BASIC + DOS
TABLES

APRIL 1985
VOL.6, NO.4

PEEK (65)

The Unofficial OSI Users Journal

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

INSIDE

Column One

You may have to dig around a bit in this issue to find all the good things that you can make use of, but do it - all the way through, including Paul Rainey's letter. The most important part, to us, are the overdue results of the Reader Survey that appeared in last November's issue. Although I am sure that various sections of it will elicit a number of comments and deductions from you, our immediate concern is the list of articles that you would like to see published in PEEK. Now we know what you want to hear about, but the irony of it all is that there probably isn't one topic on the list that one or the other of you couldn't write about. In these days, more than ever, we (the OSI community) need the benefit of your experience and knowledge. The thought of so many of you diligently re-inventing the same proverbial wheel and the waste of that kind of duplicated effort should be enough to make you set pen to paper or keyboard to disk. If you need assistance of any kind, give us a ring or drop us a line. We are optimistic that that a concerted effort on everyones part will improve the dismal 30% support level that we currently have.

On the manufacturing front, there seem to be lots of interesting projects in the works, but the fear of unfulfilled promises and unmet deadlines seems to have taken on new meaning within the manufacturing community. DBI's new machine is still in beta testing and won't be released until they are completely satisfied. OSI is about to make its splash with their Portland Board - that's a one or two user, multi-processor board with 64K per user and running at 4 MHz. It purports to run all OS-U programs and cohabitate with timeshare users even if it does require that the base user be dedicated to being the systems manager. On top of that, another product announcement is expected next month. Technical Support has been moved back to Aurora, for both hardware and software, and they can be reached at 800-321-5805. There is more, but not just yet!

Our phone rings again and again with questions about RS-232 hook-ups. That's why we asked Brian to jump off the track a bit this month and give us the technical reasons behind all of those tangled cables sometimes required to get your printer or modem up and running. Hopefully, now you will be able to "reason" out your problems. If there are other areas where immediate attention is needed, please let Brian know. You can reach him right here through PEEK(65).

We hear more and more about SASI interfacing and now Joseph Ennis has started the ball rolling. Yes, his article talks about the C1P, but it shouldn't take a great deal of effort to accommodate other machines as well. We also hope to hear more from Joseph on his other mods in the near future and the same holds true for Paul Rainey's mods. Between them, they could cause one to salivate excessively.

Don't glance over The Beginners Corner too quickly. Yes, it is simple and basic, but there isn't a programmer who cannot profit from the concepts it contains.

Whether it be for your sheer enjoyment or fun for the kids, have a bash at Mini-Logo. We did. The program structure is worth studying, if nothing else.

Just for "U" guys, we threw in a tip about padding DMS files. Rather than becoming a "U" tutorial, we hope that it will spur you to contribute some thoughts and tips of your own.

So there you have it, but it's what is coming in the near future that has us excited. Stay tuned!

Eddie

THE INSIDE STORY

PART 2

Printer and modem hook-ups seem to be many people's Waterloo. This month, Brian gives you the explanation of what happens at the OSI end and thus you will know why pin X has to be connected to pin Y.

By: Brian Hartson
Tech. Editor.

From time to time there have been many questions and much time spent on getting printers to work with the OSI computer. I will try to answer and put to rest this subject once and for all.

In the printer world there are basically two types of interfaces, parallel and serial. The parallel interface was somewhat standardized by the Centronics Printer Company some years ago. While the serial interface, although it adheres to the RS 232C standard, it does so very loosely and not every printer manufacturer is the same.

Now to some explanations. The parallel printer interface, while the hardest to implement is by far the most used and can be the fastest. When purchased "off the shelf," these interfaces usually work; while on the other hand a serial printer frequently does not. The reason for this can be seen as we progress through how each works.

The parallel interface works in the following manner, the interface presents to the printer all eight bits of the character at the same time. When all eight bits are present the interface then sets a signal called STROBE. When the printer sees this line set it accepts the data and then sets ACKNOWLEDGE. ACKNOWLEDGE, when it sets, causes STROBE to be reset and the process starts all over again. The preceding explanation is basic and shows only the basics of the Centronics Parallel interface. Listed in Table 1 are the signals that make up the Centronics Parallel interface.

TABLE 1

PIN #	SIGNAL NAME	DESCRIPTION	NEEDED
1	STROBE	LOW PULSE USED TO STROBE DATA INTO PRINTER	*
2	DATA 1	DATA BIT 1	*
3	DATA 2	DATA BIT 2	*
4	DATA 3	DATA BIT 3	*
5	DATA 4	DATA BIT 4	*
6	DATA 5	DATA BIT 5	*
7	DATA 6	DATA BIT 6	*
8	DATA 7	DATA BIT 7	*
9	DATA 8	DATA BIT 8	*
10	ACKNOWLEDGE	LOW PULSE USED TO SIGNAL INTERFACE THAT PRINTER HAS ACCEPTED DATA	*
11	BUSY	SIGNAL INDICATES PRINTER BUSY	
12	PE (PAPER EMPTY)	HIGH SIG. FOR PAPER OUT	
13	SLCT (ON LINE)	HIGH SIG. PRINTER IS ON LINE	
15	CLOCK	100-200KHZ CLOCK	
16	LOGIC GROUND		
17	CHASSIS GROUND		
18	+5 VOLTS	APPROX .5 AMPS AVAIL.	
31	RESET	RESET PRINTER	
32	FAULT	LOW INDICATES OFF LINE PAPER OUT OR OTHER FAULT	
33-36	NO CONNECTION		
14,19-30	SIGNAL RETURNS		*

Let's take a look at what happens during a print cycle between the interface and the printer. We will assume that the printer is ready and on line. The interface puts the data bits on lines two through eight. After waiting approximately 200 nano-seconds, the interface sets STROBE low. The strobe pulse must remain low for at least 500 nano-seconds but no more than 500 micro-seconds. The printer will then respond by setting BUSY within 200 to 600 nano-seconds of STROBE resetting. ACKNOWLEDGE sets 300 to 2000 nano-seconds after either the rising edge of STROBE or the falling edge of BUSY. The printer has just accepted the character that the computer has sent to it and is ready for the next one. When the printer buffer is full or the printer is printing the line BUSY will stay set and delay ACKNOWLEDGE, thereby causing the current data to stay on set on the data lines. By looking at the timing diagram one can see the relationship between all these signals.

TIMING DIAGRAM

```
DATA                      ------                     ------
        ------------            ------------------         ----------------

STROBE  ---------------     ----------------------    -----------------
                       ---                        ---

BUSY                      -------                    -------
        -----------------         -----------------         --------

ACK.    ----------------------  ----------------------  --------
                              ---                     ---
```

Now to some practicalities. Although all the control and handshake signals are available, OSI does not make use of any of them in their software. So that if you are wiring up your own board or cable, the only signals that you need worry about are those that have been starred in Table 1. Table 2 contains the cable connection for the OSI parallel printer interface. If you have not seen this board, it is the same board as the Floppy Disk Controller (470). Also, this interface is on the model 555 board.

TABLE 2

PIN #	SIGNAL NAME
1	DATA 8
2	DATA 7
3	DATA 6
4	DATA 5
5	DATA 4
6	DATA 3
7	DATA 2
8	DATA 1
9	NC
10	NC
11	NC
12	NC
13	NC
14	+5 VOLTS

TABLE 2 continued

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

15	-9 VOLTS
16	NC
17	STROBE
18	RESET
19	NC
20	FAULT
21	SELECT (SLCT)
22	PE (PAPER EMPTY)
23	ACKNOWLEDGE
24	BUSY

Now, on to serial printers. With serial printers things get more difficult. This difficulty arises because there is no real standard that the printer manufacturers use, although they all try to adhere to the RS232C standard in one way or another. Most serial printers use a basic three wire system like the following: one wire for data, one for control, and the last for a ground.

That may sound easy but it is not. What happens in practice is that the control line could be one of many signals, and there are other signals that, although they are passive, must be at a certain level or all bets are off. Let's take a look at the RS232C, also called EIA connection list in Table 3.

TABLE 3

PIN	SIGNAL NAME	SOURCE	DESCRIPTION
1	FRAME GROUND		CHASSIS GROUND
2	TRANSMIT DATA	PRINTER	DATA FROM PRINTER
3	RECEIVE DATA	COMPUTER	DATA FROM COMPUTER
4	REQUEST TO SEND	PRINTER	HANDSHAKE LINE
5	CLEAR TO SEND	COMPUTER	INPUT OF REQUEST TO SEND
6	DATA SET READY	COMPUTER	COMPUTER READY
7	SIGNAL GROUND		SIGNAL RETURN
8	CARRIER DETECT	COMPUTER	COMPUTER READY
14	BUSY	PRINTER	PRINTER BUSY
19	BUSY	PRINTER	PRINTER BUSY
20	DATA TERM. READY	PRINTER	PRINTER AVAILABLE

Before some of you crucify me, let me say that Table 3 is not really a list of the standard RS232C connections, but rather how the printer manufacturers see it. As you can see, there could be great confusion in hooking up a printer to a serial port. Not only do the printer manufacturers make it hard, but so does OSI. Let's talk about why this is so. For you 4P'ers, the process of hooking up a serial printer is really hardest because of the lack of a minus power supply. The RS232C data transmission requires that the data signal vary around zero volts in both the positive and negative direction. This is so, so that ones and zeros can be transmitted down a single wire. Ones are generated by what is called a MARK, a voltage level below zero volts (negative) and zeros are generated by SPACES, a positive voltage. By combining MARKs and SPACEs it is possible to transmit data over a single pair of wires, the other being signal return.

The rest applies to everybody. OSI in its finite wisdom decided that none of the asynchronous control lines should be made available to the user at a board connecter. Instead, they connected them to ground on the board. So now not only do some of us have to add a power supply, we must also cut and patch boards. First, let's take a look at what is needed on the 502 and 505 boards that are used in the P machines. For those of you with cassette based machines, the addition of a serial printer will mean that either you purchase a serial interface board or, that you modify the cassette interface. For those with the disk based P machine, i.e., you have a 505 CPU board, the following can be done. All further discussion makes reference to the Sams manuals that are available from PEEK. First, you must add the minus power supply that was talked about earlier. This supply doesn't have to be very large, it could be a battery, a battery eliminator or even a power supply. This power supply is connected in the following manner. Hook the minus side of the supply to backplane pin number 24, and the plus side to backplane pin number 27 or 28. You must also make sure that backplane pin number 24 is connected to backplane pins number 27 and 28 on the 505 board. Next, locate two 1000 ohm resistors R41 and R39. One end of these two resistors are connected together, then connected to ground. These two resistors must have their ground end tied to the minus supply as indicated in the drawing (in the Sams 4P manual). Next, locate W19 and W20, cut W19 and wire pin 24 of U1D to W20, that is to ground. Now, locate W21 and W49, cut W21 and wire pin 23 of U1D to W49. This completes the control handshake connections. You must also insure that your CPU board has U1G installed. After all that is done now comes the hard part. The cable that connects the printer to the computer must be wired correctly or again all bets are off. On the back of the C4P is a connector board; locate J8 and its pinout as follows:

PIN# DESCRIPTION

3	DATA FROM COMPUTER TO PRTR
7	SIGNAL GROUND
8	PRINTER BUSY HANDSHAKE

Now your printer will determine how the cable is made. Please refer to the manual that came with the printer for correct connections. The most common hookup is as shown below:

```
PRINTER                    COMPUTER

PIN NO.                     PIN NO.
 3     ----------------     3

 4     ----|
 5     ----|

 6     ----|
 8     ----|
20     ----|-----------     8

 7     ----------------     7
```

If you don't understand the chart, it goes as follows: connect pin 3 to pin 3, pin 7 to pin 7. At the printer end connect pin 4 to pin 5, pin 6 to pin 8 to 20, then connect them to pin 8 on the computer side. This is a fairly common hookup, however, that is no guarantee that the printer manufacturer didn't do something just a little different. Now, as to the rest of the OSI world a serial hookup is just about the same except that you don't have to add the power supply. Depending on what board you are going to use will determine what cuts and jumpers that will have to be made.

For the sake of simplicity and the fact that the mods will be the same for most any board, let's say that you are going to use a 550 board as your serial printer interface. This board has signals going to molex connectors across the top. You will need to make two cuts and one jumper. First, pick the port that you are going to use. Next, make sure that the 1488 and 1489 (75188, 75189) are on the board for that port. Next, locate pins 23 and 24 of the 6850 chip. You will notice that both pins are tied to ground. Cut either pin 23 or 24 from ground but not both. Now locate pin 2 of the same 6850. Cut the trace that is connected to pin 2 of 6850. Now connect pin 23 (or 24) to the trace that you cut away from pin 2 of the 6850. Connect the printer as described above with the handshake pin going to what used to be "Data Into" the computer.

Well, that just about sums up printers on the OSI computer. If you have any questions, you can contact me through PEEK, or leave a message for me on CompuServe ID #75026,3022.

★

SASI DISK INTERFACE FOR OSI USERS

A simple connector wiring change that allows access to IBM PC hardware.

By: Joseph Ennis
212 20 Street
Niceville, FL 32578

This article is dedicated to all OSI users that are still using a Superboard or C1-P. (I would really like to know how many of you are out there, so would welcome a letter). This article is a hardware article that describes a modification to the OSI 610 board so it can use the low cost disk drives that are intended for the IBM PC. This mod requires only the fabrication of an adapter cable and the adding of a data separator (which is needed anyway) to achieve access to drives like the Tandon TM100-1 ($90.00 new fall '84 or $75.00 used).

I bought my Superboard in the fall of 1978, and got along fairly well using it as a cassette only system. By the fall of 1982 and summer of 1983, I noticed a lot of OSI Software Houses disappearing, so I decided that I would finally have to spend money and buy the programs that I had always intended to buy but had put off because of lack of money. The bad part was that most of these programs were on disk and the money problem also affected the purchase of disk drives (which I did not have). During this period the lowest cost disk drives were coming from IBM PC owners, who were converting from single-sided to double-sided drives. This caused me to convert to a disk drive interface that is hardware compatible with the IBM PC. This article is a description of how I interfaced my OSI to a non-standard drive.

I already had a 610 board (an OSI board that contains 24K RAM and the OSI disk controller circuitry (if you need one, you can buy the bare board from Isotron (216-562-4136) for $5.00). Not counting the RAM chips, there are 23 other chips on the 610 board; one is a 6520, another a 6550 and the rest are 7400 series SSI that average under 50 cents each. I bought my 610 board back in 1979 but have been using it only for the memory sockets. Also, when I got it I removed the disk drive molex connectors, as I considered it very sloppy in design to have connectors in the middle of the board. Another perculiarity of my OSI computer system is that I raised my system clock up one divider tap, so my 02 clock is running at 1.96608 MHz (I also implemented a true Kansas City tape interface, switchable to 300, 600, or 1200 baud as well as making these baud rates available to the printer and modem serial ports - but that is another article). This higher speed clock gave me some concern as I started my modification, and various friends commented on the difficulties I would have with the interface, but I did not want to remove the clock mod as I had come to like my computer taking only half the time to do something. In the end, the clock speed difference turned out to be no problem. I bought two single-sided drives in used, but good condition from two different IBM PC owners who converted to double-sided drives. I also bought a dual drive cabinet with power supply, three connectors, five feet of 36 conductor ribbon cable, and a copy of HEXDOS. So that's my environment when I started to interface the TM100-1 drives to my system.

First, I built the data separator (see Figure 1 for the printed circuit card layout, - for the Data Separator schematic see the upper center part of Figure 2). When I built the Data Separator, I was at a low point for test equipment, none! The basic circuit is a popular one with OSI users, the 600 board cassette port uses this circuit and a couple of years ago there was a data separator article in PEEK(65) using this circuit. The only change I made is to form the timing circuit out of a precision resistor and capacitor rather than potentiometer and capacitor. A 1% tolerance resistor, a 3% capacitor and another 1/2% tolerance for the 74121 gives a circuit that will work without requiring test equipment for alignment. The design appears to be satisfactory for any 125K bit per second disk system, and several Data Separators have been built and successfully used with different brands and sizes of drives.

My second step was to modify my 610 board. The interface could be achieved without soldering or cutting the 610 board, but I changed the 610 because I had removed the molex connectors and I wanted my finished work to be compact and neat. I had always intended for my disk drive connectors to be on the edge of my 610 board. I achieved this by installing the 37 pin female D connector on the edge near U72 (the 6520). The connector that I used is a combination of a solder cup female and the right angle hardware that I had removed from a right angle PC connector. I carefully drilled holes in the edge of the 610 board (through the power bus tracks, they lost some copper, but no measurable drop in voltage across the 610 board) and mounted the connector using nylon screws (see Photograph 1). I wired a short 6 inch piece of ribbon cable to the connector before I mounted it, then connected the wires from the ribbon cable to the appropriate holes in the 610 board where I had removed old J3 long ago. Refer to Table 1 for the connections:

TABLE 1

New J3 (37D)	Old J3 (holes)	Signal Name (IBM PC convention)
6	17	Index Detected
7	4	Motor C Enable
8	18	Drive D Selected
9	3	Drive C Selected
10	4 (yes 4 twice)	Motor D Enable
11	6	Direction In
12	5	Step Pulse
13	NC	Select Side One
14	8	Write Gate
15	23	Track Zero Detected
16	19	Write Protected
17	to Data Sep	Read Data
18	9	Write Data
20 thru 37	6 and 7	Sig and Power Gnd.

Now the Data Separator Card has the wire from J3 pin 17 connected to its input along with a ground from J3. The outputs of the Data Separator are connected to the 610 board. Refer to Table 2 for the connections:

TABLE 2

Data Separator Card	Old J3 (holes)	Signal Name
Data Out	2	Separated Read Data
Clock Out	4	Separated Clock Out
Ground	6	Signal & Power Ground
+5 Volts	5 Volt Pickup on the 610 Board	

Now the wires from the Data Separator are dressed and the Data Separator Card is attached by the same two nylon screws (4-40 x 7/8) that are holding the new 37 pin D female connector. That's really all there is to this Mod.

For my special environment I had to move the XMIT CLOCK jumpers (near U12, 74LS93) on the 610 board down one tap (to the pad that connects to pin 8 of U12). This is so that my clock to the disk will be at 125KHz. Also, to support the disk time-out with HEXDOS I needed to add the jumpers as called for in the HEXDOS manual. This involves two of the four square pads that go to U72, the PIA. On the square pad nearest U10, I placed a jumper to the square pad that connects pin 9 on U11 (74LS390) to the second pad from the end closest to U10, (this pad connects to U72 pin 19) (this jumper provides a timing pulse to the PIA every second). There is a cluster of three square pads forming a small triangle near pin 40 of U72. I placed a jumper from the center pad (nearest old J3) to the pad that is nearest to U72 pin 19, this pad connects to U72 pins 37 and 38 (this jumper ties the NMI line to the PIA pins 37 and 38 which allows the PIA to interrupt the CPU on timeout). These changes are not necessary to get the disk to work, but are nice to have. Another nice to have mod which I included, is the Motor Control Circuit. To make this mod, cut the lead running between pin 14 of U72 and pins 1 and 13 of U75. Connect a jumper from U72 pin 14 to U5 pin 3. Connect a jumper from U5 pin 4 to U75 pins 1 and 13. This adds an inverter into the motor control line. This inverter is needed because OSI messed up the PIA initialization routine in the ROM. As coded in the ROM, this line is HIGH when it should be LOW and LOW when it should be HIGH.

One of the advantages of being hardware compatible with the IBM PC is that I can take my disk drives to a friend who has a PC, plug the cable into the disk expansion port on the back of the PC and the PC will address these drives as drive C and D (if all DIP switches are set right). This allows the drives to be checked by the IBM PC hardware diagnostic program.

A word on the DIP switches on the Tandon drives. The DIP in socket 2F is the terminating resistors. The rule for dual drives is to use only one set of terminating resistors and put these only in the drive furthest from the 610 board. In actual practice, try both of them in and out. Usually a better operation is found if both are in. The 610 board has more than enough capacity to drive them both. The DIP in socket 1E is the switches that set up the device addresses per Table 3:

TABLE 3

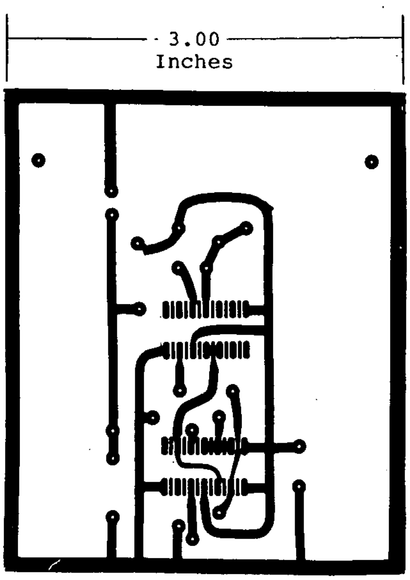
Switch	Function Name
1	Leave Open
2	Closed = Drive A Selected
3	Closed = Drive B Selected
4	Closed = Drive C Selected
5	Closed = Drive D Selected
6	Leave Open
7	Leave Open
8	Leave Open

Future Work - The disk controller hardware could control double-sided drives without any additional changes to the hardware. The only change is to the software to toggle PB1 to select either side 0 or side 1. Another change that is possible is double density. With the mod that doubles the 02 clock frequency, the system has enough cycles to support double density (250K bits per second) format. Both densities could be supported if the XMIT CLOCK and the capacitor in the Data Separator are switched. These mods would allow double and quadruple storage density over the current OSI 5-1/4" format. The Tandon drives don't care, I have already verified that they will run equally well at either bit rate. Another change that looks attractive is the parallel printer port; a 25 pin female D connector could be added beside the new 37 pin D connector and the PIA pins could be jumpered across. This would give a Centronics port, and if the 25 pin D connector is wired in IBM PC format, low (relatively) cost IBM printer cables could be used. HEXDOS already has a parallel printer port driver, the driver's target address can be changed with either a POKE or a permanent change to HEXDOS. The only real software fix needed is to correct the sense of the Motor On line (PB4) so the inverter could be removed from the Motor On line, this would simplify the printer connection. All the lines PB0 thru PB7 could go straight to the 7417 drivers and then on to control both the floppy disk drive and to the parallel printer input (Centronics pins 2 through 9). A PIA input line like PA2 would be a good line to use for STROBE (Centronics pin 1). PA3 would be a good input line to use for ACK (Centronics pin 10). PA4 would be a good input line for BUSY (Centronics pin 11). These three inputs are not used by the OSI Disk Controller.

The Past - Yes, I had problems installing this mod. It took me one month to design and install it (Oct-Nov 84), then three months to get the bugs out. My biggest problem was slow memory, I had some 300 nsec chips that could not handle the speed under all conditions. The second biggest problem was I either received a bad HEXDOS disk or I damaged it shortly after receiving it. The third problem is that the best looking of the two used drives that I bought was actually worn out, the stepper motor shaft locking collar was worn so it would not stay locked. Needless to say, all of these problems interacted. The mod I made prior to the disk interface was the 02 Clock frequency change and the mod prior to that was the cleanup of the BASIC Garbage Collector problem. The problem I cleared first was the slow chips, they checked OK with the OSI MEMORY FREE? but I had occasional failures in service, this included the video RAM. I wrote several programs to check what the disk was doing and what was messing up, this is how I found that a slow RAM at high memory would cause a glitch in the disk directory as it was being saved in RAM. This was a tough one, I was blaming this on the drives and HEXDOS. The best program I found for checking RAM memory was the "Mary had a little lamb" program that PEEK(65) published a couple of years ago as part of the test for the Garbage Collector problem. The concatenation of the string line "Mary had a little lamb" turned out to be an excellent memory test pattern, I made a slight mod to be able to tell where in memory the CPU was writing when the text started picking up bad bits. This problem was aggravated by my buying a new set of 8K to finish populating my RAM sockets on the 610 board, these new RAM proved to be mostly too slow. RAM gets slower with lower voltage and higher temperature. Next, I wrote a program to look at the raw bytes on the disk to see why the disk was messing up. I had two problems, one was the first one or two bytes on the track would be missed, especially on the lower track numbers. The other problem was I would lose the ability to read the higher number tracks which I had been able to read and write to OK only weeks before. Also, for some reason, the higher number tracks of my orignal HEXDOS disk were just a repeat of some of the lower track data instead of what was supposed to be on them (not pure copy, just kind of jumbled together). The first problem I fixed was removing the slow RAM, this sure helped a lot. The next was borrowing a copy of HEXDOS while I sent my copy back to Steve Hendrix to be refreshed. I also wrote Steve every week during this period (about eight letters). I learned a lot about HEXDOS. Once I had the RAM and HEXDOS out of the way I was still getting the problem of not being able to read tracks I had written weeks before. I suspected it was a mechanical problem and swapped the A and B drives. Booting suddenly got a lot better. Took the drive that appeared to be the trouble maker to the disk repair shop, $35.00. It held for about five weeks, during this period I had gotten my copy of HEXDOS back from Steve, and had a several week period of no problems.

Continued on page 8

FIGURE 1
Data Separator Printed Wiring Master



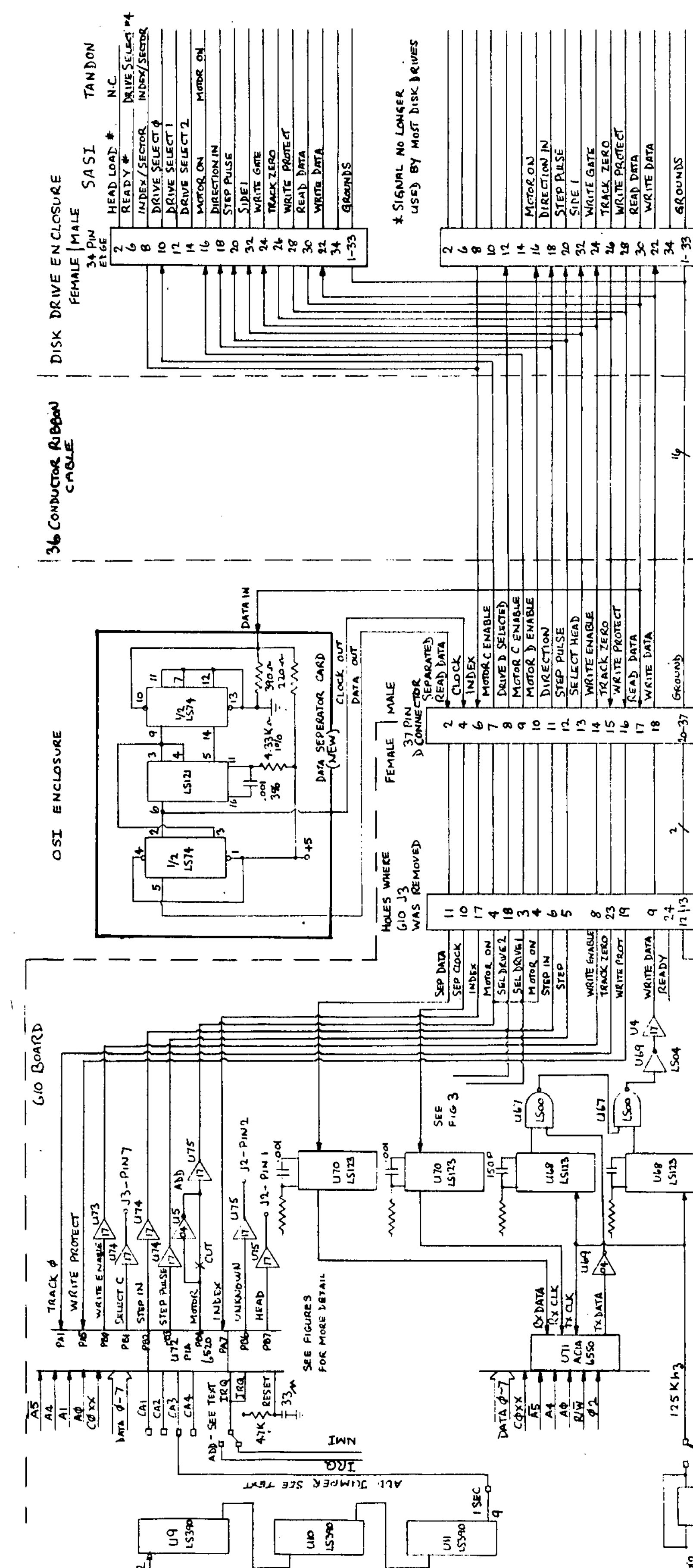


FIGURE 2
OVERALL SCHEMATIC OF
DISK CONTROLLER INTERFACE

Then I started slipping again, I tried aligning and tightening it myself, I even put fingernail polish on the shaft to lock it. It didn't hold so I threw the drive in the trash after removing the electronic board and a few small screws. I bought another for $75.00, and I can claim that I have gone a full 12 months without having a single disk problem. Not even a single bad load!!! During the time I was trying to debug this mod, I even tried shorting the ribbon cable and moving the data separator from under the 610 board into the disk drive housing. I can report that the data separator appears to work equally well at either end of the cable. Also, I could not tell any difference in the performance between a ribbon six feet long and four feet long.

FIGURE 3
THE PIA ON THE 610 BOARD

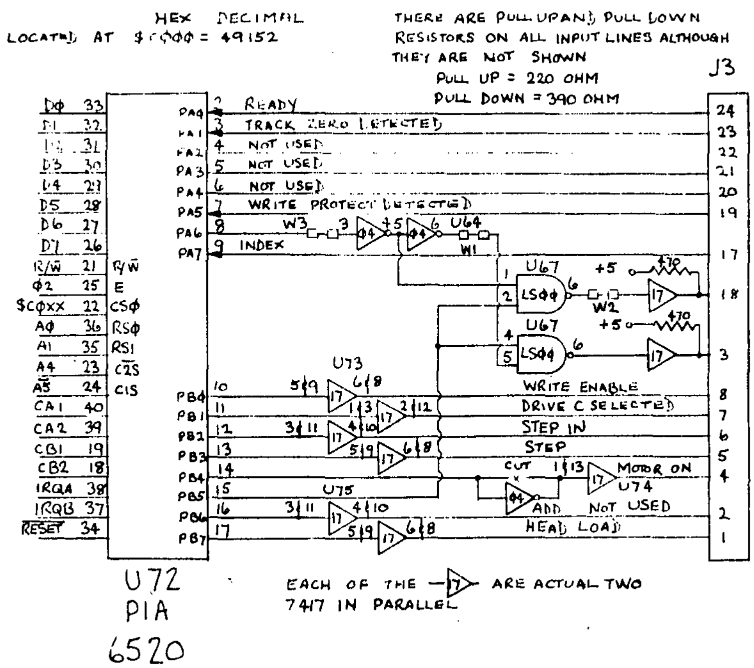
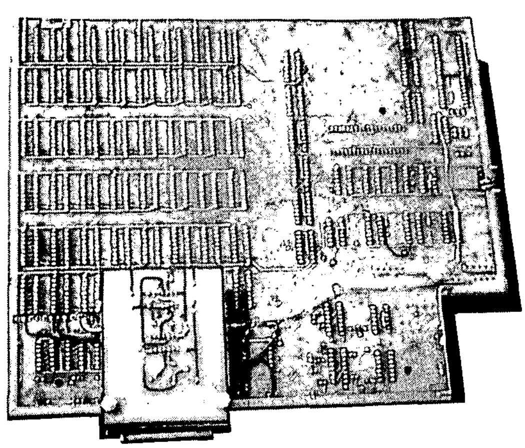


PHOTO 1
Data Separator Card
fastened to 610 board



MAPPING MACHINE LANGUAGE CODE

In case you haven't already guessed, the material presented in the last two issues under this heading represent a veritable mountain of work on the part of its author, Tom Berger and also Dana (Skip) Skipworth who has been an invaluable aid in coordinating this complex effort.

In spite of the best efforts of all, several errata should be noted in Part I which appeared in the February issue. The first is on us and the remainder are repeats of errors in the original publication. It has also been brought to our attention that the originally published article omit-

ted the three following tables which you will need.

Page 3, mid. col., last par. In the branch table, the middle letter of a branch instruction closest to...

Page 4, 1st Col., last line. Track N-1(N=9 on 8" floppies).

Page 6, 3rd col., PASS Four.
610 PRINT #6,"E"
620 PRINT #6,"E"

Page 6, 3rd col.,ML list.
232D 9005 00

Page 6, 3rd col., last sent. These values trick the disk...

OSI Microsoft Disk Basic
JMP AND JSR TABLE

```
.      OSI Microsoft Disk BASIC
.
.         JMP AND JSR TABLE
.
.
.
.
.
.     Jump vector for commands
0003   S047A
.
.      Jump vector for evaluation
006F   M0D84
.
.      Jump vector for functions
00A1   S0E83
.
.     CHRGET subroutine:  get BAISC character
00C0   S1615 S1C05 S1C12 S1C35 S2163 M220B S2259
00C0   S0484 S06D0 S079F S07E0 M07FD S0960 S09A0 S0AC1 S0B8E
00C0   S0CB3 S0D00 S0DB6 M0E1B S0E4D S0F48 S0F53 S0F7B S103D
.
.      Subentry: get previous character
00C6   S1652 S16A3 M2160 S21B0 S21EB
00C6   S0CAC S0CE7 S0F28 S0F30 S0F37 S108A S12C5 S15B5 M1624
00C6   S06C7 S0798 S089D S092C S0941 S0A32 S0B7B S0BC1 S0BDD
.
.      Search stack for FOR and GOSUB activity
03A1   S074F S08D9 S0C58
.
.      Open space in memory
03CF   S0504 S0FF1
03D6   S14A2
.
.      Test stack depth
0412   S075D S088B S0CDD
.
.      Check available memory
041F   S03CF S10EC S1142
.
.      Send error message then:
044C   M1194
044E   M1A87
044E   M085B M08E6 M0CCA M0E20 M10D2 M1232 M1352 M14D4 M1821
.
.      Warm start BASIC
0462   M20DA
0469   M084D
.
.      Wait for BASIC command
0474   M20D4 M21A7
047D   M052C
.
.048A  Handle new input line
.
.052F  Rebuild chaining of BASIC lines
.
.      Main input routine
0558   S047D M0B55
.
.      Input a character and erase if necessary
0587   S055A
.
.      Crunch keywords into BASIC tokens
05A7   S0490 S0499
.
.      Search BASIC for a given line number
0633   S049E S06C4
0637   S08C1
.
.0662  NEW
.
0678   S051A M2226
.
.067C  CLEAR
.
067E   S0883
0695   S0462
.
.      Reset BASIC execution to start
06AB   S0678
.
.06B9  LIST
.
06BD   M219E
06C1   M219B
.
.0748  FOR
.
07B4   M07E6 M08A3 M0CA4
07E0   M0493
.
.      Execute BASIC statement
07E9   S07E3 M0949
07EB   M095D
.
.      RESTORE
080A   S0692
0814   M0C16
.
.      Check stop key (Control C)
0819   S06EC S07B4
.
.0828  STOP
.
.082A  END
.
0837   M0B4C
.
.0853  CONT
.
.086D  NULL
.
.087E  RUN
.
0883   M222B
.
.0889  GOSUB
.
089D   M0886
.
.      GOTO
08A6   S08A0 M0946
.
.08D3  RETURN then:
.
.      DATA
08F9   S125D
08FC   S0C04
08FD   S07D8
.
.      Scan for next BASIC statement
0907   S0760 S08F9 S0BE8
.
.      Scan for next BASIC line
090A   S08A9 S093C
.
.0929  IF (and perhaps:)
.
.093C  REM
.
.094C  ON
.
.      Input fixed point number
096C   S0496 S06C1 S06D3 S08A6 S0963
0972   M09A3
.
.      LET
09A6   S074C M0800
09C3   S0BBE
09DB   S0BB3
0A17   M09FE
.
.0A2F Do a PRINT (from 21AA)
.
0A35   M21AD M21BF
0A3B   M21CE
0A4D   M21C5
0A6D   M0584
0A73   S0450 S0555 S06EF S0A62 S0A98 S0AFD S21A1 S21D1
0A7F   S0AE3
0AC1   M0A9B
.
.      Print string from memory
0ACC   S0B20 M0C23 M1CE9 M20FC
0ACF   S0A2F S0A65 S0B3A
0AD6   M0AE6
.
.      Print single format character
0AE9   S0A68 S0AC7 S0B52
0AEC   S0453 S0B4F S0B84
0AEE   S0459 S045F S054C S0552 S057F S070E S0743 S0A77 S0A7C
0AEE   S0A87 S0ADB
.
.      Handle bad input data
0B0D   M0BCA
.
.0B2C  INPUT
.
.      Prompt and receive input
0B4F   S0B87 S220E
.
.0B58  READ
.
```

Continued

```
0B64  M0BE5
0B8E  M0C0B
0BC1  M0BB6
.
.0C27-0C4A  Error messages
.
.0C4B  NEXT
.
0C51  S0CB6
.
.       Check type mismatch
0CB9  S077B S07A2 S1040 S12BC S1618 S1666
0CBC  S0778 S0D25 S0D90 M0E86 S1245 M1273 S1282 S12E9
0CBE  S0E5D S14C0 S151D S213D S225F
0CBF  S09BD S0EB9
.
.       Evaluate expression
0CCD  S0929 S09B8 S0CB9 S0E0A S0E57 S213A S21C2 S225C
0CD8  M0D5C
0CE7  M14F1
0CEA  M0D03
0D29  M0E27
0D4F  S0D29
0D62  S0D57
0D67  S07A8
0D72  M078E
0DB2  S0CE0 S14BD
0DD3  S0B32
.
.       Evaluate expression in parentheses
0E07  S0E74 S127F
.
.       Check right parenthesis
0E0D  S1093 S1248 S15D9
.
.       Check left parenthesis
0E10  S0E07 S0E54 S123B
.
.       Check comma
0E13  S0BE2 S0E5A S0F21 S15BC S166C S2196 S21BC S229C
0E15  S0775 S0935 S09AF S0B37 S124D S1265
0E1E  M0807 M08E9 M0AB1 M0B19 M0D5F M0F3F M12CA M2183 M227C
.
.       Syntax error exit
0E1E  M22C2
.
.0E23  Setup for functions
.
.0E2A  Setup variable name
.
.       Setup function references
0E4A  M0E04
0E79  M0E71
.
.0E89  OR
.
.0E8C  AND
.
.0EB9  Do comparisons
.
0F04  M0ECE
.
.0F24  DIM
.
.       Search for variable
0F2E  S09A6 S0B64 S0C51 S0E2A S1242
0F33  S0F25
0F35  S126C
0FB8  S0DBE S0F3A S0F4D S0F58
.
.0FC2  Create new variable
.
.       Setup array pointer
1028  S10DB S10E9
.
.       Evaluate integer expression
103D  S106B
1043  S161B
1047  S09C8 S0DEA S0E90 S0EA2
.
.       Find or make an array
1059  M0F87
10CD  M1191
10D0  M087B M1612 M18BA
1173  M10E6
11D3  S1126 S119F
11DC  S11C2
.
.1204  FRE then:
.
.       Convert fixed to floating point
1218  M0DF6 M0E44 M0EB6
.
.1225  POS
.
1227  M15F9 M160F M1690
.
.       Check not direct
122B  S0B3D S1238
.
.1235  DEF
.
.       Check FNx syntax
1263  S1235 S1276
.
.       Evaluate FNx
1276  M0DFD
```

```
12D3  M1260
.
.12E9  STR$
.
.       Do string vector
12F9  S0A05 S1340 S14D7
1301  S156D S1589
.
.       Scan, setup string
130B  S0A54 S0ACC S0DDC
1311  S0BAD
134A  S14EE M1577 M15A3
.
.       Build descriptor
1372  S1301
.
.       Garbage collection
13A4  S0433 S120B S139A
13A8  M14B4
1439  S13DD
1443  S13C4 S1434
1482  M13F6
.
.       Concatenate
14B7  M0D14
.
.       Store a string
14F4  S0A10 S14DA
1502  S1347
1506  S14E4 S15A0
.
.       Discard old string
151D  S15FC
1520  S0ACF S0ED7 S1208 S2110
1524  S0EE4 S14E1 S14EB S1590
.
.       Clear descriptor stack
1555  S0A1B S1528
.
.1566  CHR$
.
.157A  LEFT$
.
1580  M15AE
.
.15A6  RIGHT$
.
.15B1  MID$
.
.       Pull string data
15D9  S157A S15A6 S15C2
.
.15F6  LEN
.
.       Switch string to number
15FC  S15F6 S1605 S1627
.
.1605  ASC
.
.       Get byte parameter
1615  S0AAA S214B S216E
1618  S086D S094C S0E6A S15BF M166F
161B  S1566
.
.1627  VAL
.
165D  S0BB0 M0DDF
.
.       Get two parameters for POKE or WAIT
1666  S1693 S169C
166C  S16A8
.
.       Convert floating to fixed
1672  S1669 S1688
.
.1688  PEEK
.
.1693  POKE
.
.169C  WAIT
.
.       Add 0.5
16B8  S1D3E S1FCE
.
.       Floating point subtract
16BF  S18D8 S1FC6 S207C
.
.       -
16C2  S1EFE S1FBF
.
.16D1  Floating point Add
.
16D6  S0C7E M16BC S18CA S18E6 S1F52 S1F81 S1FA6 S1FE2
.
.       +
16D9  M16CE M1C9D
16E6  S19EC
1741  M1B50 M1BE1
1746  M1A9A S1F98
1766  M162C M19D7
1768  M1E55
176A  M19C7
17A5  M1789
17A7  M1B23
.
.       Complement accumulator #1
17E8  S1743
17EE  S1BA6
```

Continued

```
1810  S1B1E
.
.     Overflow exit
181F  M19DA M1CAC
.
.     Multiply a byte
1824  M1924
183A  S16D1 S1BB0
1853  S170F S1BC1
.
.1885-18B2  Constants
.
.     Logarithm
18B3  S1E77
.
.     Multiply
18F1  S1D13 S1E7E S1EC5 S1F1D M1F27 S1F41 S1F7A
.
.18F4  *
.
1922  S1908 S190D S1912 S1917
1927  S191C
1986  M18F6
.
.     Unpack memory into accumulator #2
1987  S16BF S16D6 S18F1 S1A0A
.
.     Test and adjust accumulators
19B2  S18F9 S1A19
19B4  S1F10
.
.     Handle overflow and underflow
19CF  S1EDC
.
.     Multiply by 10
19DD  S1C6A S1C80 S1D30
.
.19F4-19F8  10 in floating point binary
.
.     Divide by 10
19F9  S1C61 S1D37
1A02  S1FB2
.
.     Divide by
1A0A  S18D1 M2017 S2069
.
.     /
1A0D  M1A07
.
.1A0F  Divide into
.
1A4A  M1A72
1A8A  M191F M1A82
.
.   Unpack memory into accumulator #1
1A9D  S0795 S0C71 M0E47 S1A04 S1E4C S1F70 S2007
.
.     Pack accumulator #1 into memory
1AC2  S1F2E
1AC5  S1F18 S1FF2
1ACB  M09D7 S0C81
1ACF  S12A4 S1E5C M1F9F
.
.     Move accumulator #2 to #1
1AF7  S0E9F M16DB
1AF9  S1E72
.
.     Move accumulator #1 to #2
1B07  S19DD S19F9 S1C8E S1E45 S1FA9 S1FB5
1B0A  S1ED3
.
.     Round accumulator #1
1B16  S09C5 S0D72 S1A0F S1ACF S1B07
1B1E  S1ECE
.
.     Get accumulator #1 sign
1B26  S07A5 S18B3 S1B34 S1F66
1B2C  M1B93
.
.1B34  SGN
.
1B37  M0F1E S1C92
1B3F  M1222
1B44  S1CE3
.
.1B53  ABS
.
.     Compare accumulator #1 with memory
1B56  S0ECA S1051 S1D1E S1D29 S1E6A
1B58  S0C86
.
.     Convert floating to fixed
1B96  M1056 S167C S1BCD S1D41
.
.     INT
1BC7  S1E63 S1EDF S1FB8
.
.     Convert string to floating
1BEE  S0BB9 M0DBB S1655
1C05  M1C8A
1C35  M1CBF
1C58  M1C43
.
.   Get new ASCII digit
1C8D  S18EA S1C87
.
.1CC2-1CD0  Constants
.
.     Print IN then:
1CD1  S0471
.
.1CD8  Print Basic line #
.
1CDC  S0705
1CE9  S1CD5
.
.     Convert floating to ASCII
1CEC  S0A51 S1CE6
1CEE  S12EE
1E0F  M1D04
.
.1E1C-1E44  Constants
.
.1E45  SQR
.
.1E4F  Do power function
.
.     Negation
1E88  M1C76 S1F01 S1FDB S1FE8 S205B M2082
.
.1E93-1EC0  Constants
.
.     EXP
1EC1  S1E81
.
.     Series evaluation
1F14  S18DF M1FEF S2070
1F2A  S1F08
1F2E  S1F20
.
.1F5E-1F65  RND constants
.
.1F66  RND
.
1F9F  S2000
.
.1FA2  COS
.
.     SIN
1FA9  S1FF9
1FDB  M201B
.
.1FF2  TAN
.
201A  S2010
.
.201E-2055  Constants
.
.2056  ATN
.
.2086-20C3  Constants
.
.20C4  Transfer source file header to BASIC
.
.20C7  Head up, swap Zpage and stack then:
.
.     Reset OS return on error vector to 20D7
20CD  M0850
.
.20D7  Swap if necessary
.
.     Check keyboard for Control C
20DD  S081E
.
20EA  S11B1
.
.     Reset default IO flag
20F1  S0469
.
.     Move the IO flags to BASIC
20FF  S2186 S21AA S21DF
.
.     Reset OS pointers then:
2110  S2145 S2262
.
.     Set source file header values
211C  S223C
.
.     Pass name to OS buffer
213A  S222E S229F
214B  S2191 S21B7 S21F2
2163  S2283 S2299
.
.     List device finder
2186  M06B9
21A1  M0725
.
.21AA  PRINT
.
21B0  M0A4A
21C8  M0A35
21CA  M0AC4
21D4  S21A4 S21F9 S221C
21DF  S0B2C
21E7  S0B40
220E  S0B44
221C  S0C19
2224  M087E
.
.223C  EXIT
.
.2253  DISK
.
226B  M20C4
2271  S2237
```

```
615 .
616 .227F  DISK CLOSE
617 .
618 2291   M22FD
619 .
620 .2295  DISK OPEN
621 .
622 .22C3-22D3 Disk read and save data
623 .
624 .
625 .       Disk load and save
626 22D4   S22FA
627 .
628 .22F7  USR and disk jump
629 .
630 .       OS Input, no echo out
631 2336   S0587
632 .
633 .       OS Output
634 2343   S0591 S0596 S059B S0B07
635 .
636 .       OS Write buffer to disk
637 2477   S228B
638 .
639 .       OS Test keyboard (Control C check)
640 263D   S20DF
641 .
642 .       OS Home disk
643 2663   S22E8
644 .
645 .       OS Move head to binary track no. in $2662
646 26A6   S22EB
647 .
648 .       OS Move head to BCD track no. in acc
649 26BC   S22B9
650 .
651 .       OS Head down
652 2754   S22EE
653 .
654 .       OS Head up
655 2761   S20C7 S228E M22F4
656 .
657 .       OS Read sector to memory
658 295D   S22F1
659 .
660 .       OS Select a drive
661 29C6   S22E5
662 .
663 .       OS Kernel cold start
664 2A51   M2250
665 .
666 .       OS Set error return from OS
667 2A7D   S20D1
668 .
669 .       OS Command processor
670 2A84   S2268
671 .
672 .       OS Head down, Read sector, Head up
673 2B1A   S22BC
674 .
675 .       OS Command LOAD
676 2BA7   S2231
677 .
678 .       OS Swapper
679 2CF7   S20CA M2148 S2234 S2243 S2265 S226B S2286 S2291 S22F7
680 .
681 .       OS Check flag for a swap
682 2D50   S20D7
683 .
684 .       OS Output following text to $00
685 2D73   S2246
686 .
687 .       OS Convert byte binary to ASCII hex and output
688 2D92   S2240
689 .
690 .       OS Search directory
691 2DA6   S22A2
692 .
693 .       Transient GET and PUT loaded here
694 2E7C   M22BF
```

OSI Microsoft Disk Basic
ZPAGE TABLE

```
10 .      OSI Microsoft Disk BASIC
20 .
30 .          ZPAGE TABLE
40 .
50 .
60 .
70 .
80 .
90 . ;Index for Zpage, Jump vectors for BASIC
100 00   05AD 05D9 0609 0627 1357
110 01   135B 170D 173D 1832 1838 1848 184C 184E 1850
120 02   135F 1736 182E 1834 185A 185C 185E
130 03   172F 182A 1830 1866 1868 186A
140 04   1728 1826 182C 1872 1874 1876
150 .
160 .  ;Search Character
170 0A  1320 1BDF 1E70 1EE2
180 0A  090C 0914 0916 0976 0998 0B95 0B9D 0E97 0EB2 130D
190 .
200 .  ;Scan between quotes FLAG
210 0B  130F 1324
220 0B  05B3 0607 060D 0910 0912 0918 091E 0BA2 0E9D 0EA9
230 .
240 .  ;POINTER: Input Buffer, # of subscripts
250 0C  0EAB 0EB0 0EB4 1028 1091 10DE 1108 112F 1175 11AD
260 0C  049C 04F7 050F 05D1 05E9 061A 0E8E 0E95 0E9B 0EA7
270 .
280 .  ;Default DIM FLAG
290 0D  0F33 1059 109E 10D7 1113 116E
300 .
310 .  ;Type: FF=string 00=numeric
320 0E  09B5 0A4D 0B91 0CBF 0D10 0D36 0DB4 0E31 0ED3 0F44
330 0E  0F63 105E 1097 1204 121A 1368 1601
340 .
350 .  ;
360 0F  09B2 0BBC 0E36 0F46 0F71 105B 109A
370 .
380 .  ;FLAG: DATA scan; LIST quote; memory
390 10  05AB 05B9 0600 1372 1396 139F
400 .
410 .  ;Subscript FLAG; FNx FLAG
420 11  06A8 074A 0F6B 0F81 0F8C 1240 126A
430 .
440 .  ;0=Input; $40=GET; $98=READ
450 12  0B0D 0B5E 0B80 0C12
460 .
470 .  ;Comparison evaluation FLAG
480 13  0D97 0F18 1FD5 1FD9 1FF7 200E
490 .
500 .  ;Input FLAG (suppress output)
510 14  044E 0474 0849 0AEE 21E2
520 .
530 .  ;
540 15  0878 0A81
550 .
560 .  ;Position on print line
570 16  0A5C 0A75 0A8D 0A92 0AB8 0AF7 0B00 1225
580 .
590 .  ;Terminal width
600 17  0A5E 0AF9
610 .
620 .  ;Input column limit (0841 BIT operand)
630 18  0841 0A94
640 .
650 .  ;Integer value (for GOTO, etc.)
660 19  064E 06DA 06E2 06FD 096E 0980 0988 098A 0992 0996
670 19  099A 1683 168D 1699 16AF
680 1A  0643 06DC 06E4 06F9 08AE 0970 0978 098E 0990 0994
690 1A  099E 1685 21E9
700 .
710 .  ;Start of Input Buffer
720 .  ;71 Characters; From $1B to $62
730 1B  04E7 0579 0A6F 0B47 2211
740 .
750 .  ;BIT operands
760 38  0CBD
770 48  0CD7 1852
780 .
790 .  ;POINTERs for Descriptor stack
800 63  0697 134A 136F 13C0 155D
810 64  136A 1559 1561
820 65  1555
830 .
840 .  ;$66 - $6E  Descriptor Stack
850 .
860 .  ;Utility POINTER area
870 6F  03DB 03EA 03F5 0417 041A 04AB 04D2 04D9 0521 0528
880 6F  1443 1448 144C 1466 1475 1477 1502 150B 1524 152E
890 6F  1532 1536 1550 1597 1599 160C 1637 163C 1987 198D
900 6F  0532 0538 053D 0544 0546 078A 097A 0983 0986 098C
910 6F  1992 1997 199C 19AB 1A9D 1AA3 1AA8 1AAD 1AB2 1ABB
920 6F  1AD2 1ADA 1ADF 1AE4 1AED 1AF2
930 6F  13F9 13FF 1403 1407 140E 141B 1422 1424 1439 143E
940 6F  0AD9 0D19 0D1C 0D69 0D6B 112D 11A7 11D3 13BC 13D1
950 70  04A7 04D6 04E0 0523 053F 0548 078C 0D6E 13BE 13D3
960 70  1640 1989 1A9F 1AD4
970 70  13FB 1428 142A 1468 147B 147D 1504 1526 1552 159D
980 71  1B6C 1B75 1B7C 1B87
990 71  04BB 04DB 0C6D 163E 164B 1650 165B 1B56 1B5C 1B62
1000 72  04AF 04CF 04E2 1647 1B58
1010 .
1020 .  ;
1030 73  18FE 1940 1944 194C 194E 1950 1A8A
1040 .
1050 .  ;
1060 74  1900 193A 193E 1958 195A 195C 1A8E
1070 .
1080 .  ;Product area for multiplication
1090 75  11BE 11D7 11F6 1902 1934 1938 1964 1966 1968 1A92
1100 76  11DC 11FA 1904 192E 1932 1970 1972 1974 1A3F 1A96
1110 77  2124 2276
1120 .
1130 .  ;POINTER: Start of BASIC
1140 78  051D 0633 0667 066A 066C 06AC 080B 08BD
1150 79  051F 0635 0672 06B2 080F 08BF
1160 .
1170 .  ;POINTER: Start of variables
1180 7A  04A9 04B7 04B9 04C9 04F3 050B 0670 0686 09F6 0F8E
1190 7A  13CD
1200 7B  04BD 04C1 04FB 050D 0676 0688 09EE 0F90 13CF 212C
1210 .
1220 .  ;POINTER: Start of arrays
1230 7C  068A 0F9C 0FD7 0FF9 10A0 13D9
1240 7D  068C 0F98 0FD9 0FFB 10A2 13D5
1250 .
1260 .  ;POINTER: End of arrays
1270 7E  03D2 0507 068E 0FDF 10AC 1145 115F 1211 1385 13B0
1280 7E  13F2
```

Continued

```
7F   03D4 0509 0690 0FE1 10A8 1147 1167 1216 137F 13B2
7F   13EE
.
.    ;POINTER: String storage (moving down)
80   149A 1541 1547 1549
80   0425 0447 04EF 0682 09E8 120F 1378 1389 13A8 1454
81   041F 0441 04F1 0684 09DF 1214 137A 138B 13AA 144E
81   149C 153D 154D
.
.    ;POINTER: String utility
82   138D 150D 1514 1516
83   138F 151A
.
.    ;POINTER: End of memory
84   04EB 067E 13A4
85   04ED 0680 13A6
.
.    ;Current BASIC line number
86   0770 07D0 0837 0868 0897 08EE 0B15 0C93 1CDA
87   0C98 122B 1CD8
87   046C 048C 076D 07D5 0839 086A 0894 08AC 08F1 0B17
.
.    ;Previous BASIC line number
88   083B 0864
89   083D 0866
.
.    ;POINTER: BASIC statement (for CONT)
8A   07BD 0833 085E 0B23
8B   06A6 07BF 0835 0857 0B25
.
.    ;Current data line number
8C   0B11 0BF9
8D   0B13 0BFF
.
.    ;Current Data address
8E   0814 0B58
8F   0816 0B5A
.
.    ;Input vector
90   0B60 0B73 0BD1 0C0E 2221
91   0B62 0B75 0BD3 0C10
.
.    ;Current variable name
92   0F35 0F73 0F75 0FA0 0FFF 1068 106F 10B5 10F6 20EE
93   0F7E 0FA6 1004 1065 1072 10B9 10FE 11B8
.
.    ;Current variable address
94   1023 11C8 11D0 1254 1293 129C 12B9 1DAC 1DC3
95   1025 11CD 1251 1299 12A2 12B6
.
.    ;Variable POINTER for FOR, NEXT
96   0A22 0A27 0A2C 0B67 0C54 0C7A 169F 16B3 1ACB
96   03B4 03C0 0703 070A 072E 07AE 08D7 09A9 09CF 09D4
97   03AD 03B9 07AB 09AB 0B69 0C56 0C7C 16AB 16B1 1ACD
.
.    ;Start of work area (POINTERs, etc.)
98   0B6F 0BD5 0D2D 0D93
99   0B71 0BD7
9A   0CE5 0CF8 0CFA 0CFE 0D06 0D44 0D5A 0ED5
9B   12DE 12E2 12E6 146A 148E 14AA 14B2
9B   126F 127C 1286 128F 1297 12AD 12B2 12C0 12D6 12DA
9C   1271 1279 1289 12C3 13AE 146C 1482
9D   15AA 15CB 15E6
9D   0A08 0A17 0A20 0A25 0A2A 12FD 14DD 157D 1582 158C
9E   0A0A 0A19 12FF 14DF 158E 15E9
.
.    ;
A0   13CB 13E8 146E 1472
.
.    ;$A1 Jump vector for functions
.
.    ;Jump vector value
A2   0E7C 1470 1486 148C 14A5 15DF 15EB
A3   0E81 16E0 16FC 1721 176D 1ED1 1EFA
A4   042E
.
.    ;Misc. numeric work area
A5   11C6 13E2 13EA 1409 140B 1430 149E 14A8
A5   03F3 03F7 0401 0408 04F9 0FED 0FF4 1034 113B 1152
A6   03FB 040C 0502 0FEF 0FF6 1036 1133 1137 1156 115C
A6   11CB 13E4 13EC 1410 1412 142C 14A0 14AD 14AF
A7   03D7 03E7 03EC 03FF 0406 04F5 0FE3 1492
A8   03DE 03F0 040A 04FD 0FE5 1498
AA   11E0 11FF 1BF2 1C59 1C7E 1D18 1D33 1D3A 1D46 1D5B
AA   1DB7
AB   1C41 1C56 1C5B 1C64 1C6D 1CA0 1CB2 1CBD 1D59 1DE0
AB   1DE9
AC   03D9 04A5 04B1 04B4 04C7 0515 0639 063D 0645 0651
AC   1185 118D 11D5 11DA 13B4 145E 1462 1490 1C4C 1C4E
AC   1014 1017 1019 102D 10A4 10B2 10BB 10C0 10C3 10C7
AC   0F96 0FA2 0FA9 0FAF 0FDB 1001 1006 100B 100E 1011
AC   10E2 10F8 1100 110D 1120 1124 1161 1165 116C 1173
AC   1C50 1C52 1C7A
AC   0658 065C 06E8 06F3 06F7 0714 0719 071D 071F 08C6
AD   1C31 1C33 1C3A 1CA8
AD   10A6 10C9 116A 13B6 1458 1464 1494 1AFF 1B0C 1C2F
AD   03E0 04AD 04C3 063B 0721 08CC 0F94 0FDD 101E 102F
.
.    ;Accumulator #1: exponent
AE   18C4 19AF 19B7 19C3 19EF 1A15 1A17 1A1C 1ABD 1AF0
AE   1B16 1B26 1B4A 1B68 1B96 1BC7 1BDB 1C9B 1D00 1E88
AE   0439 0938 0D81 0DAF 0EDA 0EED 0EF5 1047 1308 132D
AE   1ED6 1EF1 1EF3 1F90 1F96 205E
AE   1355 1676 16CC 16EA 16F0 1768 179B 17A3 17A7 18BD
.
.    ;Accumulator #1: mantissa
AF   0782 0784 0D7E 0EDC 0F0B 121C 1304 1315 1359 1573
AF   173F 174A 1750 1783 1787 1796 17B1 17B3 17B5 17EE
AF   1B70 1BBB 1BBD 1BBF 1BE4 1CDC 1D8F 1D94 1F86 1F8A
AF   17F2 181C 191A 1A26 1A6D 1A8C 1AB8 1AEB 1B37 1B3F
B0   1781 1794 17BD 17BF 17C1 17F4 17F8 1818 1915 1A2C
B0   0D7B 0EDE 121E 1306 1317 135D 1738 174E 1754 177D
B0   1A67 1A90 1AAF 1AE2 1B3B 1B77 1BE6 1CDE 1D88 1D8D
B1   14BA 14CE 1520 161E 167F 1731 1752 1758 1777 177B
B1   1792 17C9 17CB 17CD 17FA 17FE 1814 1910 1A32 1A61
B1   09CD 09DD 09E6 09F4 09FA 0A03 0A59 0D78 0DF2 0E2D
B1   1A94 1AAA 1ADD 1B48 1B7E 1BE8 1D81 1D86
B1   0E3C 0E40 0E65 0E93 0EAE 107F 1180 11A3 12F9 1363
B2   1B46 1B89 1BDD 1BEA 1D79 1D7F 1F84 1F88
B2   0958 09D2 09EC 09FC 0D75 0DED 0E2F 0E62 0E99 0EA5
B2   1622 1681 172A 1756 175C 1771 1775 1790 17D5 17D7
B2   17D9 1800 1804 1810 190B 1A38 1A5B 1A98 1AA5 1AD8
B2   1084 1183 11A9 12FB 1365 14B7 1522 15B3 15D1 15D5
.
.    ;Accumulator #1: sign
B3   16F4 176A 17E8 17EC 19A0 19CC 19CF 1AB4 1AE7 1AF9
B3   1CF0 1CF9 1E8C 1E90 1F8E 1FC9 1FD1 200C 2056
B3   1B2A 1B4E 1B53 1B64 1B8D 1B9D 1BB7 1BD2 1BD4 1C97
B3   077E 0C78 0D62 0DAB 0EF9 0F02 1043 1672 16C2 16C6
.
.    ;POINTER: constants for series evaluation
B4   1BFD 1C71 1F33 1F59
.
.    ;Accumulator #1: high order (overflow)
B5   1836 1AFD 1B0E 1BA4 1BB3 1BC4
.
.    ;Accumulator #2: exponent, etc.
B6   0D9A 16E4 19AD 19B2 1E51 1EEF 1EF5
B7   0D9D 0EC2 0EC4 1785 1942 19A8 1A24 1A50 1A6B 1A6F
B8   0DA0 177F 193C 1999 1A2A 1A4E 1A65 1A69
B9   0DA3 0EE0 0EE7 0F09 1779 1936 1994 1A30 1A4C 1A5F
B9   1A63
BA   0DA6 0EE2 0EE9 1773 1930 198F 1A36 1A4A 1A59 1A5D
BB   0DA9 0EBE 16C8 16F2 199E 19A4 1AF7 1C95 1E5F 1FB0
.
.    ;Accumulator #1 vs #2: sign comparison
BC   14FA 14FE 16CA 1712 19A2 19CA 19EA 1A02 1C99 1F0D
BC   0A0C 0DAD 1311 131C 1330 1343 14C4 14CB 14E7 14F6
BC   1FBD
.
.    ;Accumulator #1: low order (rounding)
BD   1B04 1B13 1B1A 1B4C 1B85 1BD0 1EC8 1EFC 1F92
BD   180C 1828 1843 1906 197C 197E 1980 1A7F 1ABF 1AF4
BD   1723 175A 175E 176F 178E 17E1 17E3 17E5 1806 180A
BD   0A0E 1313 1334 133B 1345 14C7 14E9 16DE 1704 170B
.
.    ;POINTER: series
BE   1F31 1F35 1F3D 1F44 1F4E
BE   1633 165D 1CFB 1D62 1D73 1DAE 1DC1 1DCF 1F14 1F2A
BE   05CD 05EB 1106 1129 114D 1179 119A 11AB 11EE 1332
BF   1F16 1F2C 1F3B 1F3F 1F46 1F50 20EA
BF   10F2 112B 114B 1158 117B 1198 11F0 1339 1635 165F
.
.    ;$C0 CHRGET subroutine: get BASIC character
.
.    ;$C6 subentry to $C0: get previous character
.
.    ;POINTER: BASIC within a subroutine
C7   091A 0B27 0B6B 0B77 0B8A 0BA4 0BCD 0BD9 0BF2 0BF7
C7   07CE 07D3 082D 0860 0891 08B4 08CA 08F4 08FE 0900
C7   0480 05A7 05D4 0618 0630 06B0 0765 07B7 07C3 07C9
C7   12AA 12AF 12CE 162F 1639 1661 1CB8
C7   0BFC 0C01 0C9D 0CCD 0CD3 0D3A 0D40 0DD3 0E17 125A
C8   0482 06B6 0768 07B9 07DE 082F 0862 088E 08B6 08D0
C8   08F7 0904 0B29 0B6D 0B79 0B8C 0BA6 0BCF 0BDB 0CA2
C8   0CD1 0D3E 0DD5 1257 12A7 12B4 12D1 1631 1642 1663
.
.    ;POINTER: GET, PUT to $2E79
FE   22B5
FF   22B1
```


```
BF   10F2 112B 114B 1158 117B 1198 11F0 1339 1635 165F
.
.    ;$C0 CHRGET subroutine: get BASIC character
.
.    ;$C6 subentry to $C0: get previous character
.
.    ;POINTER: BASIC within a subroutine
C7   091A 0B27 0B6B 0B77 0B8A 0BA4 0BCD 0BD9 0BF2 0BF7
C7   07CE 07D3 082D 0860 0891 08B4 08CA 08F4 08FE 0900
C7   0480 05A7 05D4 0618 0630 06B0 0765 07B7 07C3 07C9
C7   12AA 12AF 12CE 162F 1639 1661 1CB8
C7   0BFC 0C01 0C9D 0CCD 0CD3 0D3A 0D40 0DD3 0E17 125A
C8   0482 06B6 0768 07B9 07DE 082F 0862 088E 08B6 08D0
C8   08F7 0904 0B29 0B6D 0B79 0B8C 0BA6 0BCF 0BDB 0CA2
C8   0CD1 0D3E 0DD5 1257 12A7 12B4 12D1 1631 1642 1663
.
.    ;POINTER: GET, PUT to $2E79
FE   22B5
FF   22B1
```


OSI Microsoft Disk Basic
MEMORY TABLE

```
.        OSI Microsoft Disk BASIC
.
.             MEMORY TABLE
.
.
.
.
.
.        Zpage
0001     L173A
```

Continued

DBi, inc.

p.o. box 21146 • denver, co 80221
phone (303) 428-0222

SPECIAL PURCHASE on hard disk drives
SPECIAL PRICES on **DBI BUSINESS SYSTEMS**
RUNS DB—DOS & OS—65U PROGRAMS*

DBI 420SE

(4) DB-1 MULTI-PROCESSING BOARDS
★ TRUE PARALLEL/MULTI-TASKING
★ ALL USERS RUN AT 2 MEGAHERTZ
(1) DS-1 SCSI HOST ADAPTER
★ W/BATTERY BACKED-UP REAL TIME CLOCK
(1) DP-1 UNIVERSAL PRINTER BOARD
★ 4 RS-232 SERIAL INTERFACES
★ 2 CENTRONICS COMPATIBLE INTERFACES
(1) FAST 20 MEGABYTE HARD DISK
(1) 318K BYTE FLOPPY DISK
(1) INTELLIGENT SCSI CONTROLLER
★ W/ERROR CHECKING AND CORRECTION

LIST $10,490
ONLY! $6,695
SAVE! $3,795

DBI 220SE

Same as 420SE except Two Users

LIST $7,900
ONLY! $6,395
SAVE! $1,505

DBI 440SE

Same as 420SE With 40 Megabyte Hard Disk

LIST $13,100
ONLY! $8,895
SAVE! $4,205

DBI 240SE

Same as 440SE except Two Users

LIST $10,510
ONLY! $8,595
SAVE! $1,915

* OS-65u IS A TRADEMARK OF OHIO SCIENTIFIC, INC.

QUANTITIES ARE LIMITED **PLEASE DON'T DELAY!**

```
110 0002  L1733
120 0003  L172C
130 0004  L1725
140 0016  S05EF L05F2 S0612
150 0017  L0512
160 0018  S062B
170 00A0  B0E8B
180 00A2  B0909
190 00FF  S1CF6 S1D67 S1D70 S1DB4 S1DBE L1DD1 S1E0F
200 .
210 .     Stack
220 0100  S1DF3 S1E14
230 0101  L03A6 L107B S1086 S1DEE
240 0102  L03B1 C03C2 L0FC9 L1077 S1081 S1E05
250 0103  L03B6 C03BB S1E01
260 0104  S1E0A
270 0109  L0C75 S0C8B
280 010F  L0C90
290 0110  L0C95
300 0111  L0C9F
310 0112  L0C9A
320 01DE  L0E79
330 01DF  L0E7E
340 .
350 .     Start of keyword address table
360 0200  L07F9
370 0201  L07F5
380 .
390 .     Start of operator hierarchy and address table
400 0266  C0D20 C0D48 L0D64
410 0267  L0D53
420 0268  L0D4F
430 .
440 .     Table of BASIC keywords (Start $0284)
450 0283  L061D
460 0284  S05E0 L0622 L0736 L073E
470 .
480 .     Error messages
490 0364  L0456
500 0365  L045C
510 .
520 .     BIT hiding code
530 07A9  B057C
540 08A2  B10CF
550 0EA2  B08E3
560 1410  B19BE
570 .
580 .     Constants
590 1E21  A1D91
600 1E22  A1D8A
610 1E23  A1D83
620 1E24  A1D7C
630 .
640 .     Operand pointing to IO flags
650 21D5  S2104
660 21DA  S210A
670 .
680 .     Stack pointer
690 226F  S211F
700 .
710 .     Table index for OS buffer write routine
720 228A  S217F
730 .
740 .     Buffer read write data for OS
750 22C8  L22E2
760 22C9  L22D6
770 22CA  L22DC
780 .
790 .     USR pointer to OS and disk
800 22F2  S22D9
810 22F3  S22DF
820 .
830 .     OS Input flag
840 2321  S20F5 L2101 S21D6 S2201 L2215
850 .
860 .     OS Output flag
870 2322  S20F8 L2107 S215D S21DB L21FE S2208
880 .
890 .     OS Passed char. (Control C)
900 2325  L0819 S0823
910 .
920 .     OS Disk sector number
930 265E  S22AC
940 .
950 .     BIT hiding code
960 28A9  B0E0F
970 .
980 .     OS Default IO flag
990 2AC6  L20F2
1000 .
1010 .     BIT hiding code
1020 2CA9  B0E12
1030 .
1040 .     OS Read buffer pointer
1050 2CE5  S2142
1060 .
1070 .     OS End of buffer on read
1080 2CED  S2113
1090 .
1100 .     Transient GET and PUT pointer
1110 2E7A  L22A6
1120 .
1130 .     OS Swapped value ($E1,$E2) Start pointer for buffer read
1140 305A  S2116
1150 305B  S2119
1160 .
1170 .     Pointer to SOURCE File header
1180 3178  S2126 L2273
1190 .
1200 .     Number of tracks in SOURCE File
1210 317D  S2136
1220 .
1230 .     BIT hiding code
1240 3FA9  B0AEB
1250 A4A2  B1AC4
```

★

BEGINNER'S CORNER

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

PROBLEM SOLVING
PART 2

EUREKA!

Last month's article discussed problem solving, with particular emphasis on forming a plan from which a program could be coded. The program derived from that plan is listed here as Version 1. Notice the DATA statements are where they can be easily found, at the end of the program.

When solving problems, it is important to avoid the "Eureka" syndrome. More often than not the solution is incomplete or a better one exists. A search can be made for a "better" solution. This solution could be either faster, or shorter or be just plain elegant!

A BETTER SOLUTION

A good question to begin with is: is the solution to another, similar problem already known? The answer, of course, is yes; it's version one of the program. In that program the output block uses three FOR...NEXT loops to produce the required printout. It would be more efficient to use only one loop. Imagine a BASIC program which had to use the idea four times to produce four different types of reports. There would be 12 FOR...NEXT loops. Reducing the number of loops to four would be a great advantage. The program would be easier to write, shorter and faster.

Before the problem can be solved some more information is required, another TOOLBOX.

TOOLBOX

1. Increased program efficiency can be had at the expense of using more computer memory.

2. There are 12 month names.

3. a) A list of month names which goes through December is in fact two lists.

   b) The first part of the list always finishes with month 12. The second part of the list always begins with month 1.

POINTERS

This is the tricky part. If months beyond December are required to be printed, why not continue counting beyond 12 - to 13, 14, 15 and so on to 24. All that is then required is to associate January with 13, February with 14 etc. What is wanted is some way of POINTING from numbers above 12, to the month names, e.g. 13 --> January.

This is easily done after a reminder from point 1 in the TOOLBOX. Form an array, of length 24, which contains the numbers 1 to 12 - twice! The first element in the array will be a "1", and so will the thirteenth!

N(1)=1, N(2)=2, N(3)=3, ..., N(12)=12, N(13)=1, N(14)=2, ..., N(24)=12.

Have a look at the structure diagram to see how the new OUTPUT block works.

VERSION 2

Make the following changes to Version 1.

```
10 REM Ver. 2
25 DIM N(M*2)
40 FOR C=1 TO M: N(C)=C): NEXT
50 X=0: FOR K=C TO M*2: X=X+1:
   N(K)=X: NEXT
130 IF F>L THEN L=L+M
140:
150 FOR C=F TO L
160 :   M=N(C): PRINT M$(M)
170 NEXT
```

Sometimes when a program is "finished" the real work begins. The following questions should be asked. Do the program and output make sense? Does the program work as intended? Finally, does the program survive severe testing?

SUMMARY

Writing programs teaches at least two very important problem-solving skills: procedural thinking and breaking a problem up into smaller, more easily-solvable units. The following summary is offered as a guide to writing better programs.

1. Identify and understand the problem.

2. Assemble the information required to solve the problem - is it all there, is any of it redundant?

3. What type of problem is it - can it be solved on a computer?

4. What is the connection between the problem and the information?

5. Can the problem be broken up into smaller units?

6. Are solutions known to similar problems?

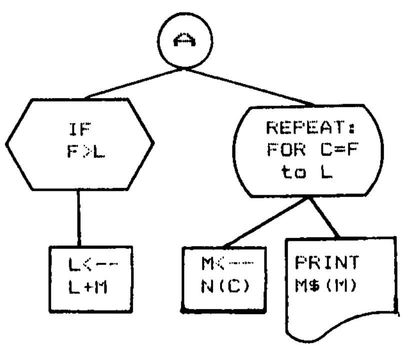
7. Is all the information being used?

8. Can the plan be coded?

9. Do the program and its output make sense?

10. Does the program work as intended?

11. Does the program survive severe testing? Do it, test it!




```
10 REM Months Problem ver.1 - by LZJ
20 PRINT!(28): M=12: DIM M$(M)
30 FOR C=1 TO M: READ M$(C): NEXT
40 :
50 :
60 :
70 REM -------------Input numbers-------------
80 INPUT "From Month, number ";F: PRINT
90 INPUT "To Month,   number ";L: PRINT
100 IF F<1 OR L<1 THEN RUN
110 :
120 REM ----- -Output List of Months---------
130 IF F>L THEN 160
140 FOR C=F TO L: PRINT M$(C): NEXT: END
150 :
160 FOR C=F TO 12: PRINT M$(C): NEXT
170 FOR C=1 TO L:  PRINT M$(C): NEXT
180 END
190 :
200 DATA Jan,Feb,Mar,Apr,May,Jun
210 DATA Jul,Aug,Sep,Oct,Nov,Dec
```

★

READER SURVEY RESULTS

A run-down on what you, our readers had to say about PEEK(65) as based upon the Reader Survey in the November 1984 issue.

You probably thought that we had forgotten about the Reader Survey, didn't you? Would you believe that we received a return today. That, in a nut shell, is why the results have been delayed this long.

First of all, we are very grateful to all of you who took the time to fill in all of those blank lines and empty squares. It is only by your responses that we will be able to better serve your interests and needs in the future issues of PEEK(65). Certainly, we will do everything in our power to see to it that you get what you want.

To start with, we must be doing something right, because a good 65% of you said that we were just right as far as technical material is concerned. Of the remaining, 10% found PEEK(65) too technical and 25% not technical enough.

We get the message on the number of reviews. A full 74% wanted to see more reviews. Two thirds wanted software reviews, and a third hardware reviews.

Another message that rings loud and clear is that almost 80% of you enjoy non-technical articles on OSI applications. Good news! Several such articles are currently in the works and we hope that those of you reading this will take the "tip" and write to us about your particular application.

Now it is my turn to get even! Did you realize that a mere 30% of you support us with your articles? Now is the time to feel guilty and write something before we come knocking at your door. Judging from the response to question 5, what will you write about, we can expect from every ten submissions: 4 hardware articles, 3 on utilities, and one each on Pascal and other software. Strangely, a quarter of you know someone who would be willing to write, but most of you failed to let us know who to contact or what subject they might write about. There is always room in the mailbox, so fill'er up, please.

Equally important to us was your list of things that you would like to see articles written about. You will find it at the end of this report. Please do read it over closely. As long as it is, you surely can find something there that you have had experience with, and you will know that you will have anxious readers waiting to receive your help and guidance.

As you know, all too well, our advertising has been restricted very closely to direct OSI type products and services, but since 60% of you would like to see advertising on other materials, peripherals and supplies, we are going out to beat the bushes to try to find reputable vendors of products that will be of interest to you. If you have any particular desires, please let us know.

Your report on satisfaction in dealing with our current advertisers is both rewarding to us and, being optimists, a bit discouraging if there is even one disappointment. We continually strive for that magical 100%, but realize that little things do crop up and that there are frequently two sides to most stories. As it stands, 94% were satisfied, but we would like to hear from the other 6% so that we can try to assist you in getting satisfaction.

Now here is a real disappointment! Only about 12% of you took advantage of our free listings of software in the October and November issues. If only you knew how many calls we get from people looking for software and the number of requests for those back issues, maybe more of you would avail yourselves of this service. I think that the problem is that you have seen some pretty elaborate pieces of software listed and feel that yours may not be in that league or just not up to "your" standards. Well, look again, a little closer. You will find software of every caliber and price. The point is that you may be sitting on a piece of software that is just what many others are craving for, but haven't the time or knowledge to put together themselves. So, start preparing yourself for the next software issue now!

That brings us to buying software directly from PEEK. Now that we know that 81% of you would like this service, we are going to have get out of neutral, stop talking about it and get something done. In fact, we have been talking about it for some time, hence the question in the first place. Unfortunately, the mechanics are not as simple as one would hope, but we hope to have the details worked out so that we can announce the service shortly.

We never have figured out why OSI is so secretive about their dealer list. Certainly we have tried and tried, even to get an unpublishable list, but the answer is always, No! This, to us seems regrettable, especially when 62% of you don't know of a dealer. OSI says that you have only to call the Fairfield office at (203) 255-7443 and they will tell you. In the meantime, we will certainly let OSI know about those of you out there in the dark.

Just as a footnote, it is not surprising that 67% of you wind up doing your own repairs. For those of you who are not that handy, we hope that you have noted the ads for the very reputable service companies that appear within our pages.

When it comes to the hardware you are using, about a third are using serial systems, when serial conversions to the "P" series are included. Among the "P" machines, the popularity list is headed by the 4P-MF. Remarkably close behind is the C1P and C8P-DF. The C4P cassette trails at a comfortable distance and is only underdone by the 300 series CP/M machines.

Our last question about 5" versions of OS-U was frankly a shot in the dark. Those of you who use your machines for business work or are heavy users of disks and have had experience with OS-U will vouch for its advantages. The uninitiated, for the most part, say, "U what?". It is also clear that we gave you too many choices. Three would have been enough. Trying to regroup the answers says that half have no interest, a quarter are mildly interested and the remaining quarter (presumed to be those who have seen U) were eager. All this means that we have another job on our hands. We know that U has been independently converted to 5", but those we have heard of were on the earlier versions of U or are not in OSI format. It is the Op. Sys. that causes the problems. Remember that disk sector lengths and structure are quite different on the two sizes of disks. Nonetheless, we will persevere.

So, there you have it. Now you know all about you! Seriously, we appreciate your effort in responding to the survey and you have our promise to make good use of this information to improve PEEK by giving you what you want and need. The following is a list of the articles you requested.

HARDWARE

1. Interfacing the outside world
2. Modems and useage
3. 68000 add-on & cross assembler
4. 6502/65816 replacement, 48K plus.
5. Floppy Disks, comparisons, & which brands can be added, increased capacity, e.g. Western Digital Controller
6. New Hardware
7. Hams, interfacing
8. Memory tests
9. Triple processor application, how to
10. EPROM programmers
11. Mods & upgrades, general
12. DBI Boards, installing, description
13. BSR-X hook-up, software, use
14. Mods for Hi-Res and G.T., how to make
15. Floppy drives - how to repair, minor
16. Speech synthesis & music
17. Hard disk installation
18. Interfacing a plotter to 550
19. Floppy disk shut down control 5 & 8"
20. Trouble shooting, simple

SOFTWARE

1. Data Base Management explained OS-U & OS-D
2. Sorts OS-U & OS-D
3. Terminal program for C1P
4. Telecommunications, general
5. Intergrated software
6. WP-3 inner workings
7. Engineering software
8. Algorithms
9. Apple to OSI program conversion
10. Zenerex software reviews
11. Games for learning programming
12. Screen dumps
13. 3.3 hints, techniques, utilities
14. PEEK & POKE lists OS-U & OS-D
15. Text editors
16. Proportional spacing & right justifying in BASIC
17. Data acquisition & process control
18. Programming for speed OS-U & OS-D
19. Utilities explained OS-U & OS-D
20. How to begin in BASIC
21. RTMON uses, how to
22. How OS-U works
23. Assembly; more classes, regenerated code, utilities, how to use, access, (please Rick)
24. DMS mods to make use of 1.4+
25. How to use internal clocks
26. Business/Commercial package reviews

***OPERATING SYSTEMS**

1. DOS/65 tools and applications
2. OS-U disassembly
3. 3.3 disassembly
4. Overview/comparison of OS-D, OS-U Hexdos, DOS/65, etc.
5. PASCAL, how to, pluses & minuses
6. Assemblers, how to use OS-D & OS-U

7. Compilers, how to use OS-D & OS-U
8. TurboDos explained
9. Hard disk interfaces
10. Disk controller blocks explained OS-U
11. Control registers & I/O interrupt routines explained
12. FORTH installation, FIG FORTH programs
13. HEXDOS, everything for 8-16K C1P (with disk)

MISCELLANEOUS

1. New applications, profiles
2. Profiles in general
3. System upgrades & variations
4. Side-Kick for OSI
5. From the dealers point of view

PEEK(65) Staff.

★

"PADDING" for DMS Files in OS65-U

By: Editor

Sometimes we forget that what is an old problem to some, is a new problem to others. One that has come up several times in recent weeks has to do with the "padding" of string data to be put in a DMS file.

Like it or not, DMS requires that data be placed at the right end of the designated field and thus the remaining space at the left must be filled or "padded" with spaces to make the total length of the string equal to the length of the field in which it is to be placed.

In days gone by, in the earlier versions of OS65-U, programmers went to great lengths to write code that would add the extra spaces. This involved creating a space string (SP$="   ") and then concatenating it to the front end of the string to be saved. Next, the surplus strings were chopped off by the use of the MID$ for the length of the field. Not only does all this wear the fingers, but it generates garbage like mad and further slows down the machine.

Enter Extended Input. Most of the features of EI are well enough known to programmers, but there is a little note that is often overlooked. It deals specifically with the problem at hand. It looks like: PRINT%1,[FL,"R"]QA$ and is read: Print to channel 1, for the length of the field (FL) and Right Justified, the string QA$. Voila, the data is at the right and padded with spaces - and no garbage generated - well all but one concatenation.

The programmer doesn't get away scott free, because he does have to collect the field lengths from the header of the file when it is opened. But then, this is a good habit to get into anyway. What one usually ends up with is a fairly standard file opening routine that picks up most everything in the header.

For those who might see other opportunities in this routine, the pad character is normally a space (ASCII 32), but, if your heart desires, it can be changed by POKEing the new value into 12098. Thus, if you need "*"'s, POKE 12908, ASC(*). Best change it back to a "32" when finished, though. How about POKEing it to a "." to get a dotted tab for your financial reports?

★

OSI MINI-LOGO

By: M. F. Putnam
2234 Nancy Place
Roseville, MN 55113

Logo is a programming language which is often used in schools to teach programming concepts to young children. It is also known as the turtle graphics language because originally it was used to control a small domed mechanism which resembled a turtle. This turtle had a pen which could be used to draw lines as the turtle was moved about.

Currently, versions of Logo are available for all of the home computers except, of course, OSI. I have, therefore, developed a program called OSI Mini-Logo which supports several of the turtle commands found in Logo. The program is written in BASIC for a C4P-MF but should run on a C1P-MF with changes as noted in the program listing. The purpose of Mini-Logo is to let one become familiar with simple turtle commands and get somewhat of a taste of what Logo is all about. This Mini-Logo is a far step from normal Logo in its capabilities and power. Normal Logo lets you write small programs called procedures which can be given a name and become a part of the computer's vocabulary. Once defined, these procedures can be executed by other procedures and the final Logo program becomes a collection of these procedures. Mini-Logo allows just 1 procedure to be written and executed at a time. This is very similar to how you can just write and run 1 BASIC program at a time.

To use Mini-Logo you do not have to run out and buy a mechanical turtle. Instead, Mini-Logo uses the OSI graphics characters which look like arrows pointing in various directions (character code 16 thru 23). This represents the turtle on the video screen. The turtle (i.e. arrow) carries an invisible pen which can be used to put OSI graphic characters on the screen as the turtle is moved about. The turtle commands supported by Mini-Logo are shown and described at the end of the listing for the Mini-Logo program.

After typing the Mini-Logo program into your computer and giving it a run, you will see a <? prompt. This means Mini-Logo is ready to accept input from the user. At this time

you can type in a turtle command and have it executed immediately, or you can type in a program by entering line numbers and turtle commands. An example of using direct commands to have the turtle draw a triangle is shown below.

```
FREEZE          turn off screen scroll
CLEARSCREEN     clear screen and show turtle
SETPEN 161      set turtle's pen to graphic char 161
PENDOWN         lower turtle's pen
FORWARD 10      move turtle forward 10 positions
LEFT 135        rotate left 135 degrees
FORWARD 10      move turtle forward 10 positions
LEFT 90         rotate left 90 degrees
FORWARD 11      move turtle forward 11 positions
```

Below is a sample Mini-Logo program which instructs the turtle to draw a square.

```
                 ------ Draw a square ------
 ZAP             clear workspace, same as BASIC NEW
 10 CLEARSCREEN  clears video screen
 20 SETPEN 161   set turtle's pen to graphic char 161
 40 PENDOWN      lower turtle's pen
 50 FORWARD 10   move forward 10 positions
 60 RIGHT 90     rotate right 90 degrees
 70 FORWARD 5    move forward 5 positions
 80 RIGHT 90     rotate right 90 degrees
 90 FORWARD 10   move forward 10 positions
100 RIGHT 90     rotate right 90 degrees
110 FORWARD 5    move forward 5 positions
120 END          end of program
 PO              list program to check if correct
 RUN             run it!!
```

This logo program may be edited by adding, deleting, or re-entering lines the same way you do for BASIC. Commands may be entered in short form using the first couple of letters as indicated by the capital letters shown in the command listing at the end of the Mini-Logo program. For example, instead of entering CLEARSCREEN you can enter CL or instead of FORWARD 10 the short form FO 10 could be entered. If you forget the Logo commands, then type HELP and the command descriptions will be displayed.

A Mini-Logo program can be saved to disk by entering SAVE N where N is a number 1 to 10 which is assigned as the program number. The program will be saved at track N+29. For example, if SAVE 1 was entered the program would be saved at track 30. Make sure the track is initialized before doing the save. To load the program back into the logo workspace, type LOAD 1.

I hope you will find Mini-Logo both educational and entertaining. Do not forget to let the children give Logo a try, they may surprise you if they had it in school.

```
10 PRINT:PRINT:PRINT
12 PRINT"          ******************************
14 PRINT"          *       OSI Mini-LOGO        *
15 REM  "          *     C4P-MF or C1P-MF       *
16 PRINT"          ******************************
18 PRINT
20 PRINT"           by M.P. Putnam  October 1984
22 PRINT
40 REM-- Reserve 2K buffer for LOGO program at top of memory.
42 REM-- Use line 44 if 48K memory, use line 46 if 24K
44 POKE129,175:POKE131,175:POKE133,175:WS=45056:REM 48K Mem
46 REM POKE129,79:POKE131,79:POKE133,79:WS=20480:REM 24K Mem
50 WE=WS+2000:WP=WS
55 REM-- IF C1P-MF USE TP=53743 & VM=1023 IN LINE 60
60 TP=54239:VS=53248:VM=2047:DR=18:TG=32:PF=0:TS=32
1000 REM----Input a line and break into 3 fields
1001 REM----P$(1)=line#  P$(2)=command  P$(3)=operand
1005 INPUT"<";A$
1010 P$(1)="":P$(2)="":P$(3)="":P=1
1020 GOSUB9000:IFPT=0THENP$(1)=P$(1)+P$:GOTO1020
1030 P=P-1
1040 GOSUB9000:IFPT=1THENP$(2)=P$(2)+P$:GOTO1040
1045 IFLEFT$(P$(2),2)="<?"THENP$(2)="":P=3:GOTO1020
1050 P=P-1
1060 GOSUB9000:IFPT=0THENP$(3)=P$(3)+P$:GOTO1060
1100 REM----Handle line nbr field
1110 IFP$(1)=""THENWP=WS:PL=0:GOTO1200
1120 PL=VAL(P$(1))
1130 IFPL>1000ORPL=0THENPRINT"ERROR:"PL"is bad line number":GOTO1000
1140 WP=WS+PL+PL
1200 REM----Handle command field
1210 IFP$(2)=""THENPOKEWP,0:GOTO1000
1220 P$=LEFT$(P$(2),2)
1230 IFP$="FO"THENPOKEWP,1:GOTO1300
1232 IFP$="BA"THENPOKEWP,2:GOTO1300
1234 IFP$="RI"THENPOKEWP,3:GOTO1300
1236 IFP$="LE"THENPOKEWP,4:GOTO1300
1242 IFP$="SE"THENPOKEWP,5:GOTO1300
1244 IFP$="CL"THENPOKEWP,6:GOTO1300
1246 IFP$="ZA"THENPOKEWP,7:GOTO1300
1248 P$=LEFT$(P$(2),4)
1250 IFP$="PENU"THENPOKEWP,8:GOTO1300
1252 IFP$="PEND"THENPOKEWP,9:GOTO1300
1260 P$=LEFT$(P$,2)
1262 IFP$="BI"THENPOKEWP,10:GOTO1300
1264 IFP$="SM"THENPOKEWP,11:GOTO1300
1270 IFP$="PO"THENPOKEWP,100:GOTO1300
1272 IFP$="EN"THENPOKEWP,101:GOTO1300
1274 IFP$="RU"THENPOKEWP,102:ONABS(PL<>0)GOTO1300:GOTO1500
1276 IFP$="LO"THENPOKEWP,103:GOTO1300
1278 IFP$="SA"THENPOKEWP,104:GOTO1300
1280 IFP$="FR"THENPOKEWP,105:GOTO1300
1282 IFP$="UN"THENPOKEWP,106:GOTO1300
1284 IFP$="HE"THENPOKEWP,107:GOTO1300
1286 IFP$="BY"THEN GOSUB3650:END
1290 PRINT"ERROR: "P$(2)" is bad command":POKEWP,0:GOTO1000
1300 REM----Handle operand value
1310 PO=VAL(P$(3))
1320 IFPO<256THEN1330
1322 IFP$="PO"THENPO=POAND255:GOTO1330
```

Continued

LETTERS

ED:

Rick Trethewey's Programming Class, Part VII, page 2 of the January 1985 issue, has a language problem that leaves the wrong interpretation of the 6502 Shift instructions. If you ignore the Carry Flag as Rick states in the middle column paragraph under the illustrations, then the result of shifting to the right divides the value by two, and shifting to the left multiplies the value by two. Rick knew what he meant, but "decrease the value" is another way of saying subtract, not divide. Similarly, "increase the value by two" translates to add two, not multiply by two as he really wants. The remaining two sentences in that paragraph show that he does indeed understand what he is trying to teach about bit shift instructions, and consistent use of correct mathematical translations of the vernacular by him will help his students learn the concepts with confidence.

Next, on page 4 of the same issue, L. Z. Jankowski gives a nice but time consuming PRINT USING for BASIC. It is nice because (at least on inspection) it appears to work. Since it is in BASIC, it is time consuming and gobbles, not eats, string workspace. For those of us who have both 65D V3.2 and V3.3, and prefer to work in V3.2 because V3.3 is so inordinately cumbersome and SLOW, it is not too difficult to steal the PRINT USING routines from your 3.3 disks and put them on your 3.2 disks. In short, the 3.2 ARCTAN routine is replaced by part of the 3.3 PRINT USING routine that resides in the same place; the balance of the 3.3 routines are loaded into page zero and swapped into the page at $2F79. In use pages zero and one are swapped regularly with the pages at $2F79 and $3079. Thus a part of the PRINT USING will be loaded at $0000 but will be used at $2F79. I have put these routines on my 3.2 disks, and have added another useful item. If your BASIC line states

PRINT USING "#####.##$";A

in other words has a dollar sign at the end of whatever pound sign combination you otherwise need, then the printout has a $ prefixed to the numeric value and adjacent

```
1326 PRINT"ERROR:"PO"is bad value":POKEWP,0:GOTO1000
1330 POKEWP+1,PO
1340 IFPL=0THENGOSUB2000
1400 GOTO1000
1500 REM----Run command
1505 TP=54239:DR=18:TG=32:PF=0:TS=32
1510 PL=VAL(P$(3)):IFPL=0THENPL=1
1520 IFPEEK(WS+PL+PL)<>0THENGOSUB2000
1530 PL=PL+1:IFPL<1001THEN1520
1540 GOTO1000
2000 REM---Sub: Perform command at line PL
2010 WP=WS+PL+PL:PC=PEEK(WP):PO=PEEK(WP+1)
2020 IFPC<100THENONPC   GOTO 3010,3010,3030,3040,3050
2022 IFPC<100THENONPC-5 GOTO 3060,3070,3080,3090,3100
2024 IFPC<100THENONPC-10GOTO 3110
2030 IFPC>99THENON PC-99 GOTO 3500,3600,3610,3620,3630
2032 IFPC>99THENON PC-104GOTO 3640,3650,3660
2050 RETURN
3010 REM---Forward or Backward command
3011 IFDR=16THENTC=-64:GOTO3020
3012 IFDR=18THENTC=1:GOTO3020
3013 IFDR=20THENTC=64:GOTO3020
3014 IFDR=22THENTC=-1:GOTO3020
3015 IFDR=17THENTC=-63:GOTO3020
3016 IFDR=19THENTC=+65:GOTO3020
3017 IFDR=21THENTC=+63:GOTO3020
3018 IFDR=23THENTC=-65:GOTO3020
3020 IFPC=2THENTC=-TC
3021 GOSUB4000:RETURN
3030 REM---Right command
3032 DR=DR+INT(PO/45+.5):IFDR>23THENDR=DR-8
3034 POKETP,DR:RETURN
3040 REM---Left command
3042 DR=DR-INT(PO/45+.5):IFDR<16THENDR=DR+8
3044 POKETP,DR:RETURN
3050 REM---Set pen command
3052 TG=PO:RETURN
3060 REM---Clear screen
3062 Y=PEEK(9761):POKE9761,207
3064 FORX=1TO30:PRINT:NEXT
3066 POKE9761,Y
3068 POKETP,DR:TS=32:RETURN
3070 REM---Zap (clear LOGO prog)
3072 INPUT"Do you really want to ZAP";P$:IFP$<>"Y"THEN RETURN
3078 FORX=WSTOWE:POKEX,0:NEXT:RETURN
3080 REM---Pen up
3082 PF=0:RETURN
3090 REM---Pen down
3092 PF=1:RETURN
3100 REM---Big Characters
3102 POKE56832,0:RETURN
3110 REM---Small Characters
3112 POKE56832,1:RETURN
3500 REM---PO Printout command
3501 PAGE=0:IFPL=0THENPO=VAL(P$(3))
3502 IFPO=0THENPO=1
3504 FOR WP=WS+2*PO TOWESTEP2:IFPEEK(WP)=0THENNEXTWP:GOTO1000
3506 PRINT(WP-WS)/2;:PC=PEEK(WP):PO=PEEK(WP+1)
3510 IFPC=1 THENPRINT"FORWARD";:GOTO3590
3512 IFPC=2 THENPRINT"BACK";:GOTO3590
3514 IFPC=3 THENPRINT"RIGHT";:GOTO3590
3516 IFPC=4 THENPRINT"LEFT";:GOTO3590
3518 IFPC=5 THENPRINT"SETPEN";:GOTO3590
3520 IFPC=6 THENPRINT"CLEARSCREEN":GOTO3595
3522 IFPC=7 THENPRINT"ZAP":GOTO3595
3524 IFPC=8 THENPRINT"PENUP":GOTO3595
3526 IFPC=9 THENPRINT"PENDOWN":GOTO3595
3528 IFPC=10THENPRINT"BIGCHAR":GOTO3595
3530 IFPC=11THENPRINT"SMALLCHAR":GOTO3595
3560 IFPC=100 THENPRINT"PO";
3562 IFPC=101 THENPRINT"END":GOTO3595
3564 IFPC=102 THENPRINT"RUN";
3566 IFPC=103 THENPRINT"LOAD";
3568 IFPC=104 THENPRINT"SAVE";
3570 IFPC=105 THENPRINT"FREEZE":GOTO3595
3572 IFPC=106 THENPRINT"UNFREEZE":GOTO3595
3590 PRINTPO
3595 PAGE=PAGE+1:IFPAGE<24THENNEXTWP:GOTO1000
3597 PAGE=0:INPUT"Continue";P$:IFLEFT$(P$,1)<>"Y"THENWP=WE
3598 NEXTWP:GOTO1000
3600 REM---END during LOGO run
3602 PL=1000
3604 RETURN
3610 REM---RUN during LOGO run
3612 PL=ABS(PO-1)
3614 RETURN
3620 REM---LOAD command (If 24K, use 5000 instead of B000)
3622 DISK!"CA B000="+RIGHT$(STR$(PO+29),2)+",1"
3624 RETURN
3630 REM---SAVE command (If 24K, use 5000 instead of B000)
3632 DISK!"SA "+RIGHT$(STR$(PO+29),2)+",1=B000/8"
3634 RETURN
3640 REM---Freeze Video Screen
3642 POKE9761,213
3644 RETURN
3650 REM---Unfreeze Video Screen
3652 POKE9761,207
3654 RETURN
3660 REM---Help command
3662 GOSUB 10000
3664 RETURN
4000 REM---Sub: Move turtle PO times
4010 FORT1=1TOPO
4020 IFPFTHENPOKETP,TG:GOTO4040
4030 POKETP,TS
4040 TP=TP+TC:TP=VS+(TP-VSANDVM):TS=PEEK(TP):POKETP,DR
4050 NEXT:RETURN
9000 REM---Get char at position P and put it in P$
```

Continued

to the leftmost numeric character. In the above statement if A=53.4 then the output is $53.40, and if the value is negative, the $ is to the left of the negative sign. There is a little more to the project, such as redoing some pointers, so more is needed than just what I have said above.

My V3.2 stuff also include the screen handling routines sold by Software Consultants of Memphis. I purchased these before V3.3 came out, but found after I purchased V3.3 and rewrote several programs that V3.3 is slower in PRINT AT functions. Scrolling is far slower than the original V3.2 and also very flickery and thus hard on the eyes. By using V3.2 with the above mentioned screen routines along with some other mods, I have a much faster system for screen handling, plus can define and work in windows; with borders instantly drawn around the windows by the screen routines, accounting programs really look sharp!

Other V3.2 mods include the Stretch Manley routines, PEEK(65) February 1982, etc., that make the random file techniques much better. I have added a

DISK FIND,dev,STRING

to this random file routines which searches all of each record so quickly (not just a certain field in the records) that the entire buffer is searched in a little less than a second. Then Stretch Manley's functions make sure the next track is loaded PDQ

```
9010 REM    PT=1  --> P$ is alpha
9020 REM    PT=0  --> P$ is number
9030 REM    PT=-1 --> end of input
9040 P$=MID$(A$,P,1):P=P+1:IFP$=" "THEN9040
9050 PT=1:IFP$>="0"ANDP$<="9"THENPT=0
9060 IFP$=""THENPT=-1
9070 RETURN
10000 POKE56832,1
10001 PRINT
10005 PRINT"               ** Mini-Logo Program Commands **
10010 PRINT
10015 PRINT" CLearscreen..Clear video screen.
10020 PRINT" FOrward N....Move forward N positions (where 0<N<256).
10030 PRINT" BAck N.......Move backward N positions (where 0<N<256).
10040 PRINT" RIght N......Rotate right N degrees (where 0<N<181).  N is
10050 PRINT"              rounded to nearest 45 degrees.
10060 PRINT" LEft N.......Rotate left N degrees (rounded to nearest 45)
10080 PRINT" SEtpen N.....Set turtle's pen to OSI graphic character N
10090 PRINT"              where 0<N<256.
10110 PRINT" PENUp........Raise turtle's pen (disable writing).
10120 PRINT" PENDown......Lower turtle's pen (enable writing).
10130 PRINT" BIg char.....Select 32X32 screen (characters are big).
10150 PRINT" SMall char...Select 32X64 screen (characters are small).
10160 PRINT" ENd..........End Logo program.
10170 PRINT
10171 PRINT"--> Note: Capitol letters of command are short entry form
10300 PRINT
10310 INPUT" Continue";A$:IFLEFT$(A$,1)<>"Y"THENRETURN
10320 PRINT
10500 PRINT
10510 PRINT"         ** Mini-Logo Control Commands **
10520 PRINT
10525 PRINT" BYE..........End this session, return to Basic.
10530 PRINT" RUn N........Run Logo program starting with line N.  N is
10531 PRINT"              optional.
10540 PRINT" PO printout..List Logo program.
10560 PRINT" SAve N.......Save Logo program N. N can be 1-10 and will
10561 PRINT"              cause program to be saved at track N+29.
10562 PRINT"              Track must be initialized.
10570 PRINT" LOad N.......Load Logo program N.
10580 PRINT" FReeze.......Freeze screen.  Only 8 bottom lines will
10585 PRINT"              scroll.  Use when doing direct command
10587 PRINT"              sequences.
10590 PRINT" UNfreeze.....Unfreeze screen.  All lines scroll.
10620 PRINT" ZAp..........Clear Logo program workspace.
10630 PRINT" HELP.........Display Logo command descriptions.
10700 PRINT:RETURN
```

★ ★

and the search continues. When a match is found, pointers are left at the start of the record so that BASIC can pick up the record and plop it on the screen in really amazing time!! Contrast the V3.3 FIND which leaves the pointer set to the end of the field in which the match is found!? What can you possibly do with any speed with something so dumb? Besides that, my FIND, if unsuccessful doesn't give an error which has to be trapped, etc., etc., but comes back with a useful binary TRUE or FALSE. Oh well, enough for now. As you can see, I have little patience for slow routines in a machine such as our OSI's which can be very fast.

The February issue of PEEK(65) just arrived, so I will make a couple of comments before finishing this session. On page 10, Rick continues his Assembly Class which is quite interesting and very good. However, he is making a serious mistake that others who send in Assembler listings of any sort most often do, namely he is not using the same labels that are used in the 65D Bible, Software Consultants OS65D V3.2 Disassembly Manual. This disassembly is the best annotated disassembly that can be had anywhere, and I stressed the point a couple of years ago in a letter to you that all who write in to a common-exchange-of-information magazine such as PEEK(65) should be consistent for the good of us all by using only those labels. I realize that they no longer are in the OSI business, but I visited Larry Hinsley at his office in Memphis in December of 1983 and he still had a stack of those manuals around. He would be glad to sell his remaining stock, and I have no doubt that he would be glad to sell the text files for printing the whole manual to PEEK(65) so that all who don't have a copy can then purchase one from PEEK(65). How about that, ED?

On the subject of Disassemblies, LeRoy Erickson wrote in a letter to the ED a year or so ago that he had disassembled and annotated a large part of the OSI Assembler-Extended Monitor and WP2. I wrote to him twice requesting he sell me a copy of same and received no response. I gave up and went ahead on my project, which works like a charm even though I was flying blind at the start. In short, I removed the cassette related commands from the Extended Monitor and put in other routines, one of which relates directly to Tom Berger's article on pages 2-8 of the February issue. This command does a "Q" disassembly but you specify a starting and ending location, and while you are seeing the disassembly on the screen, it is also filling Assembler workspace with an editable, useable source code listing. I didn't think of the HH idea that Tom uses, but it would be easy to change my code to make that happen, and also to make the .BYTE $FF appear instead of ??. In other words, I have done in machine code what Tom has done in his Pass 1 BASIC program. There is no problem about using this to disassemble the BASIC interpreter code, as I have another copy of the Extended Monitor that can be called to reside at $A700. I think his concept is really great for all the power it gives in disassembling, especially with the tables, etc., that he builds with the other passes. Will you be publishing his other programs soon? Enuf for now!

Paul Rainey
Villa Park, IL 60181

Paul:

We have made arrangements with Larry Hinsley. The disassembly manuals are now available through PEEK(65). For details see page 23 this issue. The answer to your last question is YES!

You have really aroused our curiosity with your other mods. Please do let us hear more.

Eddie

* * * * *

ED:

For those who are using WP-6502 Version 1.3 with OS-65D, here is a little more insight into the default parameters.

Loc.	Content	Application
$0225	$05	Default Printer
$0239	$01	Starting Pg. No.
$023D	$05	Paragraph Indent

On eight inch disks, WP-6502 is stored starting on Track #5. To make these changes permanent boot the system with a disk containing the Extended Monitor. When the Extended Monitor is active, insert the WP-6502 disk in the drive and enter "!CA 4200=05,1". Now the first track of WP-6502 is stored in memory starting at location $4200. This is an offset of exactly $4000 from where this code is normally stored. Using the "@" function, make whatever changes are desired. Save the corrected code back to the disk with "LSA 05,1=4200/B". That's all there is to it. I set the printer value to $01 for the serial port, the page number to $00 for no page numbers and the paragraph indent to $00 for no indenting. Hope this is useful to some of the folks using this fine word processor.

Harry B. Pye
Lansdale, PA 19446

OS-U CRASH BUG FIX

Several months ago, users of Gander Software's "The Data System" received a warning notice about possible crashed hard disk files. The culprit appears to have been found in OS-U V1.42 and later. Here is the story, but, as this is a Stop Press, you will have to wait until next month for the code for the fix.

Gander says, "After several months of "chasing" it around, we are virtually certain that we have located a bug in OS-65U's V1.42 and later COPYFI programs when used with multiple hard disks (we have found no evidence of a bug in other uses).

"SYMPTOM: When COPYFIng files or systems from "E" to "F", and where the "F" is a mirror of "E" (in other words, a backup), you can end up with trashed files on both the FROM and the TO devices.

"CAUSE: Failure of COPYFI's machine code copy routine to flush and print the last buffer sector read, writing it back to the FROM device. Since COPYFI can pull fractional sectors, but writes back whole sectors, the write to the FROM device will trash files. The TO device just doesn't get the information from that sector written to it. However, the next time you again back up "E" to "F", some of the trashed files from "E" will be copied to "F".

"We at first thought it was a problem with DBI's SCSI driver, used with a SCSI host adapter to drive two storage devices, "E" and "F". We even notified our licensees of that, and told them that it was O.K. to use COPYFI on OSI boxes. We were wrong.

"The folks at DBI very kindly spent considerable time helping us look for the source of the problem, and, once found, fixed the machine code. We added one other small safeguard.

"RESULT: COPYFI now works properly. Isotron, Inc. has been alerted. Registered Gander licensees of The Data System, who anticipate using their copy of TDS with multiple hard disks, can return their master disks to Gander for a fixed copy of COPYFI for a $25.00 handling charge. Until then, they shouldn't use COPYFI (Menu Selection 81).

AD$

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: