 FOUND  LDY S2IND      GET MATCH START INDEX
470        INY            BUMP IT ONE (ORDINAL OFFSET)
480        BNE P5         AND SKIP A BIT
490 NOTF   LDY #$00       YES! SET RESULT LSB = 0
500 P5     LDA #$00       CLEAR RESULT MSB
510        JMP GIVAYF     GIVE RESULT TO BASIC & QUIT!
520;
530 BADSTR JMP FCERR      ERROR! EXIT THROUGH BASIC
540;
550 S1LEN  .BYTE $00      MATCH STRING LENGTH
560 S2LEN  .BYTE $00      OBJECT STRING LENGTH
570 S2IND  .BYTE $00      OBJECT STRING SEARCH INDEX
580;
590        .END
```

Save the resulting machine code on disk with the command;

```
!SA 76,1=6000/1
```

Note that I used track #76 in this example. You will probably use a different track number, just remember which one you do use for reference when you run LOAD48 or LOAD32.

Now, boot OS-65U and create a BASIC program file of about 7000 bytes. This will give you enough space to enter the sample program below and to experiment with it on your own. Next, run the program LOAD48 or LOAD32 as appropriate for your system. Insert the OS-65D diskette that holds the machine code into the "A" drive. At the "A*" prompt, enter the following command;

```
C6000=76,1
```

(Note that you will not have to enter the "=" and "," as LOAD48 and LOAD32 insert them automatically). Finally, enter "GBE12" if you're using LOAD48 or "G7E12" if you're using LOAD32. That will get you to BASIC's "OK" prompt. At that prompt, enter "NEW256". That will preserve a 256 byte buffer at the start of the workspace that will protect our machine code. Now enter the following program;

```
10 POKE 8778,0: POKE 8779,96
20 A$ = "ABCDEFGHIJ": B$ =
   "DEF"
30 X = USR(B$),A$)
40 PRINT X
```

870
0 Lo 11 HI
8955 8956

Note the closing parenthesis following "B$" in line 30. This is a necessary non-standard syntax to allow BASIC to properly interpret "B$" because BASIC is expecting a balancing ")" to compliment the "(" encountered following "USR". I opted for this syntax because it is as close to normal as possible. The unlabeled routine at $15FC in the machine code program is a routine that executes FREFAC and eliminates the conflict of executing a function on a string whose result will be given to a numeric variable. Save this program in the 65U file you created above so that it will save a copy of the machine code, as well as forming a seed program for your future use. Now just enter "RUN". If all has gone well, BASIC should return "4" as the result since B$ ("DEF") occurs in A$ beginning at the 4th character. This routine has many uses. Consider the task of parsing a string entry to separate words within a string. The following code could be used as a subroutine to pick out the next word within a string;

```
1000 POKE 8778,0:POKE 8779,96:
     L=LEN(A$)
1010 IF L=0 THEN B$="":RETURN
1020 X = USR(" "),A$)
1030 IF X = 0 THEN B$ = A$:
     A$ = "": RETURN
1040 B$ = LEFT$(A$,X-1): A$ =
     RIGHT$(A$,L-X): RETURN
```

As you can see, the routine puts the next word in A$ and puts it in B$ and removes B$+" " from A$ before quitting. If A$ is null, then B$ is returned as a null as well. And all this is accomplished with a minimum of space overhead in your program and without having to deal with properly exiting a FOR-NEXT loop. Note that this routine will fall flat on its face if two consecutive spaces are embedded in the string being parsed.

I hope you enjoy this program. For all of you OS-65D users, you should be glad to know that the machine code routines in this article will work for you too. All that needs to be done is to choose a location for the machine code to reside in memory and use that as the origin address for the machine code and in the BASIC programs, instead of POKEing locations 8778 and 8779 to set the USR vector, you must POKE locations 574 and 575.

★

NETAL

AN A. C. CIRCUIT ANALYSIS PROGRAM

By: Michael A. Bryson
203 Meadow View Dr.
Buchanan, MI 49107

Here's a program that really puts your computer to work. It allows you to perform an A.C. circuit analysis on a large number of network element types. I can't take all the credit for it as it was converted from a program written for a Hewlett-Packard computer in HPL. I have added some checks to keep the floating point numbers in range when computing the node voltages as well as a PNP transistor simulation. The original program by Edward Niemeyer appeared in EDN magazine, a trade publication dedicated to the electronics industry. Since many times you wish to change only one component at a time in a large circuit, I have added the option to use data statements to read in the circuit elements. You'll need at least 24K of RAM to run this program on a disk based system. To illustrate the program syntax and software functions, we'll study some example problems. Because of the execution time and memory requirements the program has been limited to 1Ø nodes. If you can tolerate the time and memory, the size can be increased by changing the dimension statements. In any case the number of individual components in a circuit is unlimited.

As a first step you must number each node, in the circuit to be simulated, starting at 1 and consecutively number each node with the highest node number as the network common. You may assign any numbers you want to the input and output nodes. Note, the network excitation (an ideal voltage source of unit magnitude) is always between input and common, and the output readings are presented in relation to the common terminal. The order of element entry is not important as analysis doesn't begin until you select item 11 on the command list.

Let's try a simple example of a resistor, inductor, and capacitor resonant circuit. Figure 1 shows the circuit with the nodes numbered. Figure 2 has a sample run of the circuit showing the input and output steps. After you have selected an item from the command list the program then requests the necessary data for that item and prints out the results on your printer. The matrix solution used has a drawback in that only one output node is solved in each run, thus to analyze all the nodes you must repeat the analysis option of the command list. What I like most about this program is the easy simulation of operational amplifiers which are now universally used for filters and gain stages.

To simulate NPN and PNP transistors you'll need to perform a little direct current analysis of your own first. The effective base emitter resistance must be calculated based on the quiescent conditions so that the program can determine the transconductance. The formulas required are shown in Figure 3 as well as an example of an NPN transistor circuit. For those of you a little rusty with circuit theory remember that a direct current voltage source has a zero impedance relative to an alternating current signal. What did I just say in plain terms? In other words, the power supply looks like a short circuit to A.C. signals and hence the power supply is connected to the signal common for A.C. analysis as shown in the figure. To model an FET you merely provide the transconductance which can be found on the data sheet for the transistor operating with the D.C. conditions of your circuit.

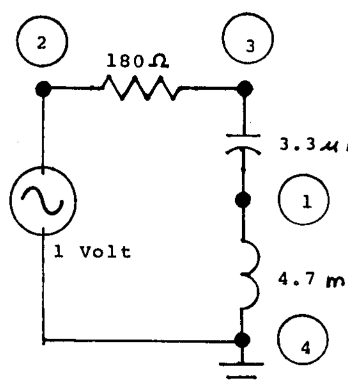
The program also accommodates transmission-line analysis. The model is more complicated and will require longer time for solution. I have provided the example in Figure 4 from the orginal article on this program. I personally haven't had a need to use transmission line analysis, but it's there if you want it. The final Figure 5 shows an example of an Op-Amp circuit. You define the inverting input and output nodes, noninverting input and output nodes, differential voltage gain and output impedance. The circuit shown is a high pass filter set for a 1ØØØ Hertz break point. The program will allow you to easily compute the effects of component tolerances and thus get the most cost effective components to meet your requirements.

A few items you should note. If you are operating with version 3.3 of OS65D, you'll have to run the ARCTAN enabling program before this program is run. What I have done is added a RUN "NETAL" to the end of the ATNENB program copy on the disk which contains the NETAL program. The NETAL program will call ATNENB if it is working under version 3.3. This is the only program I have that needs the ARCTAN function and I always forgot to run the enabling program, thus I highly recommend you make the modification. Also, if you wish to activate the printer form feed function under version 3.3 of OS65D add line 25 PRINT #1,!(65,66). This print command will work if it is activated before the ATNENB program disables the print extensions.

There is a lot of mathematics involved in calculating the circuit transfer function. The more nodes you add the longer each frequency calculation takes. A 1Ø node circuit takes about 45 seconds for a single frequency calculation - be patient! The program computes the total real and imaginary impedances between nodes. If you want to modify the circuit without reentering all the components after an analysis you can merely add a component to the same set of nodes. For example, if you enter a resistor of 1ØØØ ohms between nodes 1 and 2 two times it is the same as entering one resistor of 5ØØ ohms. You can just as easily add components between any set of existing nodes in the circuit and generate a new analysis. Once an analysis is performed you can't add nodes to the circuit, however, a new circuit will have to be entered if more nodes are necessary.

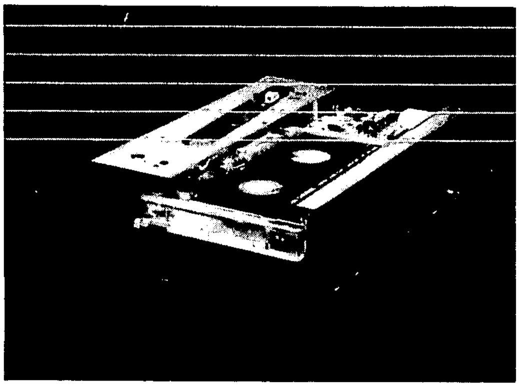
I don't expect there are too many analog designers out there but if you feel this program is useful to you and don't want to run the risk of making a mistake, I'll provide a copy of the program on a OS65D formatted disk for $5.ØØ. Just send me a check and your address and I'll take care of the disk and postage. I have a plotting routine written for another program which could be added to this one but haven't gotten around to converting it. If there's any interest in expanding the features of this program, let PEEK(65) know. If there's enough need they will make some space in a future issue and I'll either send in the additions to the letter to the editor or write another article.

FIGURE 1



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.

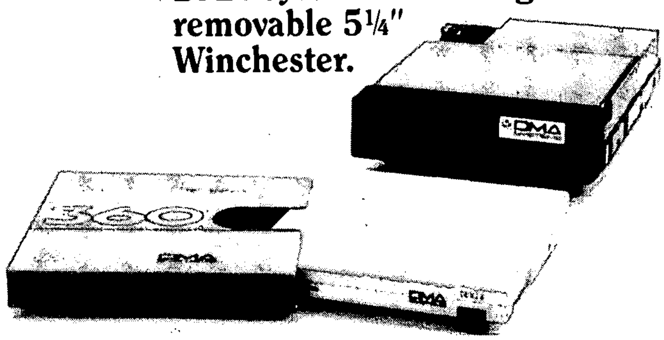
The DMA 360 protects your data in a totally sealed cartridge. Floppies don't.

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

```
CAP  NODE A- 3  NODE B- 1    3.3E-06 Farads
IND  NODE A- 1  NODE B- 4    4.7E-03  Hys
RES  NODE A- 2  NODE B- 3    180  Ohms

ANALYSIS
INPUT NODE- 2   OUTPUT NODE- 3
START FREQ (Hz)- 100   STOP FREQ (Hz)- 16331.5
 7  DATA POINTS
 LOG FREQ-SWEEP

FREQ= 100
AMPL= .9362
20LOG=-.5729
PHASE=-20.5822

FREQ= 233.7963
AMPL= .7423
20LOG=-2.5888
PHASE=-42.0755

FREQ= 546.6072
AMPL= .3718
20LOG=-8.5939
PHASE=-68.1736

FREQ= 1277.9476
AMPL= 0
20LOG=-116.4313
PHASE=-89.9999

FREQ= 2987.7945
AMPL= .3718
20LOG=-8.594
PHASE= 68.1738

FREQ= 6985.3536
AMPL= .7423
20LOG=-2.5889
PHASE= 42.0757

FREQ= 16331.5
AMPL= .9362
20LOG=-.5729
PHASE= 20.5823
```

FIGURE 2

$r_d = \frac{26\text{ mV}}{I_E}$

$r_{be} = r_d(\text{Beta}+1)$

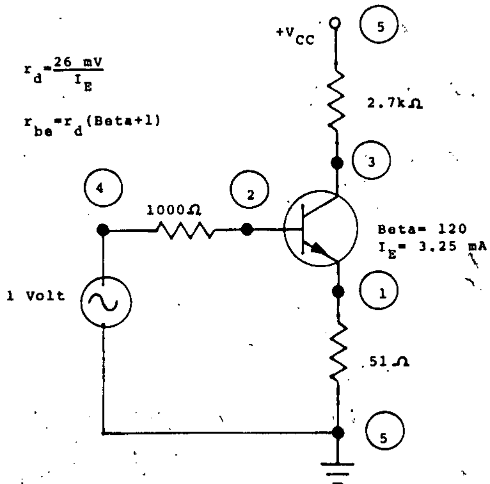


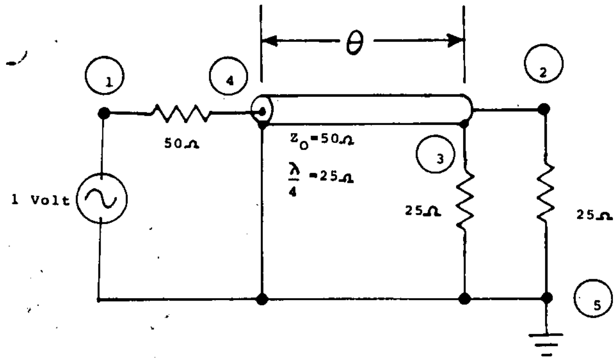
FIGURE 3

```
RES  NODE A- 4  NODE B- 2     1000  Ohms
RES  NODE A- 1  NODE B- 5     51  Ohms
RES  NODE A- 3  NODE B- 5     2700  Ohms
NPN  BASE- 2  EMITTER- 1  COLLECTOR- 3
BETA- 120   Rbe (Ohms) 960

ANALYSIS
INPUT NODE- 4   OUTPUT NODE- 3
START FREQ (Hz)- 1   STOP FREQ (Hz)- 2
 2  DATA POINTS
LIN FREQ-SWEEP

FREQ= 1
AMPL= 39.8475
20LOG= 32.008
PHASE=-180

FREQ= 2
AMPL= 39.8475
20LOG= 32.008
PHASE=-180
```



$\theta = \frac{\text{FREQ}}{\frac{\lambda}{4}\text{FREQ}}\ (90^\circ)$

FIGURE 4

```
RES  NODE A- 1  NODE B- 4    50  Ohms
RES  NODE A- 3  NODE B- 5    25  Ohms
RES  NODE A- 2  NODE B- 5    25  Ohms
 T-LINE  SHIELD IN-5  CENTER IN-4
 CENTER OUT-2  SHIELD OUT-3
 Zo-50  QUARTER WAVE FREQ-25

ANALYSIS
INPUT NODE- 1   OUTPUT NODE- 2
START FREQ (Hz)- 10.01   STOP FREQ (Hz)- 25
 2  DATA POINTS
LIN FREQ-SWEEP

FREQ= 10.01
AMPL= .25
20LOG=-12.0412
PHASE=-36.036

FREQ= 25
AMPL= .25
20LOG=-12.0412
PHASE=-89.9999
```

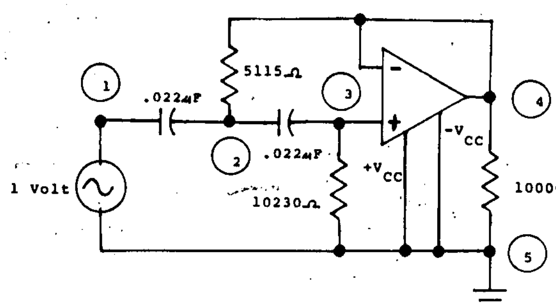


FIGURE 5

```
CAP  NODE A- 1  NODE B- 2    2.2E-08 Farads
CAP  NODE A- 2  NODE B- 3    2.2E-08 Farads
RES  NODE A- 2  NODE B- 4    5115  Ohms
RES  NODE A- 3  NODE B- 5    10230  Ohms
RES  NODE A- 4  NODE B- 5    10000  Ohms
OP-AMP  +IN  3  -IN  4  -OUT  4  +OUT  5
GAIN- 10000  OUTPUT RES (Ohms) 50

ANALYSIS
INPUT NODE- 1   OUTPUT NODE- 4
START FREQ (Hz)- 100   STOP FREQ (Hz)- 10000
 3  DATA POINTS
 LOG FREQ-SWEEP

FREQ= 100
AMPL= .01
20LOG=-40.001
PHASE= 171.8722

FREQ= 1000
AMPL= .7072
20LOG=-3.0093
PHASE= 90.007

FREQ= 10000
AMPL= 1.0001
20LOG= 5E-04
PHASE= 8.1296
```

NETAL

by:Michael A. Bryson

```
10 REM***NETAL By Mike Bryson***
20 IFPEEK(13026)<>171THEN50
30 IFPEEK(2073)=173THENRUN"ATNENB"
40 POKE2073,173
50 DIMA(10,10),B(10,10),I(20),L(20),M(20),N(20),O(20)
60 DIMP(10,10),Q(10,10),R(10,10),S(10,10),T(20),Z(20)
70 J=1
80 I=1
90 X=1:T(X)=0
100 N=0:DT=0
110 INPUT"IS DATA TO BE READ FROM DATA STATEMENTS";A$
120 IFLEFT$(A$,1)="Y"THENDT=1
130 R6=0:IFDT=1THENREADR6:PRINT:GOTO220
140 PRINT"1 RESISTOR":PRINT"2 CAPACITOR"
150 PRINT"3 INDUCTOR":PRINT"4 TRANS LINE"
160 PRINT"5 SHORTED STUB":PRINT"6 OPEN STUB"
170 PRINT"7 OP-AMP":PRINT"8 TRANSISTOR"
180 PRINT"9 FET":PRINT"10 STOP"
190 PRINT"11 ANALYSIS":PRINT"12 PNP TRANS"
200 PRINT"13 NEW CIRCUIT":PRINT
210 INPUT"SELECT FROM LIST";R6:PRINT
220 IFR6=1THENPRINT"(1) RES":GOTO650
230 IFR6=2THENPRINT"(2) CAP":GOTO790
240 IFR6=3THENPRINT"(3) IND":GOTO720
250 IFR6=4THENPRINT"(4) T-LINE":GOTO360
260 IFR6=5THENPRINT"(5) S-STUB":GOTO570
270 IFR6=6THENPRINT"(6) O-STUB":GOTO610
280 IFR6=7THENPRINT"(7) OP-AMP":GOTO1160
290 IFR6=8THENPRINT"(8) NPN":GOTO920
300 IFR6=9THENPRINT"(9) FET":GOTO850
310 IFR6=10THENPRINT"(10) PGM FINISH":STOP
320 IFR6=11THENPRINT"(11) ANALYSIS":GOTO1270
330 IFR6=12THENPRINT"(12) PNP":GOTO1070
340 IFR6=13THENCLEAR:GOTO70
350 GOTO130
360 REM T. LINE
370 IFDT=1THENREADM(X),I(X),O(X),N(X):GOTO410
380 T(X)=1:INPUT"SHIELD IN";M(X)
390 INPUT"CENTER IN";I(X):INPUT"CENTER OUT";O(X)
400 INPUT"SHIELD OUT";N(X)
410 PRINT#1," T-LINE  SHIELD IN-";M(X);" CENTER IN-";I(X)
420 PRINT#1," CENTER OUT-";O(X);" SHIELD OUT-";N(X):GOTO470
430 IFDT=1THENREADM(X),N(X):GOTO460
440 INPUT"NODE A";M(X)
450 INPUT"NODE B";N(X)
460 PRINT#1,"NODE A-";M(X);" NODE B-";N(X)
470 IFDT=1THENREADZ(X),L(X):GOTO500
480 INPUT"Zo";Z(X)
490 INPUT"QUARTER WAVE FREQ (Hz)";L(X)
500 PRINT#1," Zo-";Z(X);" QUARTER WAVE FREQ-";L(X)
510 IFI(X)>NTHENN=I(X)
520 IFM(X)>NTHENN=M(X)
530 IFN(X)>NTHENN=N(X)
540 IFO(X)>NTHENN=O(X)
550 X=X+1:T(X)=0
560 GOTO130
570 REM S. STUB
580 T(X)=3
590 PRINT#1,"S-STUB  ";
600 GOTO430
610 REM O. STUB
620 PRINT#1,"O-STUB  ";
630 T(X)=2
640 GOTO430
650 REM RES
660 IFDT=1THENREADI,J,V:GOTO680
670 INPUT"NODE A";I:INPUT"NODE B";J:INPUT"RES (Ohms)";V
680 PRINT#1,"RES  NODE A-";I;" NODE B-";J;"  ";V;" Ohms"
690 V=1/V
700 GOSUB1780
710 GOTO130
720 REM INDUCTOR
730 IFDT=1THENREADI,J,V:GOTO750
740 INPUT"NODE A";I:INPUT"NODE B";J:INPUT"IND (Hy)";V
750 PRINT#1,"IND  NODE A-";I;" NODE B-";J;"  ";V;" Hys"
760 V=1/V
770 GOSUB1700
780 GOTO130
790 REM CAPACITOR
800 IFDT=1THENREADI,J,V:GOTO820
810 INPUT"NODE A";I:INPUT"NODE B";J:INPUT"CAP (Farads)";V
820 PRINT#1,"CAP  NODE A-";I;" NODE B-";J;"  ";V;"Farads"
830 GOSUB1840
840 GOTO130
850 REM FET
860 IFDT=1THENREADK,J,I,V:GOTO880
870 INPUT"GATE";K:INPUT"SOURCE";J:INPUT"DRAIN";I:INPUT"GAIN (A/V)";V
880 PRINT#1,"FET  GATE-";K;" SOURCE-";J;" DRAIN-";I;" GAIN-";V
890 L=J
900 GOSUB1900
910 GOTO130
920 REM NPN
930 IFDT=1THENREADK,J,I,R5,V:GOTO960
940 INPUT"BASE";K:INPUT"EMITTER";J:INPUT"COLLECTOR";I:INPUT"BETA";R5
950 INPUT"Rbe (Ohms)";V
960 PRINT#1,"NPN  BASE-";K;" EMITTER-";J;" COLLECTOR-";I
970 PRINT#1,"BETA-";R5;"  Rbe (Ohms)";V
980 V=1/V
990 L=I
1000 I=K
1010 GOSUB1780
1020 I=L
```

Listing continued on page 8.

OSI BEATS IBM PC

By: Joseph Ennis
212 20 Street
Niceville, FL 32578

I enjoy the brief period when my program out benchmarks everyone elses, especially if I do it on the OSI and beat an IBM PC or compatible. Besides it has been awhile since PEEK(65) has published a HEXDOS article.

BACKGROUND

Not long ago several of my friends and I were approached and asked if we could help write a computer program to solve some puzzles that are offered in various contest magazines. The puzzle is where letters are assigned specific numerical values and a collection of 5 five letter words are required to be arranged so that the diagonal or any one column spells the word BINGO and the sum of the values of the letters produce the highest score is the winner. I recognized the problem as belonging to the class of problems called Magic Squares. So I accepted the challenge, and after 10 days I had the best solution. My friends all wrote their solutions on IBM PCs and used BASICA, while I used a 1978 vintage Superboard, HEXDOS and OSI RCM BASIC. Looking at all the solutions side by side, the OSI running under HEXDOS is definitely faster and leaner compared to the IBM PC.

I have doubled my system clock to 1.9 MHz which is still a lot slower than the IBM's system clock. I am using a 6502B in a very straight forward architecture with a very lean and efficient DOS compared to the IBM 8088 that has a 20 bit address bus which has to be multiplexed (essentially halving its effective clock speed), a fairly efficient DOS, and a really poor architecture (it is like halving its effective clock speed once more, due to all the bus buffers and latches it has to use). The disk drives on both systems are almost identical, both are Tandon TM-100s, the IBM uses the -2 and I use the -1. The IBM's disk holds about four times the data per disk and shifts the bits at twice the speed, but I can format a disk faster. In addition, at the time of this article I had a total system memory of only 24K as compared

Continued on page 10.

```
1030 L=J
1040 V=R5*V
1050 GOSUB1900
1060 GOTO130
1070 REM PNP
1080 IFDT=1THENREADK,J,I,R5,V:GOTO1110
1090 INPUT"BASE";K:INPUT"EMITTER";J:INPUT"COLLECTOR";I:INPUT"BETA";R5
1100 INPUT"Rbe (Ohms)";V
1110 V=-1/V:L=I:I=K:GOSUB1780
1120 I=L:L=J:V=-R5*V:GOSUB1900
1130 PRINT#1,"PNP  BASE-";K;" EMITTER-";J;" COLLECTOR-";I
1140 PRINT#1,"BETA-";R5;"  Rbe (Ohms)";1/(V/R5)
1150 GOTO130
1160 REM OP-AMP
1170 IFDT=1THENREADK,L,I,J,R5,V:GOTO1200
1180 INPUT"+IN";K:INPUT"-IN";L:INPUT"-OUT";I
1190 INPUT"+OUT";J:INPUT"GAIN (V/V)";R5:INPUT"OUTPUT RES (Ohms)";V
1200 PRINT#1,"OP-AMP  +IN ";K;" -IN ";L;" -OUT ";I;" +OUT ";J
1210 PRINT#1,"GAIN-";R5;" OUTPUT RES (Ohms)";V
1220 V=1/V
1230 GOSUB1780
1240 V=R5*V
1250 GOSUB1900
1260 GOTO130
1270 REM *ANALYSIS*
1280 INPUT"INPUT NODE";E:INPUT"OUTPUT NODE";F:N=N-1
1290 INPUT"START FREQ (Hz)";G:INPUT"STOP FREQ (Hz)";H
1300 INPUT"# DATA POINTS";M
1310 INPUT"FREQ-SWEEP LOG=0(LIN=1)";R6
1320 PRINT#1:PRINT#1,"ANALYSIS"
1330 PRINT#1,"INPUT NODE-";E;"  OUTPUT NODE-";F
1340 PRINT#1,"START FREQ (Hz)-";G;"  STOP FREQ (Hz)-";H
1350 PRINT#1,M;" DATA POINTS":IFR6=0THENPRINT#1," LOG FREQ-SWEEP"
1360 IFR6=1THENPRINT#1,"LIN FREQ-SWEEP"
1370 PRINT#1
1380 D=(H-G)/(M-1)
1390 R4=10↑((LOG(H/G)/LOG(10))/(M-1))
1400 R0=G:R9=0
1410 R9=R9+1
1420 W=6.28318*R0
1430 O=E:Z=F
1440 GOSUB2910
1450 LP=0:FP=0
1460 GOSUB2590
1470 V=R5:U=Z
1480 IF(E+F)/2=INT((E+F)/2)THEN1500
1490 U=U-180
1500 O=E:Z=E
1510 GOSUB2590
1520 U=U-Z
1530 IFV=0THENR7=-9999:GOTO1560
1540 IFR5=0THENR7=9999:GOTO1560
1550 V=V/R5:R7=20*LOG(V)/LOG(10)
1560 IFU>180THENU=U-360
1570 IFU<-180THENU=U+360
1580 PRINT"FREQ";R0:PRINT"AMPL";V:PRINT"20LOG";R7
1590 PRINT#1,"FREQ=";INT(R0*10000+.5)/10000
1600 PRINT#1,"AMPL=";INT(V*10000+.5)/10000
1610 PRINT#1,"20LOG=";INT(R7*10000+.5)/10000
1620 PRINT#1,"PHASE=";INT(U*10000+.5)/10000:PRINT#1
1630 PRINT"PHASE";U:PRINT
1640 IFR6=0THENR0=R0*R4
1650 IFR6<>0THENR0=R0+D
1660 IFR9<>MTHEN1410
1670 N=N+1
1680 DT=0
1690 GOTO130
1700 REM INDUCTOR
1710 R(I,I)=R(I,I)+V
1720 R(J,J)=R(J,J)+V
1730 R(I,J)=R(I,J)-V
1740 R(J,I)=R(J,I)-V
1750 IFI>NTHENN=I
1760 IFJ>NTHENN=J
1770 RETURN
1780 REM RESISTOR
1790 P(I,I)=P(I,I)+V
1800 P(J,J)=P(J,J)+V
1810 P(I,J)=P(I,J)-V
1820 P(J,I)=P(J,I)-V
1830 GOTO1750
1840 REM CAPACITOR
1850 Q(I,I)=Q(I,I)+V
1860 Q(J,J)=Q(J,J)+V
1870 Q(I,J)=Q(I,J)-V
1880 Q(J,I)=Q(J,I)-V
1890 GOTO1750
1900 REM TRANS
1910 P(I,K)=P(I,K)+V
1920 P(J,L)=P(J,L)+V
1930 P(J,K)=P(J,K)-V
1940 P(I,L)=P(I,L)-V
1950 IFK>NTHENN=K
1960 IFL>NTHENN=L
1970 GOTO1750
1980 REM COMP
1990 IFN>1THEN2020
2000 O=A(1,1):Z=B(1,1)
2010 RETURN
2020 O=1
2030 Z=0
2040 K=1
2050 L=K
2060 S=ABS(A(K,K))+ABS(B(K,K))
2070 I=K-1
2080 I=I+1
2090 T=ABS(A(I,K))+ABS(B(I,K))
2100 IFS>=TTHEN2120
2110 L=I:S=T
2120 IFI<>NTHEN2080
2130 IFL=KTHEN2190
2140 J=0
2150 J=J+1
2160 S=-A(K,J):A(K,J)=A(L,J):A(L,J)=S
2170 A=-B(K,J):B(K,J)=B(L,J):B(L,J)=A
2180 IFJ<>NTHEN2150
2190 L=K+1:I=L-1
2200 I=I+1
2210 A=A(K,K)*A(K,K)+B(K,K)*B(K,K)
2220 S=(A(I,K)*A(K,K)+B(I,K)*B(K,K))/A
2230 B(I,K)=(A(K,K)*B(I,K)-A(I,K)*B(K,K))/A
2240 A(I,K)=S
2250 IFI<>NTHEN2200
2260 C=K-1
2270 IFC=0THEN2350
2280 J=L-1
2290 J=J+1:I=0
2300 I=I+1
2310 A(K,J)=A(K,J)-A(K,I)*A(I,J)+B(K,I)*B(I,J)
2320 B(K,J)=B(K,J)-B(K,I)*A(I,J)-A(K,I)*B(I,J)
2330 IFC<>ITHEN2300
2340 IFJ<>NTHEN2290
2350 C=K
2360 K=K+1:I=K-1
2370 I=I+1:J=0
2380 J=J+1
2390 A(I,K)=A(I,K)-A(I,J)*A(J,K)+B(I,J)*B(J,K)
2400 B(I,K)=B(I,K)-B(I,J)*A(J,K)-A(I,J)*B(J,K)
2410 IFJ<>CTHEN2380
2420 IFI<>NTHEN2370
2430 IFK<>NTHEN2050
2440 L=1
2450 C=INT(N/2)
2460 IFN=2*CTHEN2490
2470 L=0
2480 O=A(N,N):Z=B(N,N)
2490 I=0
2500 I=I+1
2510 J=N-I+L
2520 S=A(I,I)*A(J,J)-B(I,I)*B(J,J)
2530 A=A(I,I)*B(J,J)+A(J,J)*B(I,I)
2540 T=O*S-Z*A
2550 Z=Z*S+O*A
2560 O=T
2570 IFI<>CTHEN2500
2580 RETURN
2590 REM DET
2600 R5=N
2610 N=N-1
2620 I=0
2630 K=0
2640 K=K+1
2650 IFK<>OTHEN2670
2660 I=1
2670 J=0:L=0
2680 L=L+1
2690 IFL<>ZTHEN2710
2700 J=1
2710 A(K,L)=P(K+I,L+J)
2720 B(K,L)=W*Q(K+I,L+J)-R(K+I,L+J)/W+S(K+I,L+J)
2730 IFL<>NTHEN2680
2740 IFK<>NTHEN2640
2750 GOSUB1980
2760 N=R5
2770 Y=Z
2780 IFLP=1GOTO2800
2790 IFO<1E-15THENFP=1
2800 IFFP=1THENO=O*1E15:Z=Z*1E15
2810 R5=SQR(O*O+Z*Z)
2820 LP=1
2830 IFO=0THEN2890
2840 Z=57.29578*ATN(Z/O)
2850 IFO>0THENRETURN
2860 Z=Z+SGN(Y)*180
2870 IFY=0THENZ=180
2880 RETURN
2890 Z=90*SGN(Y)
2900 RETURN
2910 REM T. LOAD
2920 IFT(1)=0THENRETURN
2930 X=0
2940 R1=0
2950 R2=0:R1=R1+1
2960 R2=R2+1
2970 S(R1,R2)=0
2980 IFR2<>N+1THEN2960
2990 IFR1<>N+1THEN2950
3000 X=X+1
3010 IFX>20THENRETURN
3020 IFT(X)=0THENRETURN
3030 IFT(X)=1THEN3090
3040 IFT(X)=2THEN3280
3050 R1=-1/(Z(X)*TAN(.25*W/L(X)))
3060 Q=M(X):R=N(X)
3070 GOSUB3320
3080 GOTO3000
3090 R1=-1/(Z(X)*TAN(.25*W/L(X)))
3100 R=I(X):Q=M(X)
3110 GOSUB3320
3120 Q=N(X):R=O(X):GOSUB3320
3130 R1=1/(Z(X)*SIN(.25*W/L(X)))
3140 P=I(X)
3150 R=N(X)
3160 S(R,P)=S(R,P)-R1
3170 S(P,R)=S(P,R)-R1
```

Continued

```
3180 R=O(X)
3190 S(R,P)=S(R,P)+R1
3200 S(P,R)=S(P,R)+R1
3210 P=M(X)
3220 S(R,P)=S(R,P)-R1
3230 S(P,R)=S(P,R)-R1
3240 R=N(X)
3250 S(R,P)=S(R,P)+R1
3260 S(P,R)=S(P,R)+R1
3270 GOTO3000
3280 R2=1/(Z(X)*TAN(.25*W/L(X)))
3290 R3=1/(Z(X)*SIN(.25*W/L(X)))
3300 R1=R3*R3/R2-R2
3310 GOTO3060
3320 S(Q,Q)=S(Q,Q)+R1
3330 S(R,R)=S(R,R)+R1
3340 S(Q,R)=S(Q,R)-R1
3350 S(R,Q)=S(R,Q)-R1
3360 RETURN
3370 END
```

★

Continued from page 7.

to a typical IBM of 256K. Good thing that HEXDOS only takes 2K bytes.

PROGRAM DEVELOPMENT

I had been working in FORTH just prior to this exercise and I feel this enhanced my style. My first trial solution was a "brute force" approach, that is try every possible combination and select the best results. It worked well with a small dictionary, but when I tried it with the full dictionary it just took too long (hours for a run!!). Also, the first program was getting slow to test as its length increased, so I dropped this approach. Then it occurred to me to split the application into tasks, small concise ones like in FORTH definitions. Each task is to do a single thing. Each task is stored as a separate program file on disk and the data that is to be passed between programs is in data files on the disk. One task does not need to know how the data got into a data file, it only needs to know the format and to be assured that the data is correct for its task. Kind of like the use of the stack in FORTH. This may not sound like much of a discovery or technique, but I recommend that you try it as it is quite powerful in producing clear testable code fast. I divided the problem up into three tasks. It took a lot less time to write these three programs and a lot less time to verify them. The three separate programs are presented in Listing 1. The first has the single function of accepting the keyboard entry of the letter value data for each new contest and packing it into a file called "Values.Data", then it calls the second pro-

LISTING 1A

by: Joseph Ennis

```
1 REM VALUES  CREATES THE VALUE OF EACH LETTER TO BE USED IN A SQUARE
5 DIMA$(39)
10 LOAD#4,"Values.Data"
20 INPUT"WHAT GAME";A$
30 PRINT#4,A$
35 PRINT:PRINT"ENTER VALUES"
40 FORI=1TO26
50 PRINT" FOR-";CHR$(I+64);:INPUTA$
60 PRINT#4,A$
65 PRINT#2,CHR$(I+64);A$
70 NEXTI
80 PRINT:INPUT" ARE VALUES OK";A$
90 IFLEFT$(A$,1)="N"THEN35
100 SAVE*4
110 PRINT:INPUT"DO YOU WANT TO RUN IT";A$
120 IFLEFT$(A$,1)="Y"THENRUN"WORDS"
130 END
```

LISTING 1B

```
1 REM WORDS, WORD FILE CREATOR, WORDS (11AUG84)
2 L=7:N=1:W=1949
10 LOAD*5,"Words1.Data":LOAD*6,"HiWords.Data":LOAD*7,"Words2.Data
20 DIMWRD$(W),S$(5,5),S(5,5),V$(26),V(26)
30 FORL=1TOW
40 IFL<1168THENINPUT#5,WRD$(L)
45 IFL>1167THENINPUT#7,WRD$(L)
50 A=0:IFLEN(WRD$(L))>5ORWRD$(L)="BINGO"ORWRD$(L)="PEKOE"THENNEXTL
55 IFLEN(WRD$(L))<5THENNEXTL
60 PRINTL:WRD$(L)
70 NEXTL
80 PRINT"DONE FIRST PART"
140 FORL=1TOW
145 PRINTTAB(12)L;WRD$(L)
150 FORJ=1TO5
160 IFMID$(WRD$(L),J,1)="B"THENL$="B":GOSUB2000
170 IFMID$(WRD$(L),J,1)="I"THENL$="I":GOSUB2000
180 IFMID$(WRD$(L),J,1)="N"THENL$="N":GOSUB2000
190 IFMID$(WRD$(L),J,1)="G"THENL$="G":GOSUB2000
200 IFMID$(WRD$(L),J,1)="O"THENL$="O":GOSUB2000
210 NEXTJ,L
220 PRINT:PRINT"DONE SECOND PART"
230 FORI=1TO5:FORJ=1TO5
240 PRINT#6,S$(I,J):PRINT#6,S(I,J)
245 PRINT#2,S$(I,J);S(I,J)
250 NEXTJ,I
260 SAVE*6
270 PRINT"DONE THIRD PART"
280 END
999 REM SUBROUTINE FOR HIGH VALUE WORDS
2000 SUM=0:FORK=1TO5
2010 SUM=SUM+V(ASC(MID$(WRD$(L),K,1))-64)
2020 NEXTK
2030 IFL$="B"THENS=1
2040 IFL$="I"THENS=2
2050 IFL$="N"THENS=3
2060 IFL$="G"THENS=4
2070 IFL$="O"THENS=5
2080 FORK=1TO5
2090 IFMID$(WRD$(L),K,1)=L$THENGOSUB4000
3000 NEXTK:RETURN
3010 END
4000 IFSUM>S(S,K)THENS(S,K)=SUM:S$(S,K)=WRD$(L)
4010 PRINTS;K;" ";WRD$(L)
4020 RETURN
```

Listings continued on page 11

gram. The second program performs three tasks. Since these tasks are not separable they must be in a single program. The tasks are marked with REM statements. They are:

1. OPEN an input buffer and LOADs the output of the first program into this buffer, OPEN an input buffer and link it to the disk file called "Words. Data" (the buffer is only 2K bytes but the file is approximately 53K bytes long; however, as the data is needed HEXDOS is smart enough to see that the proper disk track is read off the disk and into the buffer), OPEN an output buffer to hold the results that are passed on to the next program.

2. The second part is my algorithm for finding a solution.

3. The third part PRINTs the output to the screen so progress can be monitored, and PRINTs the results to the output buffer, as the file "HiWords.Data" and when done calls the third program called "Solutions".

The third program takes the output of the second program by OPENing an input buffer, then forms the highest value Magic Squares with the word BINGO appearing in each of the vertical columns, then it forms the highest value Magic Square with the word BINGO on the upper left to lower right diagonal (that is the only diagonal asked for). Since there are only 6 things to be ranked at this level in the solution a simple bubble sort is sufficiently fast to finish off the final determination as to which solution is the highest value. The final 6 solutions are then PRINTed to the screen, the printer and to a back-up output file called "Solutions.Data".

CHALLENGE TO 65D and 65U

Anyone using 65D or 65U is challenged to try to out benchmark this application in size and speed. My current program has a run time of less then 4 minutes on a dictionary of 1945 five letter words and finds solutions, (no matter what letter values are given) in the 500 plus range. To out benchmark me, your system and code must produce the same solution faster. Also, if you find a higher solution than my algorithm missed, then you win at any speed.

Anyone wishing a copy of the dictionary disk file, and can

LISTING 1C

```
10 REM SOLUTION RANKING PROGRAM
20 DIMWRD$(5,5),SOL$(6),VA(5,5),V(6)
30 LOAD#5,"HiWords.Data":LOAD#4,"Solutions":LOAD#7,"Values.Data"
40 FORI=1TO5:FORJ=1TO5
50 INPUT#5,WRD$(J,I),VA(J,I)
60 NEXTJ,I
70 REM J=1 IS ALL B's, J=2 IS ALL I's, ECT. I=1 IS FIRST COLUMN, ETC.
80 REM SOLUTION OF ARRAY
90 FORI=1TO5:V(I)=0:SOL$(I)="":FORJ=1TO5
100 SOL$(I)=SOL$(I)+WRD$(I,J)
110 V(I)=V(I)+VA(I,J)
120 NEXTJ,I
130 REM PRINCIAPL DIAG SOLUTION
150 V(6)=0:SOL$(6)="":FORI=1TO5
160 SOL$(6)=SOL$(6)+WRD$(I,I)
170 V(6)=V(6)+VA(I,I)
180 NEXTI
190 REM SORT TO RANK SOLUTIONS USING SIMPLE BUBBLE
200 FLAG=0:FORI=2TO6
210 A=V(I):B=V(I-1):A$=SOL$(I):B$=SOL$(I-1)
220 IFA>BTHENV(I)=B:V(I-1)=A:SOL$(I)=B$:SOL$(I-1)=A$:FLAG=1
230 NEXTI
240 IFFLAGTHEN200
245 INPUT#7,A$:PRINT#2,A$:PRINT#2
250 FORI=1TO6
260 PRINT#4,V(I):PRINT#4,SOL$(I)
270 PRINT#2:PRINT#2,"This solution is equal to:"V(I)
280 FORJ=1TOLEN(SOL$(I))STEP5
290 PRINT#2,"     ";:PRINT#2,MID$(SOL$(I),J,5)
300 NEXTJ,I
310 PRINT#2:PRINT#2,"JOB IS COMPLETE. SOLUTION STORED UNDER Solutions."
320 PRINT"DONE"
330 END
```

read a HEXDOS disk, send enough to cover the cost of a disk, a disk mailer, and postage.

★ ★

BEGINNER'S CORNER

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

EDIT & ZERO FOR SFE

Gee-whizz - what an omission! Last month the first part of the Sequential File Editor was discussed. The program, as it stands, cannot be used to delete records from a file. But, having written a program with some structure, the fix is simple. During EDIT mode mark the field of the record to be deleted, and then omit to save those fields when the file is written back to disk.

Make these changes:

```
550 GOSUB 5370:DISK OPEN,6,FI$:
    FOR C=1 TO L: IF D$(C)="d"
    THEN 560

555 PRINT#6,D$(C)

560 NEXT C: DISK CLOSE,6:
    GOSUB5370
```

Any field which consists of just the letter "d" is not saved to disk. Simple. Choose your own marker if "d" is unacceptable - even the word "deleted" could be used as a marker.

Add 'N$=""' to the end of line 140.

The program will allow files to be merged - merely load them one after the other. The limit on the number of fields is set by "X=2000" in line 120. A check for this limit is required; add this line:

455 IF L=X THEN 490.

EDIT

The EDIT block (see this month's listing) makes extensive use of the cursor-addressing commands of DOS 3.3. The "print at" command in line 800 moves the cursor to the position at which the message, in line 810, will be printed. In line 830, the "!(15)" command will blank the line from the given "print at" position. Text deleted from these lines is the text printed by lines 840 and 860. These deletions become necessary when the loop back to line 800 is made after the error N>L.

WINDOWS

The window command, line 890, has its curiosities. In "(22,62,19)" the "22" can be thought of as the window command code. The "62" is the window width and the "19" is the window height. Maximum values are "62" and "22", respectively. One reason for this is probably because both counts begin with zero. Another is because both a width of "63" and a height of "23" would force a line-feed and the window would scroll up a line. Another point to remember is that the first screen address in the window is (0,0). Line 900 provides an example of this. The command "PRINT!(24)" clears the screen/window from the current cursor position.

The program presents a screen of 9 fields for editing. Nine, because if each field consisted of more than 55 characters then 18 lines would be printed on the screen. It is possible to force the printing of up to 18 single-line fields. If the number of lines is too great to fit the window then that number is automatically adjusted, in line 930, until the right number of lines are on the screen. How it works is cunning.

SCREEN CHARACTER INPUT

The first line (line 0) of the EDIT screen should be all blanks. In line 920, the character at position (2,0) is input off the screen, into Y$, with the "!(33)" command. If the number of lines printed in the window is correct then the window has not scrolled and Y$ will contain a blank. A check for this is made in line 930. But wait, the check is for a null! Not another bug! Yup! The blank has been converted to a null and the manual says nothing about this. If Y$ contains a character then the number of lines is reduced by one, a new EDIT screen is printed and the same check made again.

The "!(33)" can be used to "remember" a character on the screen if it is about to be overwritten by a character input from the keyboard - line 960.

SEQUENTIAL DATA FILE EDITOR

```
10 REM Sequential Data File Editor. (c) LZ Jankowski 1985
20 REM All Rights Reserved 1985. Version May 12 '85
30 REM part two
40 :
790 REM ----------------- EDIT ------------------------------
800 PRINT&(4,10);:D=50
810 PRINT"EDIT. How many entries on screen (max. 18 lines)   9"L1$;
820 INPUTY$:F=VAL(Y$):IFY$="x"ORF>XTHEN1090
830 PRINT&(16,8)!(15)&(10,10)!(15):IFF<1ORF>18THENF=9
840 PRINT&(10,10)"From entry #   1"L1$;:INPUTY$:IFY$="x"THEN1090
850 N=VAL(Y$):IFN=0THENN=1
860 IFN>LTHENPRINT&(16,8)"Too large!"&(16,10);:GOTO800
870 :
880 PRINT&(0,2)"*FD entry -> <RETURN> *FD page -> ; *BK entry -> /";
890 PRINT" *BK page -> -":PRINT&(0,3)!(22,62,19):POKE2797,T2:T=8
900 FORK=NTOLSTEPF:PRINT&(0,0)!(24):POKEG,T2
910 FORC=0TOF-1:PRINTB$K+CTAB(T)D$(K+C):IFC+K=LTHENC=F-1
920 NEXTC:E=0:PRINT&(2,0)!(33):INPUTY$
930 IFY$<>""THENK=K-F:F=F-1:GOTO1080
940 Y=0:FORC=1TOD:POKEG,T2:PRINT&(0,C)!(33):INPUTY$
950 IFY$<>B$THENGOSUB1150:GOTO1070
960 PRINT&(T,C)!(33):INPUTT$:PRINT&(T,C)L$L$;:POKEG,CU:INPUTY$
970 IFT$=""THENT$=" "
980 E=E+1:IFY$=""THEN1070
990 IFY$="-"THENC=D:K=K-2*F:GOTO1070
1000 IFY$=";"THENC=D:GOTO1070
1010 IFY$="x"THENC=D:K=L:NEXTC,K:GOTO1090
1020 IFY$="/"THENPRINT&(T,C)T$:GOSUB1110:GOTO1070
1030 IFLEN(Y$)=2ANDLEFT$(Y$,1)=";"THENGOSUB1210:GOTO1070
1040 IFY$=N1$ORY$=N2$THENY$=""
1050 D$(K+E-1)=Y$:IFLEN(Y$)>F5THENC=D:K=K-F:GOTO1070
1060 Y=-1:PRINT&(LEN(Y$)+T,C)!(15)
1070 NEXTC:IFK<0ORK>LTHENK=L
1080 NEXTK:N=1:GOTO900
1090 POKE2797,63:PRINT!(21):RETURN
1100 :
1110 C=C-2:IFC<0THENC=0
1120 E=E-2:IFE<0THENE=0
1130 T$=Y$:RETURN
1140 :
1150 IFY$<>""THEN1170
1160 PRINT!(33):INPUTY$:IFY$=""THENC=D
1170 IFT$="/"THENC=C-2:T$="":IFC<0THENC=0
1180 IFYTHENPRINT!(12)!(15):Y=0
1190 RETURN
1200 :
1210 C=D:H=100*VAL(RIGHT$(Y$,1)):K=K+H-F:IFK>LTHENK=K-H+F
1220 RETURN
1230 :
1610 REM -------------- ZERO OUT A FILE ---------------------
1620 PRINT"ZERO. Which file   "F2$L2$;:INPUTY$:IFY$="x"THEN1790
1630 DEFFNA(Y)=10*INT(Y/16)+Y-16*INT(Y/16)
1640 FORC=11897TO11930:READN:POKEC,N:NEXTC
1650 DATA 72,138,72,152,72,169,126,133,208,169,57,133,209,162,12
1660 DATA 160,0,152,230,209,145,208,200,208,251,202,208,246,104,168
1670 DATA 104,170,104,96:POKE8955,121:POKE8956,46:Y=USR(Y)
1680 RESTORE:IFY$<>""THENF2$=Y$
1690 IFLEN(F2$)<6THENF2$=F2$+" ":GOTO1690
1700 PRINTC$&(16,10)"* Reading directory *"&(0,0):GOSUB5370:F=0
1710 Y=11897:E=12145:DISK!"CA 2E79=08,1":GOSUB 1810:IFFTHEN1740
1720 DISK!"CA 2E79=08,2":GOSUB 1810
1730 IFF=0THENPRINT&(16,10)!(15)NF$:ER$="^ No such name! ^":GOTO1780
1740 PRINT&(5,10)F2$". Zero tracks"N"through to"T". Continue ?"L$;
1750 GOSUB5400:IFY$<>"y"THEN1780
1760 PRINTC$&(16,10)"* Zeroing file on disk *"&(0,0):Y$=",1=3A7E/C"
1770 FORC=NTOT:T$=RIGHT$(STR$(C+100),2):DISK!"SA "+T$+Y$:NEXTC
1780 GOSUB5370
1790 RETURN
1800 :
1810 FORC=YTOESTEP8:T$="":FORK=CTOC+5:T$=T$+CHR$(PEEK(K)):NEXTK
1820 IFF2$=T$THENH=C:C=E:F=-1
1830 NEXTC:N=FNA(PEEK(H+6)):T=FNA(PEEK(H+7)):RETURN
4980 :
```

An edited line could well be shorter than the original entry and the excess is best deleted. This is done in line 1060 with the clear to end of line command.

RAPID SEARCH

It is possible to move forward through the fields in multiples of 100. Press the ";" key followed by a single digit. For example, ";3" will move the EDIT screen forward by 300 fields - line 1030.

A BETTER EDITOR

The EDIT block employs fancy footwork but is still unsatisfactory. To edit a line it is necessary to retype the whole line, even if only one character change is required. Selecting a line for editing is clumsy. A cleaner edit method is presented in "WAZZAT."

ZERO

The "ZERO" utility presented here is self-contained and gives the same result as the OSI utility. The Assembler listing is a short program that fills the device #6 buffer RAM with zeroes. Users of 5" disks change line 60 to "PC=$08". DOS 3.2 users will

```
 10                    ; New ZERO utility
 20                    ; (c) L.Z. Jankowski
 30                    ;                1985
 40 2E79                   * = $2E79
 50 00D0=                 ZP = $D0
 60 000C=                 PC = $0C
 70                    ;
 80 2E79 48                  PHA
 90 2E7A 8A                  TXA
100 2E7B 48                  PHA
110 2E7C 98                  TYA
120 2E7D 48                  PHA
130 2E7E A97E                LDA #$7E
140 2E80 85D0                STA ZP
150 2E82 A939                LDA #$39
160 2E84 85D1                STA ZP+1
170 2E86 A20C                LDX #PC
180 2E88 A000                LDY #$00
190 2E8A 98                  TYA
200 2E8B E6D1      TWO       INC ZP+1
210 2E8D 91D0      ONE       STA (ZP),Y
220 2E8F C8                  INY
230 2E90 D0FB                BNE ONE
240 2E92 CA                  DEX
250 2E93 D0F6                BNE TWO
260 2E95 68                  PLA
270 2E96 A8                  TAY
280 2E97 68                  PLA
290 2E98 AA                  TAX
300 2E99 68                  PLA
310 2E9A 60                  RTS
```

need to change the second byte in line 150 of the Assembler listing to $31 for 5" and to $30 for 8" disks. Notice that the value here is one less, e.g., $39 and not $3A. The increment to $3A takes place in line 200.

The utility is POKEd up into the directory buffer and then called with "Y=USR(Y)" - line 1670. Notice that the disk USR addresses, 8955 and 8956 are used. The BASIC USR addresses, 574 and 575, cannot be used.

Why the RESTORE command in line 1680? This command resets the pointer in a program's DATA list to the first item. The same DATA items can be read over again. To be able to do this is necessary because the data is POKEd into the directory buffer. This buffer will be subsequently overwritten by the directory when it is copied from disk.

After the device #6 buffer has been filled with zeroes the file name is padded out with blanks if its length is less than 6. Next, the first half of the directory is copied from disk. A search is made for the file name, lines 1810-1830.

Once the file name has been found, a final choice is offered on whether or not to zero the appropriate tracks. If the choice made is "yes" then the contents of the device #6 buffer are written to disk as many times as there are tracks to be zeroed. The whole thing is short in code and quick in action.

To force the use of the "zero" subroutine this line should be added:

```
535 F2$=F1$: GOSUB 1630
```

Next month - a fast, sequential file, selective sort.

★

MISCELLANEOUS SORT ALGORITHMS

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

I have had a lot of experience in sorting files, and the program SORTER summarizes my experience. Although I have 3 Machine-language sorts available, I nearly always sort in BASIC as it is a lot more flexible and the sorting time is insignificant compared to the printing time.

A( ) is the array to be sorted, N the number of elements.

All sorts will run faster if the variables used are listed in the first line of the program.

For an alphabetic (ASCII) sort, just substitute A$( ) for A( ), and K$ for K. To sort on two fields at once, say DEPT$ and NAME$, set A$(I)=DEPT$+NAME$ for each I. To sort on two numeric fields at once, say CUST and INV, set A(I)=M*CUST+INV for each I, where M is bigger than any invoice number. The maximum A(I) must be less than 4,294,967,296 if the last digit is critical. To sort on an alphabetic field and a numeric field, say NAMES$ and INV, set A$(I)=NAME$+RIGHT$("   "+STR$(INV),6) or a similar formula.

CHOOSING AN ALGORITHM

(1) INDIRECT VS. DIRECT

Usually one needs to keep track of the original order, so an index or pointer array P( ) or P%( ) is needed as well as the main array A( ). Before sorting, set P(1)=1, P(2)=2, etc. You can carry the pointer array along passively (the direct sorting method) or use it to do the work and leave the main array unsorted (the indirect method). After an indirect sort, you read the main array as in line 80.

You must choose an indirect sort if you are sorting strings and the SWAP verb is not available. This program contains code for the SWAP verb, enabled by the routine at 800. If SWAP is not available, you can substitute T=A(I): A(I)=A(J): A(J)=T. But in a direct string sort this causes garbage-collection delays.

An indirect sort is also preferable if you have two or more related arrays, say accounts AC$( ) and amounts AM( ). A direct sort would rearrange one array but not the other.

A direct sort is preferable when you are sorting certain records from a file, as you can use the pointer array P%( ) for the record numbers, so that P%(1)=6, P%(2)=8, say. If the indirect method is necessary, then a third array R%( ) is needed for the record numbers, and after sorting they can be read in order as R%(P%(1)),R%(P%(2)), etc.

The indirect Shell-DPM runs 6% slower and the indirect Quicksort 2% slower, if an integer array P%( ) is used for the pointers. But an integer array will save 3*N bytes of memory, and in string operations such as reading a file before a string sort, it will save time by making garbage collection less frequent.

(2) QUICKSORT VS. SHELL-DPM

The ideal situation for Quicksort is when the array is randomly arranged and has few or no duplicates. If you are sure the array is random you can speed up the sort 7% to 10% by eliminating the SWAP in lines 100 and 300, which chooses the middle element as the "pivot" in case the array is partially sorted.

```
1 REM *******************    S O R T E R    ***************************
2
10 I=0: J=0: K=0: G=0: T=0: L=0: U=0: S=0: CR$=CHR$(13)
20 N=1000: REM Number to sort
30 DIM P(N),A(N),L(20),U(20),RS(5)
40 FOR I=1 TO N: P(I)=I: A(I)=N*RND(1): NEXT
50 INPUT"QUICKSORT OR SHELL-DPM SORT";R$
60 IF R$="Q" THEN L=1: U=N: S=0: GOSUB 800: GOSUB 100: GOSUB 900
70 IF R$="S" THEN G=N: GOSUB 200
80 PRINT CHR$(7): FOR I=1 TO N: PRINT A(P(I)): NEXT: REM Indirect sort
90 END
97
98 REM   INDIRECT QUICKSORT  (1000 elements in 100 seconds, at 2 Mhz)
99
100 PRINT L;CR$;: NULL P(U);P((L+U)/2): J=L-1: K=A(P(U))
110 FOR I=L TO U: IF A(P(I))<=K THEN J=J+1: NULL P(J);P(I)
120 NEXT: IF J+1<U THEN S=S+1: L(S)=J+1: U(S)=U
130 U=J-1: IF L<U GOTO 100
140 IF S THEN L=L(S): U=U(S): S=S-1: GOTO 100
150 RETURN
160
198 REM   INDIRECT SHELL-DPM SORT  (without SWAP, 140 seconds)
199
200 G=2*INT(G/7)+1: PRINT G;CR$;: FOR J=1 TO N-G: T=P(J+G): K=A(T)
210 FOR I=J TO 1 STEP -G: IF A(P(I))>K THEN P(I+G)=P(I): NEXT
220 P(I+G)=T: NEXT J: IF G>1 GOTO 200
230 RETURN
240
298 REM   DIRECT QUICKSORT  (110 seconds)
299
300 PRINT L;CR$;: J=(L+U)/2: NULL A(J);A(U);P%(J);P%(U): J=L-1: K=A(U)
310 FOR I=L TO U: IF A(I)<=K THEN J=J+1: NULL A(J);A(I);P%(J);P%(I)
320 NEXT: IF J+1<U THEN S=S+1: L(S)=J+1: U(S)=U
330 U=J-1: IF L<U GOTO 300
340 IF S THEN L=L(S): U=U(S): S=S-1: GOTO 300
350 RETURN
360
398 REM   DIRECT SHELL-DPM SORT  (using SWAP, 130 seconds)
399
400 G=2*INT(G*.22)+1: PRINT G;CR$;: FOR J=1 TO N-G: FOR I=J TO 1 STEP -G
410 IF A(I)>A(I+G) THEN NULL A(I);A(I+G);P%(I);P%(I+G): NEXT
420 NEXT J: IF G>1 GOTO 400
430 RETURN
440
450
800 REM   ENABLE "SWAP" COMMAND
810
820 FOR I=0 TO 3: RS(I)=PEEK(9025+I): NEXT: REM Save reserved word
830 RS(4)=PEEK(8738): RS(5)=PEEK(8739):      REM Save dispatch address
840 POKE 9025,83: POKE 9026,87: POKE 9027,65: POKE 9028,208: REM "SWAP"
850 POKE 8738,255: POKE 8739,95: RETURN:     REM SWAP code at 24576
860
900 REM   DISABLE "SWAP" COMMAND
910
920 FOR I=0 TO 3: POKE 9025+I, RS(I): NEXT
930 POKE 8738,RS(4): POKE 8739,RS(5): RETURN
940
950
```

A Shell-DPM sort is a much safer choice if there may be a number of identical elements. Zeroes and null strings are particularly disastrous: if there is a block of zeros in the middle of the array, for example, the above version of Quicksort will make almost no progress for several minutes. You may also need to consider whether the array is sometimes zeroed out. For example, if you are sorting a customer file by sales year-to-date, the Quicksort will be fine in December, but hopelessly slow in January. You may be able to avoid the problem by sorting on two fields at once, or by saying IF A(I)=0 THEN A(I)=RN/100000, where RN = record number. This may be desirable anyway, to keep the duplicates in a fixed order.

The advantages of Quicksort are its speed (though with smaller arrays there is less difference) and its a more satisfying way of showing progress.

The advantages of the Shell-DPM are its reliability, its smaller memory requirement (by about 300 bytes), and the fact that the indirect version doesn't use SWAP (another 100 bytes).

Next month, more on Sorting Algorithms.

★

DATA RECORDER

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

Listing 1 contains routines which allow you to use your printer to plot curves for real-time data observation in a laboratory or industrial environment. The same technique can be used to plot template curves for engineering design studies. Figure 1 shows results obtained with the test program shown in Listing 2. The method is directly applicable if your printer is an OKIDATA 82A or compatible matrix printer. Almost every dot matrix printer has block-graphics capability, however, and the method described below can be adapted by changing a few values if your printer is different.

BLOCK GRAPHICS

With block graphics, the printer organizes the print head into a 3x2 matrix as shown in Figure 2. Each of the 6 cells are mapped from bits 0-5 of the byte transmitted to the printer. Bits 6 and 7 are set to 0 and 1, respectively. Hence, the ASCII character 128, containing all low-bit zeroes, will print a blank. The character 191, containing low-bit ones, will print a solid matrix. All combinations of blank and solid for the 6 cells are obtained by transmitting a number between 128 and 191.

Up to 120 characters per line are permitted. As each graphics character is divided horizontally into two cells a resolution of 240 cells is obtained. Hence, a plotting accuracy of better than 1/2 per cent is possible. When several channels are plotted the full-scale accuracy for each channel is degraded. For example, with four channels, the accuracy is approximately two per cent of the range for each channel.

CHANNEL SETUP

The channel setup is given by the data contained in lines 300 to 350. In line 300 the value 5 indicates that five channels are to be plotted. The succeeding lines contain scaling information for each channel. For example, in line 320 the values 30 and 50 indicate that the first channel occupies tab settings 30 to 50. The values -100 and 100 indicate the data range for this channel.

In BETA/65 notation, the colon character is used to denote a label. Lines 300 to 350 contain labeled data, similar to the DATA statement in BASIC. As may be seen, labeled data may be referenced by name. For examples, see lines 10 and 110-150. Label names may also be branch targets, as shown in lines 100 and 200.

You may wish to modify the channel setup. This is easily done by changing the values given. This illustrates the value of using software. It is possible to plot multiple data within the same channel, as is done for channels four and five. It is also possible to dynamically modify the channel setup to magnify selected channels if the data for that channel must be accurately displayed.

LINE COMPOSITION

For each curve which is plotted, it is necessary that three data points be collected to complete the matrix. Also, several curves, each occupying a separate channel, must be composed. A buffer string, BUF$, is used to compose the line to be sent to the printer. Each character of BUF$ is initialized to 128, a blank. (Note that, unfortunately, the video characters above 128 do not match the printer set. This results in a random screen pattern.) Those component of BUF$ corresponding to the channel edge are marked by a period. Initialization of BUF$ is done by the subroutine INIZ, starting at line 40000.

The subroutine PLOT receives plot values, along with channel scaling information. Each data point is mapped into the appropriate component of BUF$ by the OR function. Students of logic will recognize that the OR function leaves the bit cell unchanged when a zero is sent and places a one in the cell when a one is sent. This is exactly what is needed for plotting curves on a point-by-point basis. Intersecting curves will also print with this logic.

The channel edges are marked by the period character. Should the data point intersect the channel edge, the marker is replaced by the data point. See line 40232. In the present program, the curve is prevented from overflowing into the adjacent channel by limiting to the edge value. See line 40212 of Listing 1.

Continued

LISTING 1.

```
35000 ! ***********************************
35002 ! **** DATA RECORDER SUBROUTINES ****
35004 ! ** USED TO PLOT SEQUENTIAL DATA ***
35006 ! **** ON OKIDATA 82A PRINTER *******
35008 ! **** CODED IN BETA/65 NOTATION ****
35020 ! ***********************************
35028 !
40000 SUBR INIZ chart
40010 FOR I FROM 1 TO 128 \CHR$(128)=BUF$(I) NEXT I
40012 ! Set high-bit value to blank character.
40018 !
40020 0=Nrow=Nvar
40026 ! Zero row and variable counters.
40028 !
40030 REF chart READ ch.lim
40040 FOR K TO(ch.lim-1)
40042 READ I \B$=BUF$(I)
40044 READ I \B$=BUF$(I)
40046 READ I,I, NEXT K
40048 ! Mark channel edges.
40050 !
40052 RET
40058 !
40060 STRING BUF$(128)
40070 :B$ "." ! Edge marker.
40196 !
40198 !
40200 SUBR PLOT tab.lim % value
40210 REF tab.lim READ T1,T2,V1,V2
40212 value MAX V1 MIN V2=value
40216 ! Limit if out of range.
40218 !
40220 (value-V1)*(T2-T1)/(V2-V1)+T1=tab
40222 (value-V1)*(T2-T1)*2/(V2-V1) MOD 2+Nrow=bit
40226 ! Compute tab setting and low-bit value.
40228 !
40230 TEST BUF$(tab) VS B$ IFNE 40240
40232 CHR$(128)=BUF$(tab) ! Blank over edge marker.
40238 !
40240 CHR$(2^bit OR ASC(BUF$(tab)))=BUF$(tab)
40246 ! Re-compute buffer ASCII value.
40248 !
40250 Nvar+1=Nvar
40252 TEST Nvar MOD ch.lim IFNE 40290
40256 ! Exit if not last variable.
40258 !
40260 Nrow+2=Nrow
40262 TEST Nrow-6 IFMI 40290
40266 ! Exit if not last row.
40268 !
40270 FOR I FROM 1 TO 120
40272 , PRINT TAB(I),BUF$(I)
40274 , NEXT I ! Send to printer.
40278 !
40280 CALL INIZ chart ! Re-initialize buffer.
40284 DELAY 100 ! Increase if printer slow.
40288 !
40290 RET
40296 !
40298 !
40300 ! *************************************
40302 ! ** END OF DATA RECORDER SUBROUTINES **
40304 ! *************************************
```

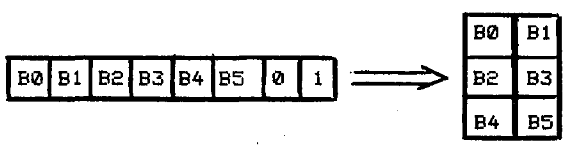
LISTING 2.

```
3 !*****************************
4 !**** TEST PROGRAM USING *****
5 !** DATA RECORDER-TRACK 30 ***
6 !**** CODED IN BETA/65 *******
7 !*****************************
9 !
10 CALL INIZ chart1 ! Initialize buffer.
20 PRINT CHR$ 29 ! Set printer to 120 columns.
30 0=step.nr \0=sn \1000000=cs ! Initialize variables.
78 !
80 PRINT TAB 13,"CH1",TAB 38,"CH2"
90 ,TAB 58,"CH3",TAB 88,"CH4/5"
```

Listing 2 continued

```
92 ! Label first line.
94 PRINT "" ! Skip second line.
98 !
100 :loop ! Plot test profile.
110 CALL PLOT ch1 % step.nr
120 CALL PLOT ch2 % (sn/10000)
130 CALL PLOT ch3 % (cs/10000)
140 CALL PLOT ch4 % (sn/10000)
150 CALL PLOT ch5 % (cs/10000)
198 !
200 :restep
210 step.nr+1=step.nr
220 -sn/1000=dcs
230 cs/1000=dsn
240 dcs+cs=cs
250 dsn+sn=sn
260 TEST step.nr MOD 100 IFNE restep
270 GOTO loop ! Plot 100th step.
296 !
298 !
300 :chart1 5 ! Denotes five channels.
310 :ch1 5 25 0 10000
320 :ch2 30 50 -100 100
330 :ch3 50 70 -100 100
340 :ch4 70 110 -100 100
350 :ch5 70 110 -100 100
390 ! lowtab,hightab,lowvalue,highvalue
398 !
400 ! ************************
402 ! ** END OF TEST PROGRAM **
404 ! ************************
```

Figure 2- Bit Mapping to Print Head.



CONCLUSIONS

If you have spent several hundred dollars for a matrix printer it is only necessary to write less than a page of code to use it to plot curves. The software solution presented here allows you to receive full value from your printer for a small investment of programming time.

Fig. 1 on page 18

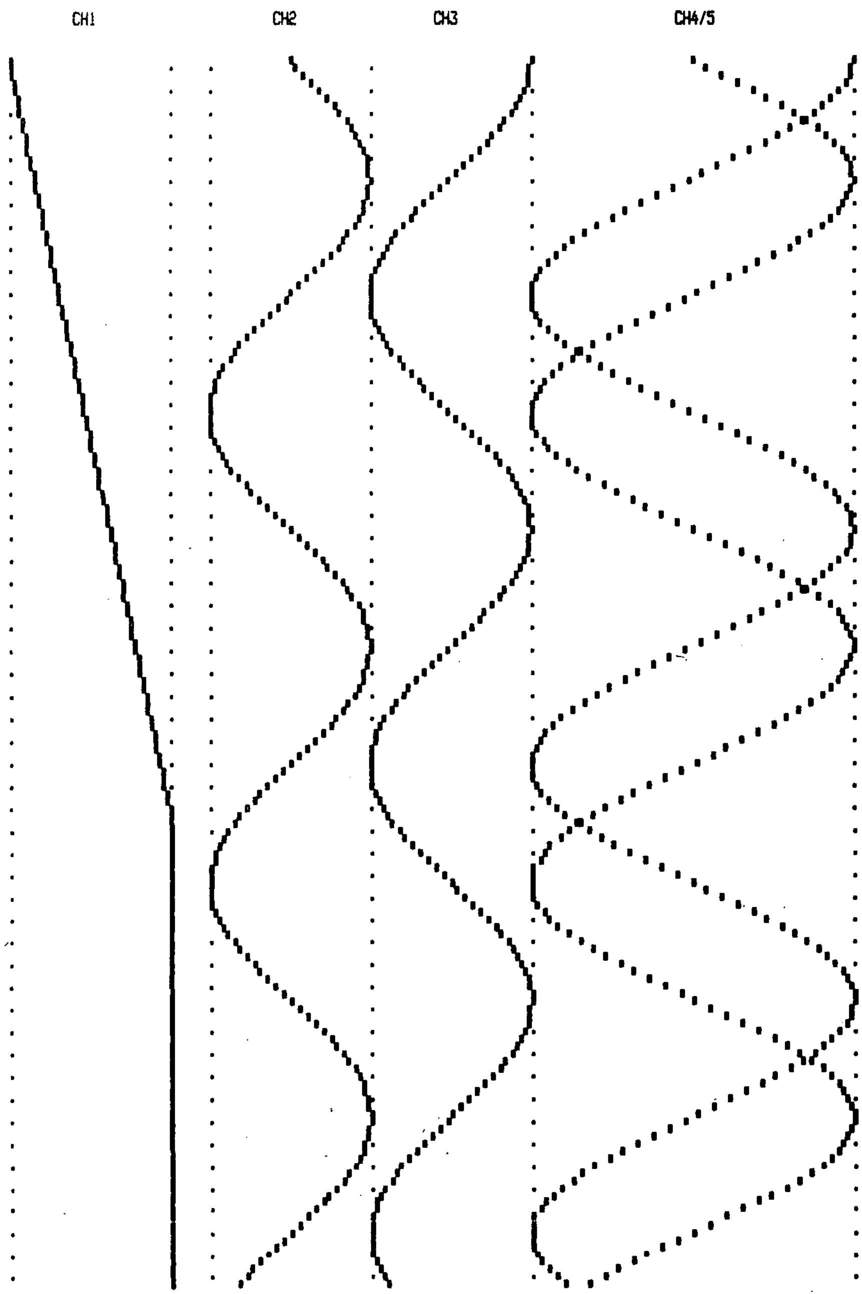
★

GREAT LANGUAGE DEBATE

By: Roy Agee

The current "great debates" over what "computer language" should be taught would be comical-- if the situations it creates were not so serious. Hundreds of thousands of high school and college students are entering the work force totally unprepared with Information Industry job skills. Hundreds of millions of dollars are being spent to equip schools and colleges to "teach" computer languages with little or no value, except to their developers, outside of the classroom.

FIGURE 1.



★ ★

The debate seems to center primarily around which of 2 or 3 languages should be taught; to whom and when. Some of the best, most articulate and humorous of the debating team seem to have one thing in common: they are against BASIC, and for some, the "teaching" of PROGRAMMING in general. (On that point, the traditional teaching of PROGRAMMING, we agree.) However, learning to write a set of instructions to solve routine problems--is a must. To provide a course for computer studies, without learning how to "program" is like a course in Drivers Ed -- without a steering wheel.

The arguments against BASIC include: BASIC is sloppy; BASIC is not efficient; other languages are more efficient; and students cannot transfer skills to other languages. These individuals generally promote teaching such computer languages as LOGO, PASCAL, or, as one particularly humorous individual advocated, PROLOG.

Let's evaluate each of these positions:

1. BASIC is a sloppy language, students can't transfer skills to other languages: BASIC is not sloppy. However, it is often taught in a "sloppy manner." This is due to the fractured and fragmented approach, methods and materials used in the classroom. When BASIC is learned properly, in a structured manner, the concepts and fundamentals of computer languages are easily transferred to the other computer languages such as COBOL, APL, FORTRAN, etc.

2. BASIC is not an efficient language; other languages are more efficient and effective: This, of course, is true. There are several other computer languages that are more efficient than BASIC. There is, however, a flaw in the premise. This analogy will best illustrate the point. There are several languages which are more efficient than English ... so, why teach English? Why not French or Latin? And, in fact, a very good argument can be made for teaching other languages: French - Spanish - Latin, for example. However, these languages are from other cultures and are taught after the student has gained an effective use of the language he/she will NEED to cope with the culture in which they live. Why then do we teach English? Because English was chosen 200 years ago and is USED by a major portion of our population. Why learn BASIC? Because BASIC was chosen (over twenty years ago) and is USED by a major portion of the business/industrial community. Also computer manufacturers provide BASIC with nearly

every microcomputer. A major reason to learn BASIC is that BASIC is USED.

3. LOGO, PROLOG, PASCAL

Of these languages LOGO has the greatest application in elementary and some areas of secondary education. LOGO is excellent for graphics, art, etc. PROLOG presents another problem. Question to software stores, computer specialists, etc. elicits a "blank stare" when asked about PROLOG. So far, no one seems to be aware of PROLOG.

PASCAL is required by the College Board, N.Y. as part of the AP test. This was advanced and heavily promoted by the "ivory tower" types who developed, and profit, from it. While PASCAL does have some limited application outside of the classroom, this is tantamount to requiring LATIN for all students. There is nothing terribly wrong with requiring Latin. But of what real - world value is PASCAL? (It is this observers opinion that Latin is of far more value than PASCAL). Additionally, it has been recently reported that many colleges and universities are now rejecting PASCAL. This country's schools have spent (wasted?) hundreds of millions of dollars in special equipment and software to place PASCAL in the curriculum. Will the $1/2 Billion gamble be a pay-off or a "busted hand"!??

These "ivory-towered" promoters/developers of PASCAL are generally very articulate, intelligent, etc. However, are they sacrificing the practical NEEDS of the student in their pursuit of a perfect language?

The individuals who expend so much talent, time and effort in promoting and participating in such debates are causing far more confusion than clarification. Those most adversely affected are the young people enrolled in the nation's schools and colleges. Too often, school officials use these "great debates" as the reason for doing nothing or continuing with their decades old computer science courses. While this rather foolish debate continues, thousands of high school and college students are graduating, or just leaving school, ignorant of how to use a microcomputer as anything more than a typewriter with a screen - if that!

Until such time as the manufacturers of computers, and the business/industrial complex, adopt another language, students need to learn BASIC for the real world of the 1980's and probably into the 21st Century. The primary objective of a Computer Studies Course must be to teach the use of the microcomputer as a tool for solving problems or to improve and enhance those skills (not to learn a language). Learning or developing new languages is best reserved for the "talented 10%" and those pursuing advanced studies in computer science. The rest of us need to learn to use the tool for more routine activities -- such as survival!

All of us in education need to ask this question: "Am I teaching my students what I want them to learn -- or what they NEED TO KNOW?" This question has relevance to all aspects of education. In computer education, it is of paramount importance to the economy, to our country, and most of all, to our youth who will need to cope with and conquer the challenge of the 21st Century. Those "movers and shakers" in the educational establishment had better "wake up and smell the coffee," before several hundred thousand more high school and college students graduate -- without the necessary skills and abilities required of them in this - The Information Age.

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.

★

WAZZAT CORNER!

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

DISK UTILITIES FOR "HOOKS"

Despite major improvements OS65D 3.3 has, for me, a great failing. Disk utilities cannot be used from within BASIC programs! True, there is one excellent improvement over DOS 3.2 - the TRAP command. Disk errors can be trapped; no dropping out into Immediate Mode upon disk error!

Thanks to "HOOKS," disk utilities can be used from within BASIC programs - (see the source code in PEEK(65) Vol. 4/12 & 5/1, 5/6 - by R. Trethewey). Amazing things can now be done. For example, MAKE new files from within a BASIC program, or view the disk directory! All this, and more, has been gathered together into one program as listed here.

The program also illustrates how disk errors can be properly recorded on the screen. For example, if from the menu, choice "2" was made and the drive was not ready, then the program would report "INIT. --> DISK ERROR <---" and DOS would print "ERR #6 ERROR". All of this is made visible on the screen and tidily presented, thanks to the BASIC 3.3 "PRINT&" (print at) command.

The utility can be appended to any BASIC program. Use "GOTO" to go to the utility as shown in line 50. Making choice "8" from the menu will "return" to the line number found at the end of line 5110. Purists may be fretting here at the lack of "GOSUB". After extensive disk use, "GOSUB" crashes. Notice that "TRAP 6000" is set in line 5010. If you have another "TRAP" remember to reset it after coming back from the utility. If stop/start of disk is not required then just write "RETURN" on line 5370. The command "POKE 2073,173" in line 5430 restores "CTRL-C" control. If you require the standard cursor then, in line 5430, change "128" to "171". In line 5420 insert the correct line number for a return back to the main program. All spacing and REM lines can be removed without affecting the program.

Appending the utility to BASIC programs is simple. Load the utility and send it to the indirect file with "LIST 5000-", press "shift-key and K", followed by a "shift-M" when the listing is finished. Load the BASIC program and now "CTRL-X" brings the utility into BASIC and merges it with the BASIC program.

MORE RAM

One possible problem with "HOOKS" and other extras written to run with DOS and BASIC is that they could use up to 4K of RAM. Where to put it all?

I use the remains of a Superboard and have 54K of RAM using the "Tasker Bus." The extra 6K of RAM is at $C800-$CFFF (2K) and at $E800-$F7FF (4K). The extra 4K is made possible by removing the ACIA chip from the Superboard. (The ACIA is addressed at $F000,1!) "HOOKS" is loaded into, and run from, the 4K block. The 2K block holds the modified Extended Monitor and so it can be used at any time, even from BASIC.

```
10 REM DISK UTILITIES for 'HOOKS' by LZJ. May '85
20 REM Program assumes ZZ=0, for STOP/START disk subroutine.
30 REM Used here: array U$, variables C,Y,Y$,ZZ,W9, TRAP 6000.
40 POKE2888,0:POKE8722,0     :REM Null INPUT enable.
50 GOTO5000
80 :
220 END
4980 :
5000 Y=21:PRINT!(28)&(Y,8)"-------------"&(Y,9)": UTILITIES :"
5010 PRINT&(Y,10)"-------------";TRAP6000
5020 :
5030 U$(1)="Directory":U$(2)="Init. a track":U$(3)="Make a file"
5040 U$(4)="Kill a file":U$(5)="Save the program":U$(6)="Reset"
5050 U$(7)="Select drive":U$(8)="MAIN MENU":U$(9)="X> EXIT program"
5060 FORC=1TO8:PRINT&(Y,11+C)RIGHT$(STR$(C),1)"> "U$(C):NEXTC
5070 PRINT&(Y,12+C)U$(9):POKE13026,63 :REM New cursor
5080 PRINT!(11)TAB(Y-7)"Choice ";:GOSUB5400:IFY$="x"THEN5420
5090 POKE13026,128:IFY=0ORY>7THEN220
5100 PRINT!(28)&(16,8)"To leave press <RETURN>"&(16,10);
5110 ONYGOSUB5240,5130,5170,5210,5270,5300,5330:GOTO5000
5120 :
5130 INPUT"INIT. Which Track ";Y:IFY<1ORY>76THENRETURN
5140 Y=100+Y:Y$=RIGHT$(STR$(Y),2)
5150 GOSUB5370:DISK!"INIT "+Y$:GOSUB5370:RETURN
5160 :
5170 INPUT"MAKE. File name ";Y$:PRINT!(12)!(15);:IFY$=""THENRETURN
5180 PRINT&(16,10);:INPUT"MAKE. # of Tracks ";Y:IFY=0ORY>68THENRETURN
5190 GOSUB5370:MAKEY$,Y:GOTO5250
5200 :
5210 INPUT"KILL. Which file ";Y$:IFY$=""THENRETURN
5220 GOSUB5370:KILLY$:GOTO5250
5230 :
5240 GOSUB5370
5250 PRINT:D$:GOSUB5370:PRINT"Ready ? ";:GOSUB5400:RETURN
5260 :
5270 INPUT"SAVE. File name ";Y$:IFY$=""THENRETURN
5280 GOSUB5370:SAVEY$:GOTO5250
5290 :
5300 PRINT"RESET. <-- Sure ? ";:GOSUB5400:IFY$="y"THENRUN
5310 RETURN
5320 :
5330 PRINT&(14,10);:INPUT"SELECT. Drive ";Y$:IFY$=""THENRETURN
5340 GOSUB5370:S*Y$:GOSUB5370:RETURN
5350 :
5360 REM STOP/START DISK
5370 ZZ=NOT(ZZOR254):POKE49154,ZZ*-1:FORW9=1TO1200:NEXTW9:PRINT
5380 RETURN
5390 :
5400 DISK!"GO 2336":Y$=CHR$(PEEK(9059)OR32):Y=VAL(Y$):RETURN
5410 :
5420 PRINT!(28)&(16,12)"To RESTART type:-  GOTO 'line number'"
5430 POKE13026,128:PRINT&(16,14)"Bye !":POKE2073,173:GOSUB5370:END
5440 :
6000 PRINT:PRINT&(16,8)!(15):Y=21
6010 PRINT&(Y,10)"-> DISK ERROR <-----------":GOSUB5370:GOTO5060
```

★ ★

HEAD LOAD MODS FOR 5 1/4" DUAL DISK DRIVES

By: Ed Richardson
146 York St., Nundah 4012
Queensland, Australia

In the C4-MF head load modification published in PEEK(65) a few months ago, I mentioned that a more complex mod would be needed for a DF system. While the MF mods can be done on both drives of a DF system, both heads will be loaded even though only one drive is selected, doubling the noise created.

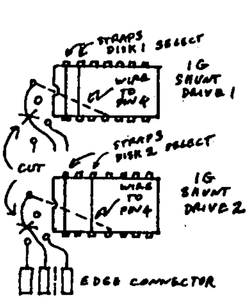
The DF mod requires much more expert track cutting and a difficult I.C. removal. This modification should not be attempted unless you are experienced at this kind of work. The tracks around the IC to be removed are very fine, and the board uses plated-through holes, meaning that it is easy to damage the board.

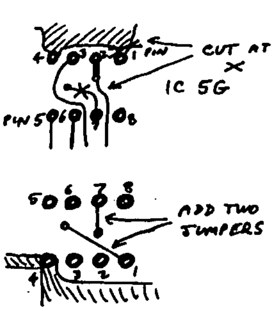
There are only two acceptable techniques for removal of I.C. 5G. The best one is to use a Pace desoldering station, an item which costs hundreds of dollars. This will remove the I.C. without damage. The other method is to first buy a couple of spare 75478 chips. Cut the pins off where they enter the chip on the board, and remove each pin separately with needle-nosed pliers and fine soldering iron. Following this latter procedure, the holes in the printed circuit can be emptied of excess solder with a solder sucker or solder taul (prefluxed copper braid). Both sides of the board are then cleaned with a cotton bud dipped in methylated spirits, to remove any remaining flux.

On the top of the board, two tracks must be cut. Under the board, two very short jumpers are added. These changes are shown in the drawings below. Be careful to solder the I.C. in the right way around. The same wiring changes are needed around the IG shunt as were done in the MF mod, but the actual shunt arrangement is different as shown below. Also, the activity light wiring is changed as in the MF mod, on both drives.

SEE SCHEMATIC ON PAGE 22.

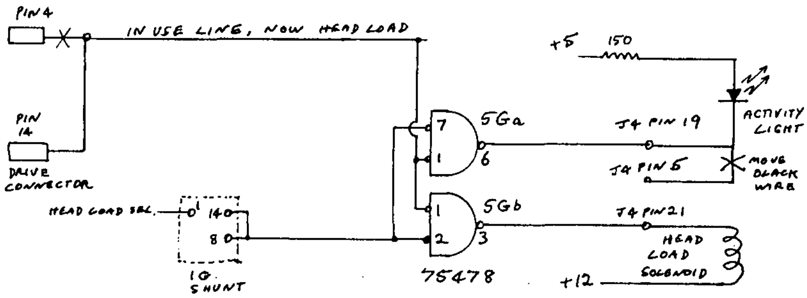
OK, what have we done? The modification around the 1G shunt area disconnects pin 4 from the IN USE line, which





After removing 1C 5G, perform two track cuts. Solder in 1C again.
(Top of board)

Under Board Below 1C 5G.



isn't used in the OSI set-up. It connects pin 14 which is actually the drive select for the third drive, marked DS2 on the printing near the shunt, just to be confusing. The IN USE line now becomes HEAD LOAD.

The modification around the 5G integrated circuit puts the two halves of the chip in parallel. The 75478 chip is actually two OR gates, but here the logic is used in the negative sense. Only when both inputs are low do the outputs go low. Output pin 3 will operate the head load solenoid, and pin 6 will be used to drive the ACTIVITY LIGHT which formerly was on whenever the drive was selected.

LETTERS

ED:

I've written to CompuServe for details for their services, and yes there is a cheaper way than owning the phone company.

I'm sure you must be aware that sending 300 b.p.s. data over a telephone link that can carry the information contained in speech is inefficient to say the least. Telephone companies know this and are well able to provide multiplexed, low-speed data channels at a cheaper rate than toll speech circuits. It's probably just a matter of knowing how.

To give you an example. In this country it would cost me $59 to send 108K to Leo Jankowski in Timaru via the telephone network. But if we were both registered as Pacnet users (a packet switched data network) which costs $75 joining plus $4.50 a month, I could send the same data for about $5!

Pacnet users in New Zealand have access to the USA for $12 per hour plus $12 per 64K as against normal telephone charges of $168 per hour.

I am quite sure that similar services are available for Australian and British subscribers, apart from yourselves within the USA. In New Zealand the Post Office provides and controls all communications - I work for them. In the UK I imagine British Telecom would be the people to approach, and in Australia, Overseas Telecoms. In the USA I wouldn't know - perhaps Bell, ITT, WUI.

We already have many subscribers in New Zealand who use Pacnet to access data bases all over the world and find it very useful and cheap. Obviously, this is business-related. Whether it is cheap for hackers like me is another story. I suspect that it will be for OSI SIG because there is no other source of software in this god-forsaken country - OSI is dead!

Ray Osborn
Rotorua, New Zealand

HUMOR!

Real Programmers don't write specs - users should consider themselves lucky to get any programs at all, and take what they get!

Real Programmers don't comment their code. If it was hard to write, it should be hard to understand.

AD$

*** * PEEK SOFTWARE LISTINGS * ***

Listings will be sorted by Basic, Type and Machine to make it easy for you to zero in on the programs that may be of interest to you.

Each listing will have an encoded "head line" that should tell you everything you need to determine if it is interesting and will run on your machine. Next, will be the program name and the author's name and address. Lastly, will be the author's description and any special comments. (See example below.>

B/1.43/2/82/MR/M/D/12/1000
WONDER ACCOUNTING SYSTEM
I. M. Crazy
123 Pecan St.
Funny Farms, NZ 12345
123-456-7890

This system will handle up to three A/R and four R/P accounts at one time. Complete record locking, provided that no more than one user at a time is active. Average storage space required is 8MB. Mammoth overhaul required to run on SSII cassette system. And that makes 10 lines.

....

For example, the first line: (B) it's a business package, under BASIC vers. No. (1.43) on a (2) C3A/B or 200 series with a minimum of (82) two 8" FD's. It's (MR) Multi-user with Record locking, supported by (M) modem and sold by (D) dealers. There are (12) 12 x 10 = at least 120 copies in use at (1000) bucks retail.

Just "X" the appropriate boxes and fill in the remaining blanks on the form. If you have more than one program to submit (we certainly hope that you do), please feel free to make photo-copies of the form - one for each program!

The hard part will be writing a description that will not exceed PEEK's physical limits (10 lines, each not to exceed 30 characters). We would like to give you more room, but 10 lines of carefully chosen words should be adequate to whet the appetite.

If your software is not directly supported by you, the author, please fill in the DEALER ADDRESS as well as your own address block. If both blocks are filled in, only the dealer address will appear in PEEK. The dealer address may be either the selling dealer or an address where those inquiring may get a list of vendors.

FREE PEEK(65)
SOFTWARE LISTING FORM

PROGRAM NAME ________________________________

BASIC - VERSION /_/_/_/_/

/D/ OS65-D   /U/ OS65-U   /C/ CP/M

CATEGORY

/G/ GAME   /B/ BUSINESS   /U/ UTILITY   /O/ OTHER

MINIMUM COMPUTER

/1/ SB,SBII, C1P,C2/4P   /4/ C4P   /8/ C8P   /O/ C2/3OEM

/D/ C2/3D   /2/ C200,C3A/B   /3/ C300

MINIMUM STORAGE REQUIREMENTS

/C/_/ CASSETTE   /5/_/ 5 1/4"   /8/_/ 8" FD   /7/_/ CD-7

/2/_/ CD23/28/36/74   (USE 2ND BOX FOR NO. UNITS REQUIRED)

TERMINAL SYS.   /S/ SERIAL   /V/ VIDEO   /B/ BOTH

SYSTEMS SUPPORTED - MAX 2 CODES - RECORD LOCK ASSUMES MULTI-USER

/S/ ONE USER   /M/ MULTI-USER   /H/ HARD DISK   /R/ RECORD LOCK

SOFTWARE SUPPORTED BY:

/D/ DEALER   /P/ PHONE   /M/ MODEM   /N/ NONE

/O/ OTHER

SOLD BY

/A/ AUTHOR   /D/ DEALER   /M/ MAIL ORDER   /O/ OTHER

COPIES IN CIRCULATION (# / 10); 1 = <11, 11 = 100-110 /_/_/

PRICE, RETAIL, PLUS TAX & HANDLING OR SASE $/_/_/_/_/NO CENTS

NAME AND ADDRESSES

AUTHOR	**SELLER** (or SAME)
NAME ____________________	NAME ____________________
STREET ____________________	STREET ____________________
CITY ______________	CITY ______________
STATE /_/_/ ZIP ________	STATE /_/_/ ZIP ________
PHONE (___) ___-____	PHONE (___) ___-____

DESCRIPTION - MAXIMUM 10 LINES OF 30 CHARACTERS

1. ________________________________
2. ________________________________
3. ________________________________
4. ________________________________
5. ________________________________
6. ________________________________
7. ________________________________
8. ________________________________
9. ________________________________
10. ________________________________

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:

1/2 PRICE INVENTORY SALE

OUR SHELVES ARE BULGING!

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

Get a one year volume set (12 back issues) for $15.00 and we will pay UPS.
Get one back issue of the OSIO newsletter free with order.
Foreign orders by VISA/MASTER/CHOICE only, plus postage.
Orders **can not** be sent to P.O. Boxes.

NAME.......................... **STREET**..........................

CITY.......................... **STATE** ....

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

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

<u>Vol 2. 1981</u> ( )

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

<u>Vol 3. 1982</u> ( )

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

<u>Vol 4. 1983</u> ( )

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

<u>Vol 5. 1984</u> ( )

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